Guru Guide
Nobody goes deeper into technology

RASPBERRY Pi PROJECTS 2015

FULLY REVISED FOR THE Pi 2

EXPAND YOUR KNOWLEDGE AND BECOME A Pi GURU

• RASPBIAN • MINECRAFT
• PYTHON • GPIO HACKING

180 PAGES OF TUTORIALS

MASTER NEW SKILLS AND BUILD EXCITING Pi GADGETS

Future

RASPBERRY Pi PROJECTS 2015

RASPBERRY Pi PROJECTS 2015

EDITORIAL TEAM

ART EDITOR
Efrain Hernandez-Mendoza

EDITOR
Neil Mohr

EDITOR-IN-CHIEF
Graham Barlow

CONTRIBUTORS
Jonni Bidwell, Neil Bothwick, Kent Elchuk, Russel Hughes, Les "Pi Man" Pounder, Mayank "The Machine" Sharma.

MANAGEMENT

CONTENT & MARKETING DIRECTOR
Nial Ferguson

HEAD OF CONTENT & MARKETING, TECH
Nick Merritt

GROUP EDITOR-IN-CHIEF
Paul Newman

GROUP ART DIRECTOR
Steve Gotobed

MARKETING

MARKETING MANAGER
Richard Stephens

PRINT & PRODUCTION

PRODUCTION MANAGER
Mark Constance

PRODUCTION CONTROLLER
Marie Quilter

CIRCULATION

TRADE MARKETING MANAGER
Juliette Winyard
Phone +44(0)7551 150984

LICENSING

LICENSING & SYNDICATION DIRECTOR
Regina Erak
regina.erak@futurenet.com
Phone +44(0)1225 442244
Fax +44 (0)1225 732275

SUBSCRIPTIONS

UK reader order line & enquiries: 0844 848 2852
Overseas reader order line & enquiries: +44 (0)1604 251045
Online enquiries: www.myfavouritemagazines.co.uk

PRINTED IN THE UK BY

William Gibbons on behalf of Future.
Distributed in the UK by Seymour Distribution Ltd,
2 East Poultry Avenue, London EC1A 9PT. Phone: 020 7429 4000

Future Publishing Limited
Quay House, The Ambury, Bath, BA1 1UA, UK www.futureplc.com
www.myfavouritemagazines.co.uk
Phone +44 (0)1225 442244 Fax +44 (0)1225 732275

Contents

RASPBERRY Pi PROJECTS 2015

It's time you got more from your amazing Raspberry Pi, we're here to guide you into a world of fun projects and engaging learning.

First steps

If you're not already, we'll get you up and running with the Pi in no time.

10 **Install Raspbian**

17 **Get connected**

22 **Troubleshoot booting**

24 **Discover the command line**

28 **Using storage**

30 **The history of the Pi**

Projects

Let's dive straight into our Pi projects! There is a ton of fun to be had here.

34 **Kodi media centre**

40 **Whatsapp on the Pi**

44 **Install Ubuntu 15.04**

48 **Create a Tor box**

52 **Using OwnCloud**

56 **Create an internet radio**

60 **Make a Pi home server**

64 **Mathematica Pi**

68 **Multi-Pi processing**

72 **Minecraft Pi hacking**

74 **Build a Minecraft trebuchet**

78 **Make 2048 in Minecraft**

Linux

Linux is the software that makes the Pi work. So let's explore what it can do.

84 **Discover Linux**

92 **Welcome to the Terminal**

94 **Installing software**

96 **Core terminal commands**

98 **Using Raspbian packages**

100 **Linux help**

103 **Make your own Pi distro**

Hardware

You have to run your apps on something! Hardware projects you'll just love.

110 **Build a network drive**

114 **Game on an Xbox controller**

118 **Create a networked printer**

123 **Using the Pi camera**

128 **Get started with the GPIO**

130 **Using the Raspberry Pi 2**

134 **Build a CCTV with the Pi**

138 **3D graphics with the GPU**

142 **The Raspberry Pi 2**

144 **The Raspberry Pi Model B+**

146 **Pi2Go robotics**

147 **Pi hardware reviews**

Coding

To get the most from the Pi you need to code, here's our intro to Python.

154 **Get started with Python 2.4**

158 **What are types?**

160 **Using types to sort things**

162 **Functions for the win!**

164 **Accessing files**

168 **Get into Python 3.0**

170 **Creating fast-sort routines**

174 **AstroPi and Minecraft**

RASPBERRY Pi PROJECTS 2015

First Steps

Install Raspbian 10
Connecting things 17
Troubleshoot booting 22
Discover the command line 24
Using storage 28
The history of the Pi 30

Raspbian: Easy

The Windows operating system offers perhaps the easiest tools for writing your Raspbian image to an SD card.

Step-by-step: Windows

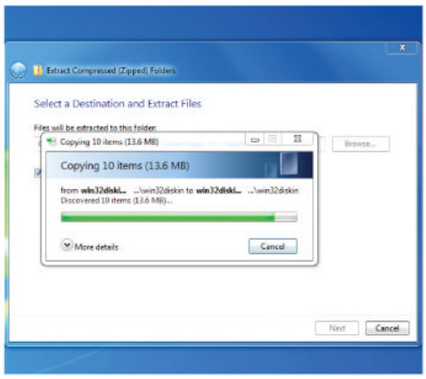
1 Download the tools

While Microsoft Windows doesn't have that much in common with the version of Linux that runs on the Raspberry Pi, it doesn't mean Windows users are at a disadvantage. This is because the types of applications that run on Windows are just as diverse, and it's an operating system that can lend itself to so many different uses, that the do-everything attitude behind Windows is just as applicable to Linux.

You can even find a Linux-like command line if you look close enough, and many pieces of open source software have been built to run on Windows. So the two worlds aren't that far apart. Windows users also benefit from having the easiest and least-risk installation routine, thanks to a piece of open source software called *Win32 Disk Imager*. Your first step should be to download a copy of this from the following URL:

http://sourceforge.net/projects/win32diskimager

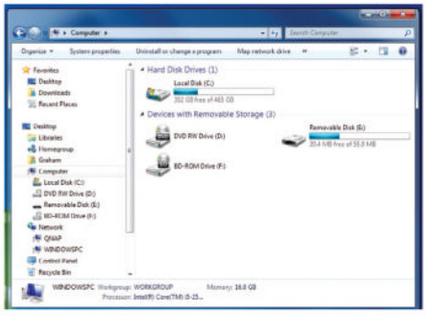
It doesn't need to be installed, because the download is a ZIP file which, when uncompressed with a right-click > Extract All, contains everything needed to run the application directly. It should also go without saying that you'll need a copy of the Raspbian image file for writing to your SD card. But this method will work for any Raspberry Pi operating system you might want to install,



We used an open source tool, called *Win32 Disk Imager* to easily write the image to the SD card.

2 Check your USB storage

But before we run the image writing tool, we need to make sure the SD card has an identifiable volume name and contains no data you want to keep. Many Windows PCs include an SD card reader built in to the case, so you might want to try using this first. However, there have been quite a number of problems reported when users attempt to use these to create a bootable version of Raspbian. This is something to be aware of if you run into problems. We've always had great results from using an inexpensive external USB SD card reader, which is what we're using here. With this inserted, you should find the device appears in your Computer overview from the Windows file manager. It's usually labelled as Removable Disk, and to the right of this you'll see a drive letter. In our example, it's (E:). You need to remember this. You should also open up the drive from *Explorer* and make sure there's nothing on there you want to keep, because we'll be overwriting everything with the Raspberry Pi operating system. Which is exactly what we're going to be doing in the next step.

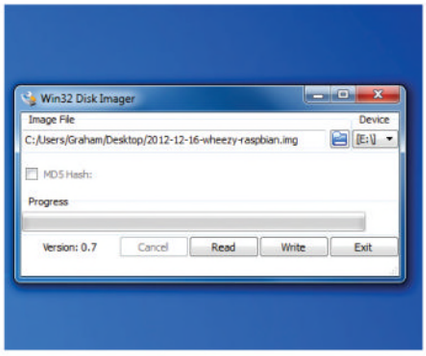


Windows displays the size of the mounted partition and not the entire storage device.

3 Launch Win32 Disk Imager

It's now time to launch the executable we unzipped from the download in step one. Windows will probably warn you that there's a risk to running something downloaded from the internet. If you want to be certain your files aren't infected with some kind of virus, you ought to scan it with your virus checker. Make the most of this, because after you've got used to Linux, you won't have to worry about your files getting infected again.

When the main application window opens, you'll notice a very sparse interface that isn't that clear. There are two buttons, one for locating the image file and another – just to the right of this – is for selecting the device where you're going to write code from the image file. Click on the small folder icon and use the file requester that appears to locate your Raspbian image – the requester is already configured to show only files with an IMG file type, so if you can't find it, this will be the problem. Secondly, use the tiny 'Device' button to select the destination. As all data on the destination is going to be overwritten, it's important to get this right, and it's the reason why we checked for the drive letter in the previous step. You need to make sure it's the same. *Win32 Disk Imager* normally guesses this correctly, but it's worth checking.



The user interface may be sparse, but it's got all the functionality we need.

installation

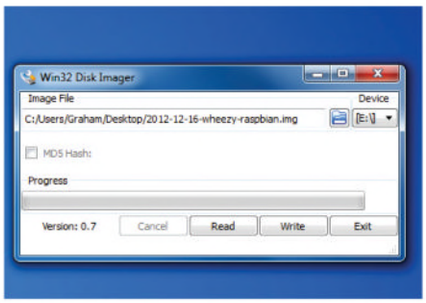
4 Write the data

When you've double-checked everything is configured correctly, click on the 'Write' button. A warning will pop up to say that writing to a physical device can corrupt the device, but you can safely ignore this.

Your external storage is going to be written to at its lowest level, changing the partition table and some of the formatting, which is why this warning appears. But this low-level is also necessary to get the RPi booting.

After saying 'Yes', the first thing you should check for is any access LED on your USB device. If this starts flickering, you've got the correct device and you can go and make yourself a cup of tea while the data is written. If not, you need to make triple sure you've got the correct device because this kind of low-level copy on an external hard drive could make it unusable.

If you need to stop to check, hit the 'Cancel' button before the process gets any further. Writing the image can take around 20 minutes, depending on your hardware, and you'll see the progress indicator in *Win32 Disk Imager* update as well as a small text field showing you how many megabytes of data are being written per second.

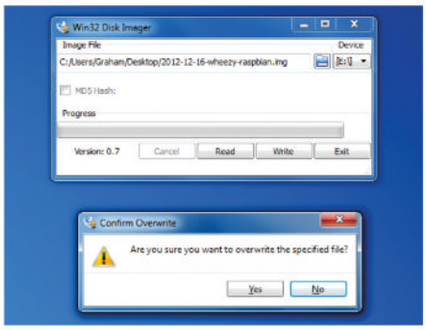


While copying the data, both the progress bar and the data transfer speed update to show you what's happening.

5 Checking the installation

Before putting your freshly made SD card into your Raspberry Pi, it's worth having a quick check to make sure the write process worked as expected. It's also a good way of learning a little about how the SD card has been formatted. You could look again at your computer's device overview to check that the files stored on your SD card aren't the same as before. But you won't be able to get any further information, because one of the partitions on your SD card is now formatted with a Linux filesystem – making it unreadable in Windows. The solution is to use one of Windows' built-in administrator tools, and it can be found by searching for **Create And Format Disk Partitions** within the desktop.

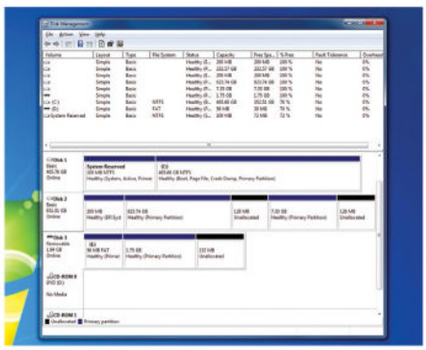
Selecting the top result will open the *Disk Management* application, and within its main window you can see details of all the storage connected to your machine. Locate your SD card volume – ours was E – and look at its partitions in the table below. You should see three – one for booting, which is a 56MB FAT partition, one holding the Linux filesystem, and the remainder of the space marked as unallocated. If you see these three, everything has gone well.

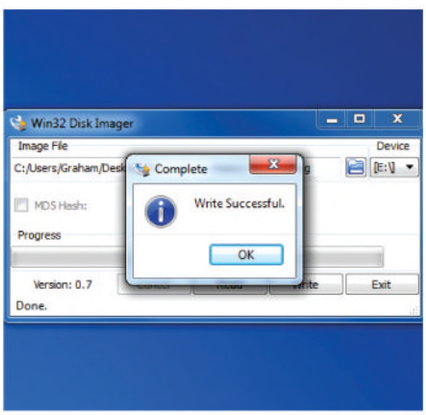


The Windows partitioning tool can be a good source of information on your hardware.

6 Back up your Raspberry Pi

Another excellent feature of the *Win32 Disk Imager* tool is that it can do the reverse of what we just did. It can take an SD card (or any USB storage device) and turn it into an image file. This is a great way of taking a snapshot of your Pi's operating system. If, for instance, you spend some time modifying your setup, adding packages and adjusting configuration files, you can come back to your Windows desktop and use the 'Read' option in the main window to copy the contents of the storage device to a single file. You can then use this single file as a new start point if you need to format another SD card, or if you want to give someone else a hand with your own configuration. It's also great for backup, because the Raspberry Pi can be fickle when it comes to power provision, and random resets can occur, possibly corrupting data on the card. If you've got a backup, you're safe.





***Win32 Disk Imager* can be used to turn the contents of your SD card into an image file.**

Raspbian: Easy

There's a quick and easy method for Apple users that doesn't involve typing any complicated commands.

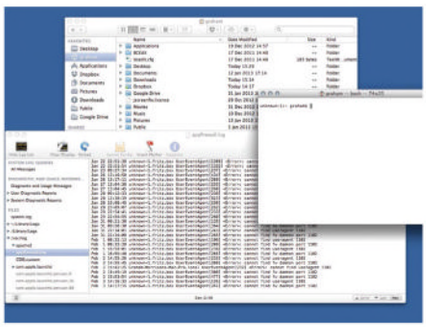
Step-by-step: Apple

1 Be prepared

Despite its air of point-and-click friendliness, and Apple's tendency to precede its own applications with the letter 'i', its OS X operating system is actually a lot like Linux. Both OS X and Linux have a similar system at their core, based on an old-school multi-user operating system called UNIX, and many of the tools and utilities that make OS X useful are the same as those you'll find within Linux.

Consequently, this gives you something of an advantage when it comes to using the Raspberry Pi. If you've ever used the command line from OS X, for example, you'll find exactly the same command line, with almost all the same commands, on the Raspberry Pi. Similar, too, are the concepts of user accounts and home directories, network printing and file sharing, and you'll find that the Raspberry Pi will work almost as well with OS X as it does with Linux. You can easily connect to it from the command line, share a virtual desktop environment and configure your Pi remotely, all without installing a single piece of additional software on your Mac.

But before you get to that stage, you will have to first install Raspbian on to your Pi's memory card.

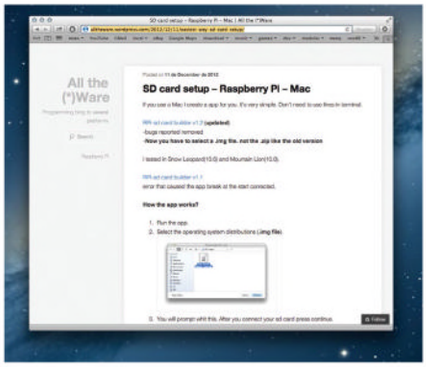


OS X bundles tools that you'll also find on a Raspberry Pi.

2 Download the tools

From OS X, like Linux, the standard way of installing the Raspbian operating system on to your SD card is from the command line. If you want to try this method, follow the instructions for Linux but replace the device name with those from OS X. But there's a safer, easier option, and that's using a graphical tool developed to do exactly the same job. This tool is called the *RPi-sd card builder*, and at the time of writing, version 1.2 is available from: **http://alltheware.wordpress.com/2012/12/11/easiest-way-sd-card-setup**. Download the builder to a local folder, and also make sure you've got hold of the latest Raspbian image, as the card builder tool needs this in the first step. The latest version of the image is always available from: **www.raspberrypi.org/downloads**.

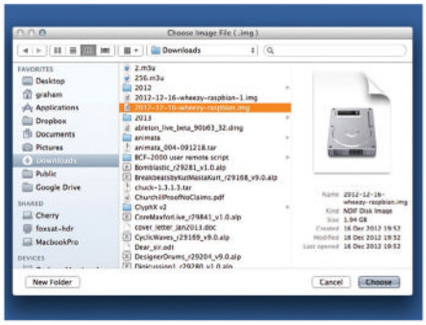
Make sure you grab the version labelled 'Debian Wheezy' and not the version labelled 'soft-float'. Clicking on the download link will take you to another page where hopefully the download will automatically start. If it doesn't, try another link listed as 'mirror' from the same page. These mirrors are different servers on the internet that host the same file so that the download burden is shared. The download is usually around 500MB, and should only take a few minutes with a good connection, although the speed is often more limited by the mirror rather than your connection.



To save you from resorting to the command line so early on, the *Rpi-sd card builder* will do the same job with just a couple of clicks.

3 Run the builder

The download is actually a 'zipped' version of the image. Zipping files in this way reduces their size by cutting out the duplication within the file. But it also means a file needs to be unzipped before it's useful. On OS X, this should happen automatically after the download has completed, leaving you with the raw file – in our case, this is called **2015-05-05-wheezy-raspbian.img**, but the date element will change depending on when the Raspbian image was constructed. If you need to unzip the file manually, just double-click it. The zip file will be replaced with the image. Now it's time to run the *RPi-sd card builder*. You should find this in your Download folder, complete with a Raspberry Pi application icon. Run the application and click through the warning/disclaimer that this file was downloaded from the dangerous wilderness that is the internet. Milliseconds later, without even a splash screen, you'll be presented with a file requester asking for the location of the image file. After successfully completing the previous step, you should have no difficulty locating this file and giving it to the requester. Press the 'Choose' button with the image file selected to progress.



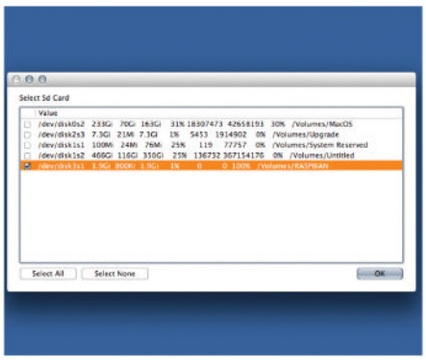
Use the file requester to locate the Raspbian image file and click on 'Choose'.

installation

4 Choose the card

The *RPi-sd card builder* will now complain that it can't detect your SD card whether it's connected or not. We use a standard USB SD card reader, and with the SD card for your Raspberry Pi inserted into the reader, connect the reader to a spare USB port – make sure your SD card is write-enabled, if it has this feature. This is usually a small sliding switch on the side of the card to protect valuable data from being overwritten. Without wanting to state the obvious, you need to make sure there's nothing on the card you want to keep, because it will all be overwritten. With the SD card reader connected, you'll see the device automatically mounted on to your desktop.

This is an opportunity to make a final check before starting the writing procedure. In the *RPi-sd card* application, click on 'Continue'. The following window lists all the storage devices connected to your system and you need to select the device that corresponds to your SD card. But first click 'Select None'. This is important, because selecting the wrong device could be disastrous and you will lose important data. Make sure the device you choose has the same volume name (on the far right) as the name for the new removable storage on your desktop, and that the capacity column (second from the left) is the same as the capacity your card is capable of. When you are sure, click 'OK'.

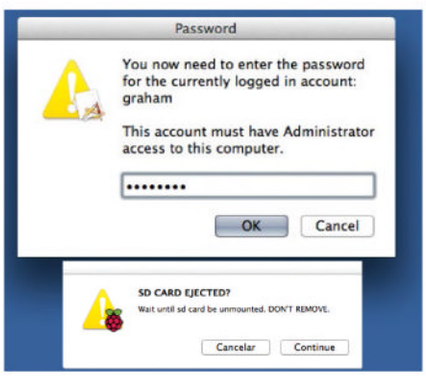


› It's vital you only select the device that corresponds to your SD card. Triple-check the volume name to make sure.

5 Write Raspbian

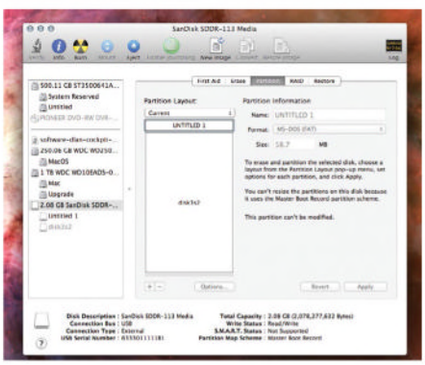
You will now be asked to enter your password. You need to make sure your current user has Administrator privileges, which is another concept that's used by Linux. Most default OS X users are also administrators, so just enter your password. Another window will pop up saying you need to wait for the device to be unmounted. Devices need to be inaccessible from the desktop for them to be written to at the low level Raspbian requires, so wait for the desktop icon for your SD card to disappear before clicking 'Continue'. If you've chosen the correct device, you should see its read/write LED flicker as data is being written to the card. You can now relax – you got the correct device, and you'll need to wait a while for the data to be written. On our machine, this took about 15 minutes. The progress indicator is hidden behind the small rotating gear hidden within the menu at the top of the screen.

› You will need to enter your password to give the application enough privileges to write to the raw device.



6 Check the installation

When the process has completed, your card should be ready for its first live test within your Raspberry Pi. But before you do, it's worth taking a look at the card's layout from your desktop. This way you can be sure the writing procedure has worked and avoid any unnecessary troubleshooting when it comes to booting up the Pi. Firstly, after the process has finished, you should see a new drive mounted on to your desktop. This will contain only about 18MB of data, and Finder will claim there's still only 41MB free. But this is because you're only looking at the boot partition, as this is the only partition OS X can now read. The other partition is formatted with a Linux filesystem, and if you want to see it, launch Apple's excellent *Disk Utility* application. This tool allows you to format and re-partition drives, and if you select the remote storage drive from the left panel and then select 'Partition' from the tabbed functions on the left, you'll see a vertical layout of the partitions on the drive. There should be two – the UNTITLED boot partition, plus the other, containing Linux. There should also be some unused space, which you'll be able to take advantage of after you've booted into Raspbian and used its configuration tool for resizing Linux into any unused space.



› Use *Disk Utility* to check the validity of the installation before you try your card.

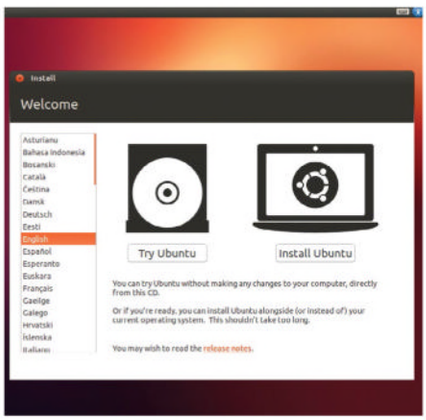
Raspbian: Easy

Installing Linux from Linux is perhaps the safest option, and you'll learn some important concepts along the way.

Step-by-step: Linux

1 Share the features

If you've never used Linux before, don't worry. It's just as easy to use as any other operating system, and in many ways, it's easier. There are no drivers to chase and new applications are always installed through the Linux equivalent of an app store. And, as you're going to be installing and using Linux on your Raspberry Pi, it makes good sense to create your SD card from within a Linux environment. It doesn't make the installation any better, but it gives you a great opportunity to try it out before plugging in your Raspberry Pi. We recommend the Ubuntu distribution, as it's ideal for beginners, but these instructions will work for nearly any other version of Linux – replace the *Ubuntu Software Centre* with your package manager of choice and ignore the desktop specifics. Linux is also a good failsafe option, because it can be run from a live CD without installing anything. Just insert the CD and boot your machine from the optical drive. After a few moments, choose the 'Try Ubuntu' option from the menu, rather than 'Install'. This will take you to the Ubuntu desktop without needing to install anything on your machine.



Even without Linux installed, you can use a live CD to boot your machine into a Linux desktop.

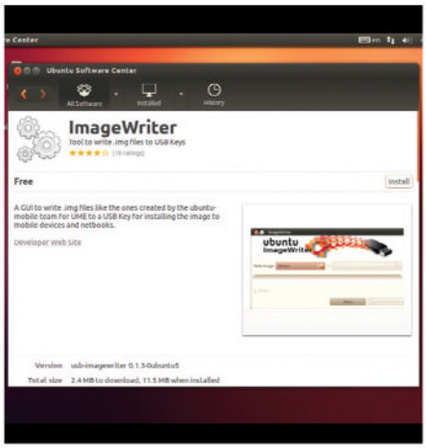
2 Be prepared

The one problem with using the live CD for an installation is that you won't be able to download the Raspbian image. There isn't enough RAM allocated for storage space on the desktop session, so you'll need to download the image on to some external storage (but not the SD card we're using for the Raspberry Pi). Users with Linux installed won't have to worry about this, and they can just download the latest image directly to their hard drive. With the image sorted, you should also check the state of your SD card. Insert this into a card reader and it should appear on your desktop with a window for each partition on the drive. All this data will be lost when we install Raspbian, so you need to make sure there's nothing you want to keep. As Ubuntu loads the contents of each partition, regardless of the way each partition is formatted, you can check every spare byte of your storage if you

When you insert a USB storage device into Ubuntu, it will display the contents of any partitions it finds.

3 Install ImageWriter

We're going to use a tool called *ImageWriter* as a graphical front-end for writing the Raspbian image. This can be installed from Ubuntu's *Software Centre* application, which can be launched by clicking on the basket icon in the launch bar. Search for 'imagewriter'. A single result should be returned. Double-click on this and the next screen will announce this is available from the 'universe' source. This is an additional repository for software, and it's not enabled by default, but you need to click on the 'Use This Source' button to access it. Wait for the progress button to finish updating the internal package list, then clear the search field and search for 'imagewriter' again. You should find that the package has been updated, and when you select it, an 'Install' button appears. Click on this and the package will be downloaded and installed automatically. You might wonder why this worked when you're using a live CD, but the answer is that there's enough room in the memory to install quite a few packages, just not enough to hold the entire Raspbian image. With *ImageWriter* installed and your SD card mounted, you're now ready for writing the Raspbian image to your card.



***ImageWriter* can be installed and run from an Ubuntu live CD, which means you don't even need Linux.**

installation

4 Write the Raspbian Image

ImageWriter needs to be launched with your SD card connected, otherwise it won't run and instead complain it can't find any storage. When the application window appears, you need give it one or two parameters. The first is the location of the Raspbian image you want written to the USB stick, and the second is the device you wish to write the image to. It's the second that's most important because if you've got more than one device connected – such as to read the image off an external drive while you write it to the SD card – the wrong selection could overwrite your data. Both Windows and OS X suffer from the same problem, but at least with Linux it will only let you choose an external USB storage device. It will also display the name of the manufacturer so you can be sure you've selected the correct device. When you're ready, click on 'Write to device.' If you've got the correct one, the activity LED for the SD card should start flickering to indicate data is being written. If not, 'Close' the ImageWriter window as soon as possible to halt the process.

The write process can take a while, as it depends on the speed of your storage and USB ports. Ours took 15 minutes, but the progress indicator kept us updated, and when complete it was time to test the new SD card.

› *ImageWriter* needs only the source image file and a place to write it, but it won't launch without a card inserted.

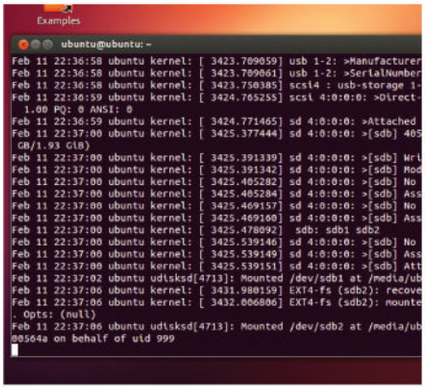
5 Test the SD card

Unlike with both Windows and OS X, Linux is the only operating system that can read both of the partitions created by the write process. The first is formatted with a Windows FAT filesystem and is almost 60MB in size. This is how the USB stick boots the Raspberry Pi, as this partition is read first before passing control on to the second partition. The second takes up the best part of 2GB and contains the root Linux filesystem. As both of these partitions will be mounted when you next insert the SD card into Ubuntu, you'll be able to take a closer look at the files both partitions contain. The Linux one will be very similar to the desktop version of Ubuntu you might be running, and this is because they're both derived from the same 'parent' distribution, called Debian. The home folder, for example, contains a user's own folder, where they can store their files and settings. Raspbian is pre-configured with only a single user, called 'pi', although this can be easily changed when you've got the distribution running, and you can see this folder and the files it contains when you click on 'Home'. When you've finished, unmount the device from the file manager and insert the card into your RPi.

› Linux can read Windows, Linux and OS X filesystems without needing to install any further files.

6 Failsafe Install

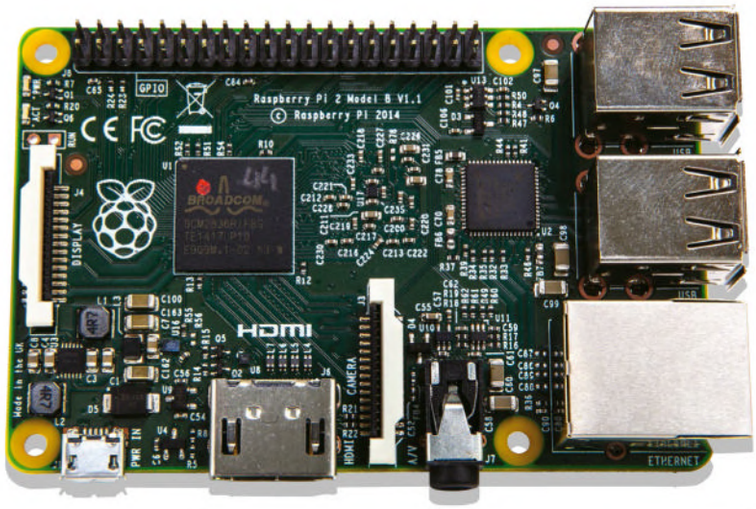
There's one other method for installing Raspbian on the SD card, and we want to cover it because it's useful as a fallback. But this method does make it easy to accidentally overwrite your data, so we'd only recommend it if nothing else works. This method involves the command line and the **dd** command. This takes a raw input and copies it – byte for byte – to another device. Get the destination device wrong, and you'll be overwriting a hard drive with your precious photos on it. To get the device correct, first disconnect your SD card and look for and launch **Terminal** from Ubuntu. This will open the interface to the famous Linux command line, but it's really not all that difficult. Now type **tail -f /var/log/syslog** and insert your SD card. What you're doing is displaying the output logs of the system, and you need to look for a line that looks like **sdb: sdb1**. This means the system has detected a new device and given it a node on your file system of **sdb** (**sdb1** is the first partition on **sdb**). There should be lots of other output as your Linux box attempts to read the filesystem and mount it. If it is mounted, unmount it from the GUI and then type **sudo dd bs=1M if=raspbian.img of=/dev/sdX**, replacing both the IMG filename and the **/dev/sdX** node with those of your specific configuration. The image will now be written to the SD card with not a GUI in sight.

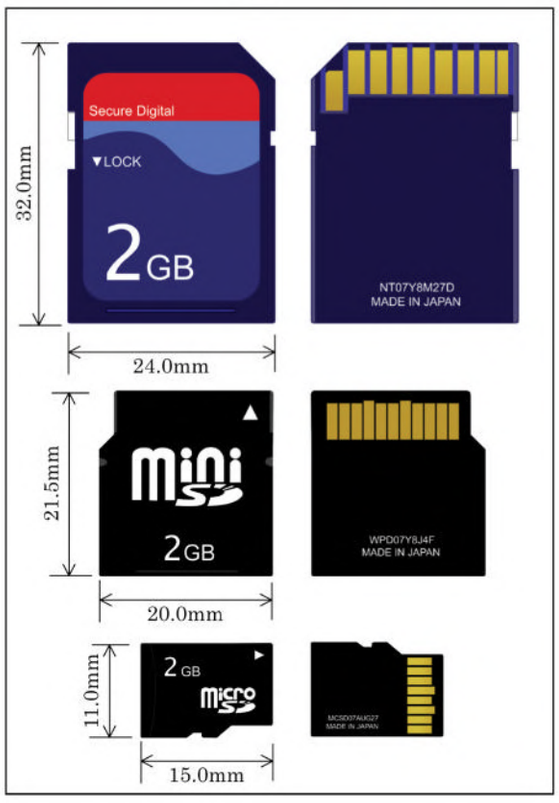


› You can cancel the dd command mid-way through by pressing [Ctrl] and [C] together.

Get connected: Peripherals

Our in-depth guide to picking the right peripherals, and which connections you need to make for the best performance.





The Raspberry Pi can be choosy when it comes to SD card compatibility, so make sure you buy one you know works and fits. NB: the right size is the one at the top! (Image: Tkgd2007 CC)

It might seem obvious, but before you can get started with the Raspberry Pi, you need to plug in lots of things. But getting those things right can make the whole process much easier, and problems that occur from the wrong connection being made, or an incompatible peripheral, can be difficult to track down. For that reason, while you should dive in and connect your Pi as soon as possible, it's worth looking into which hardware works best – especially if you're planning on buying it new.

The best place to start is with the SD card. This is because it's the most important peripheral you need to buy for your Raspberry Pi. These tiny rectangles with a chomped-off corner are for storing the OS, as they're the only device your Raspberry Pi can boot off. And especially in the beginning, it can take a hammering. Your SD card needs to withstand random reboots, being inserted and removed many times, and continually runs the risk inherent in being connected to an exposed circuit board with no flat base.

As a result, getting the right SD card can be a little tricky. You might not want to go for capacity, for example, as this will push the price up on a device that could break. But a certain amount of capacity is essential, because you need at the very minimum 2GB (gigabytes) to install the default Rasbpian operating system. Any space left over can be resized and used as storage for your own data and additional packages, so it's always worth getting more.

We'd recommend going for a 4GB device to start with, and if you need more storage, use a USB storage device, such as an external hard drive. These are cheaper for large storage, and if you use them for media or configuration files, you can keep the data intact when you overwrite the operating system on the SD card. It's also worth checking a card for compatibility before you spend your money, because there is a random array of devices that just won't work with the Raspberry Pi. You should also make sure your card is labelled SDHC (HC for high-capacity), rather than either SDSC, SDXC or SDIO, as these have been known to cause problems. The site to check is **http://elinux.org/RPi_SD_cards**. This lists all working and faulty cards detected by users – even some from major brands such as SanDisk and Kingston fail, so it's definitely worth checking or asking for known compatibility first.

If you're a complete beginner, you might want to consider an SD card with Raspbian pre-installed, as this side-steps the compatibility issue and the difficulty of getting the OS on to the SD card. But in reality, we've had very few problems with »

» SD cards we've bought from ordinary stores, so it's worth looking for a good deal or seeing if you've got any space from an old camera.

Power

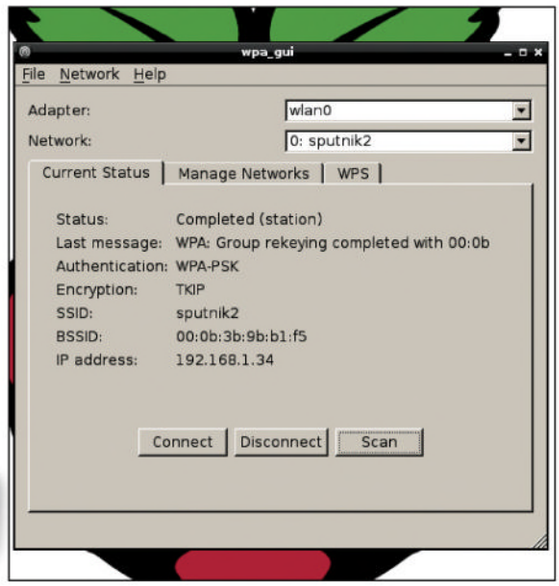
You might not think power should be that much of a consideration for your Pi, but it is. The problem is that if you don't get the power provision correct for your configuration, you can't easily tell whether any problems you experience are a result of the power supply or user error with the software. While writing a tutorial on using the Pi to grab television data, for example, we struggled for hours because we couldn't get our USB hardware to discover any channels. Everything looked perfect – there was no sign of any errors, but the problem was that the USB hub we used to connect those external devices wasn't powerful enough. You also need to make sure the power for the Raspberry Pi itself is in good condition, and for that reason we recommend powering the Pi solely from a micro-USB charger that's capable of at least 1.2A. Any more than 1.2A won't cause problems, but any less may generate unpredictable behaviour. You can find these kind of adaptors bundled with many modern mobile phones, and they can be purchased from any electronics store, but anything less than 1.2A may cause problems when you start putting your Pi under load.

For external USB devices, we also highly recommend using a decent USB 2 (HiSpeed) powered hub. This will connect to your Raspberry Pi with a standard B plug to standard A USB cable. Older USB 1 hubs will appear to work but we've experienced problems with hardware compatibility so they're not worth the risk. You also need to make sure the hub is powered separately, and not just from the Raspberry Pi. Many cheaper hubs will not include a PSU for self-powering, although they'll typically have an input on the side if you can find one that fits both the size and the power requirements. They all need to provide 5V, and the best will give 3.5A, which will be needed if you want to power many devices. The main problem is finding one where the adaptor fits your hub, so it's often easier to spend a little more on a hub that comes with its own PSU. Don't forget to connect everything through the hub, rather than directly to the Pi, as this is the only way to take advantage of the external power.

Networking

For installation and configuration, the Raspberry Pi needs to be connected to your home network. This is best done through the on-board Ethernet port, which is the large RJ45 connector on the side. Most internet connections at home go through a wireless hub that should also include several RJ45 connections for any wired devices you might want to connect – it could be a games console or a television receiver, for example. The Raspberry Pi connects in exactly the same way, and as long as your home hub is working, simply connecting an Ethernet cable between the two is all you need to do. If you can't access the hub, another option is to use an 'Ethernet

You need to consider how you power both the Raspberry Pi itself and any USB devices you connect to it.



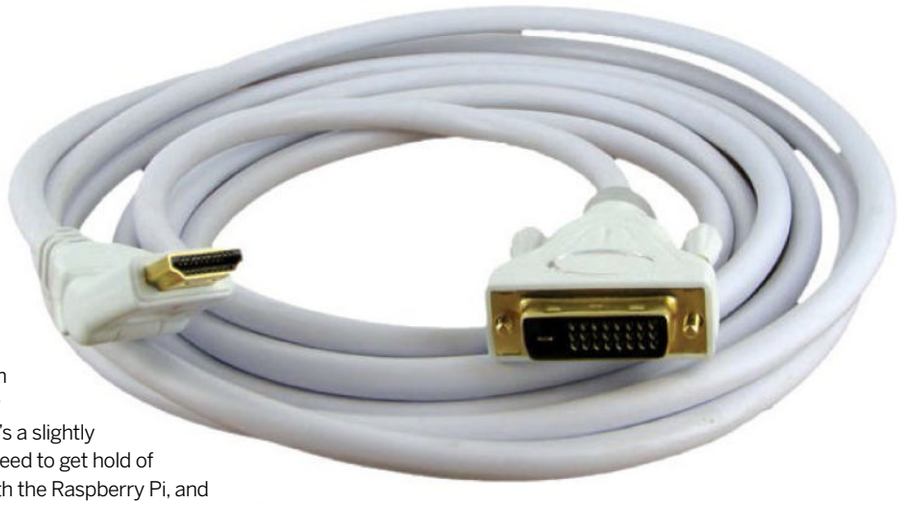
Ethernet connections are easy. But many USB wireless devices will also just work using the desktop Wi-Fi tool.

Get a case

Even though it won't affect the function of your Pi directly, another peripheral to consider is a case. By default, the Pi is fairly prone to knocks and bumps, and as there's very little circuit protection, moisture and metal surfaces can create a short-circuit and break the Raspberry Pi. For those reasons, it's worth putting your Pi in a case. And thanks to the DIY ethos behind the whole project, there's a case to suit every budget. At the free end of the scale, there are many everyday objects that can be turned into a case. Our favourite is one built from a compact cassette, but we've also seen some excellent examples built from Lego blocks, and nearly everyone must have a box of those handy. The blocks can be used to work around the ports on the Pi and protect the exposed parts of the circuit board. Clear bricks can even be used to ensure you can still see the status LEDs. If you're prepared to spend a little money, there are many colourful acrylic cases that cost around £5. Many also have the advantage of being mountable, which means that once your Pi is screwed into the case, you can then attach the case to anything else, whether that's the inside of an arcade machine or the rafters of your house. However, cooling should be an important consideration if you do attach your Pi to anything, because they can get hot when performing some heavy processing.

through the power' adaptor, often known as a powerline adaptor. One connects to a spare Ethernet port on your hub and into a spare power socket, while the other connects to a power socket closer to your Raspberry Pi, with an Ethernet cable connecting the Pi to your network.

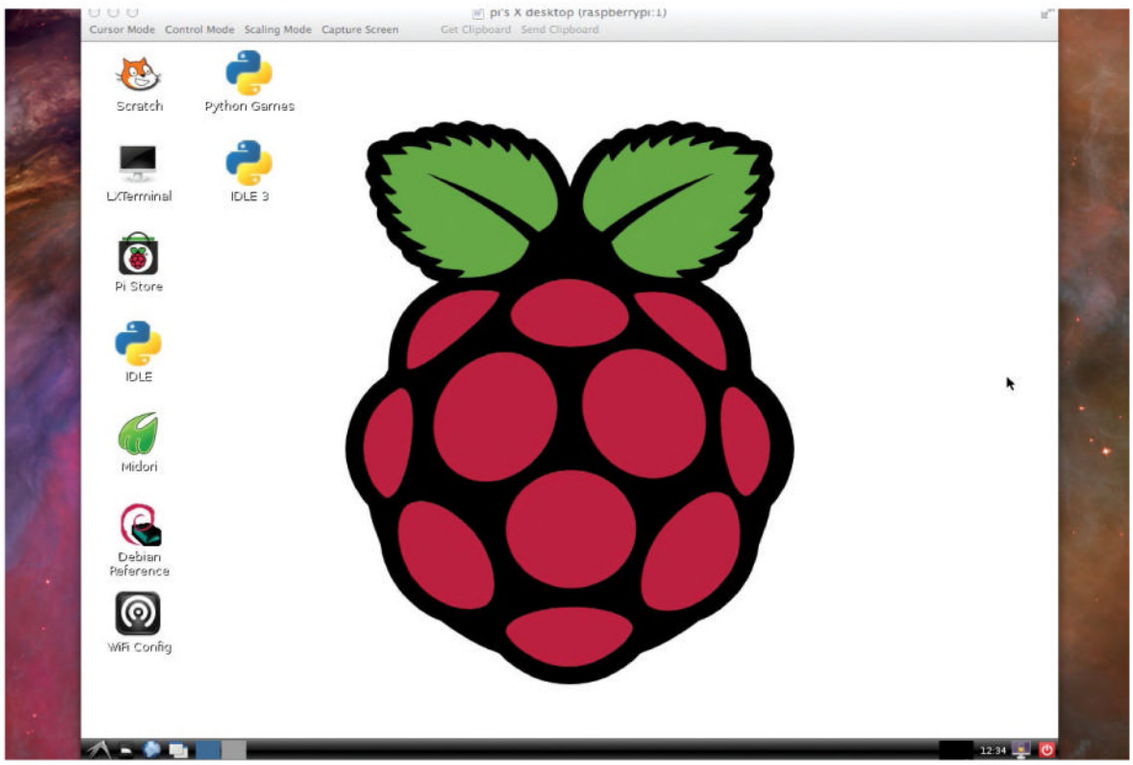
If you've got a model A without an Ethernet connection, or you'd rather connect to a wireless network, there's a slightly more convoluted process. You first need to get hold of a USB wireless device that works with the Raspberry Pi, and we'd recommend first checking the verified peripherals list at elinux.org (**http://elinux.org/RPi_VerifiedPeripherals**) and choose a device known to work out of the box. That way you won't have to install a driver without an internet connection. Also make sure you connect the wireless device to a powered hub rather than directly to the Pi, because wireless networking can draw considerable power from the USB bus. All you then need to do to configure a wireless connection is click on the 'WiFi config' button on the Raspbian desktop. If your device has been detected correctly, you'll see wlan0 as the adaptor name and you just need to click on the 'Scan' button. With a bit of luck, your local wireless network will be detected, and you need to double-click on it to open the network configuration window. Enter the network password into the PSK field and click on 'Add'. Close the Scan Results window and you should see that the Current Status page shows your Pi is connected to the network. Network settings can always be changed from the Manage Networks page, and you should choose 'Save Configuration' from the File menu before closing the tool or restarting your Raspberry Pi.



› HDMI can easily be converted into DVI for use with the majority of flat-screen panels.

USB keyboard and mice

If you've got a working network connection on your Raspberry Pi you don't necessarily need a keyboard or a mouse. With the Raspbian operating system installed, you can connect remotely to your Pi using a protocol called SSH. From a Windows machine, install the free Putty client. OS X and Linux users will find an SSH client built in to their command interfaces – just type **ssh** followed by **pi@** the IP address of your Pi (your home router or hub will have a configuration page listing the IP addresses of any connected devices). The password is 'raspberry' for the default 'pi' user »



› If you connect to your Pi remotely, you can use the mouse, keyboard and display of a different machine.

account, and when connected, you'll be able to type and configure your Pi directly without having to buy or connect a mouse, keyboard or screen. If you want a graphical interface, you can even install a package called **tightvncserver** and run the command **vncserver :1** to launch a remotely accessible desktop. Just use any freely available VNC client on another machine to connect to the IP address and port 5901 on your Pi. You'll then be able to use your keyboard and mouse remotely, which is ideal if your Raspberry Pi is inaccessible.

But if you do want to connect a keyboard and mouse, you shouldn't encounter too many difficulties. Most models use a USB standard that means keyboards and mice will just work, so unless you buy something esoteric – such as the latest wireless models from Microsoft or Logitech, mice and keyboards will work without any further configuration. But you do need to make sure you connect these devices to a powered USB hub (as we keep saying), as this will ensure there's not too much power drain on the Raspberry Pi itself. If you need to customise the keyboard layout, type **sudo dpkg-reconfigure keyboard-configuration** from the command interface. This should ensure the keys of your keyboard line up with those being input on the Pi.

> **"If you do want to connect a keyboard, you shouldn't have too many difficulties"**

Display and sound

There are two connectors on the model B Raspberry Pi that are capable of sending a video signal to a display device. The yellow phono jack is for composite video and you can connect this to a great number of television sets that usually have a yellow composite video-in port for external cameras or recorders. It will often be close to a red and white jack for a left/right analogue audio signal. While the Raspberry Pi does default to enabling the HDMI connector if it detects a connection, it will fall back to the composite video connector if it doesn't, so make sure there's nothing connected to the HDMI port if you want to use the composite.

However, composite isn't the connector we'd recommend using unless you've no other option. The modern HDMI connector on the board is much better suited to display duties, and you don't necessarily need an HDMI input on a display to be able to use it. The video part of the signal carried across the HDMI signal is exactly the same as the video carried by the very common DVI-type cable that most computer monitors use. And for that reason, you can easily user either a cable with a male DVI-24+1 connection on one end and a male 19-pin HDMI connector on the other, or an adapter that uses a male 19-pin HDMI connector for the Raspberry Pi and a female DVI connector for connection to a regular DVI cable. Despite the slightly technical nature of those descriptions – which we've given for completeness – both the DVI/HDMI cable and the adaptor are common devices, and almost any electronics store will be able to sell you either cheaply and easily. Because they carry digital signals, don't spend any extra on special gold adaptors or converters, because there's very little evidence to show they improve the signal strength.

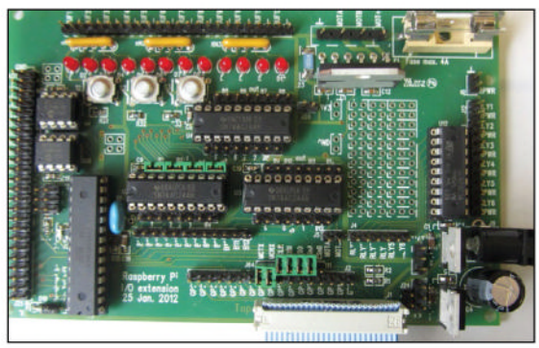
Another strength of the HDMI connector is that it can also carry the digital sound data from the Raspberry Pi. Your TV or amplifier will need to be compatible with this feature to work, and you will need to play around with the Raspberry Pi configuration, but it's a neat solution if you want to turn your Pi into a media hub.

It won't work with an HDMI>DVI converter, because it's the sound component that's dropped in the conversion, but then you've still got the analogue audio port to use. This is a noisy output if you want decent quality, but it's good enough if you're streaming video to a TV.

GPIO

One set of connections that isn't quite in the same category as the rest is the series of pins on the top surface of the Raspberry Pi. These are the General Purpose Input Output connectors (GPIO for short), and they're intended to be used for hardware projects and for connecting your Pi to low level peripherals. Each pin has its own function, and they can be controlled from a development environment or programming language running on the Raspberry Pi. You might want to use them to light up some LEDs, for example, or at the heart of a Raspberry Pi robot where the GPIO pins are used to physically control the motors.

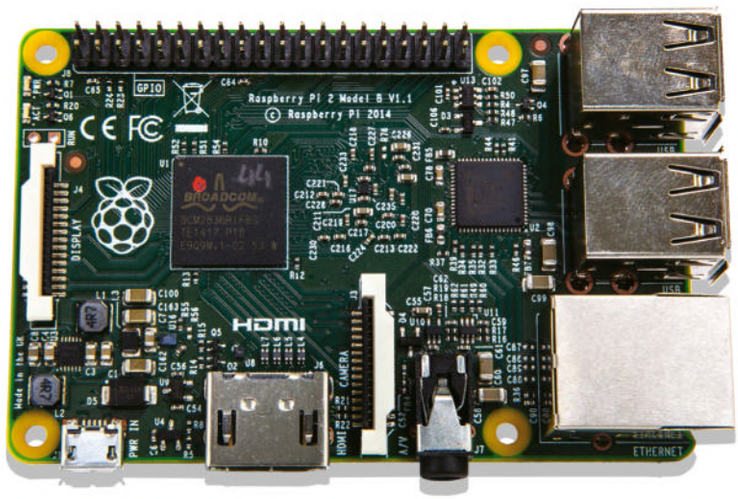
There are 26 pins on the board (labelled as P1), and their function has changed slightly with different revisions of the Raspberry Pi. For this reason you need to know which revision you have if you plan to buy a daughterboard that plugs directly on to the GPIO pins, but any Pi from the last six months or so will be revision 2. The most famous of these expansion boards is the Gertboard, which adds all kinds of extra programmable functionality to the Raspberry Pi, including a motor controller, two analogue/digital converters, 12 LEDs and many more jumpers and connectors. But you don't necessarily need an additional board to get started, because the pins can be used for many projects without modification. The latest update also adds a P5 header to the board just adjacent to the P1 GPIO pins. These extra eight pins add power and four additional GPIO functions, which can be used mainly to provide a secondary I2C channel, which is a standard bus for serial communication.



Boards that connect to the GPIO pins on your Raspberry Pi, such as the Gertboard, can be used.

Fixing faults:

Nine times out of ten, you won't need these two pages – your Pi will simply boot. But this is the place to look if things go wrong.



The big difference between your Raspberry Pi and your PC is that the Pi doesn't have a BIOS. It's the BIOS you see first when you turn on your computer – the retro-looking splash screen or the text detailing CPU, memory and storage, only it appears too quickly to read. If there's a problem with your PC, the BIOS can often be used to troubleshoot the booting, and it will also create noises and flash LEDs if bad memory or a dodgy CPU is detected. But without the facilities of a BIOS, the Raspberry Pi doesn't have the same level of fallback, and while hopefully your Pi will boot first time without problems, the more you play with your Pi, the more likely it's going to be that you encounter a boot problem. Which is what these two pages are for. Before we start know that every Pi before it leaves the Sony factory is tested, so if it's new and unwrapped it should work:

What the LEDs mean

The only way the Pi can impart diagnostic details is through the various LEDs it has on its board. Similar to the obscure BIOS beeps a big PC uses to wail its problems at you with potentially no working display.

Each model of Pi sports its own array of LEDs. The original Model B had five, the A two and the latest Pi 2 also only has two. The number isn't really important as only one explains boot issues. First lets see which versions have what LEDs. The Model B sports five LEDs, while the Model A has only two, although the printed labels remain.

All models of Raspberry Pi

- **LED1:** Green, labelled ACT: SD card access
- **LED2:** Red, labelled PWR: 3.3V power is present

Only on the original Raspberry Pi Model B

- **LED3:** Green, labelled FDX: Full duplex (LAN) connected
- **LED4:** Green, labelled LNK: Link/activity (LAN)
- **LED5:** Yellow, labelled 100: 100Mbit (LAN) connected

On the Model A/A+ with no wired networking, the last three LEDs aren't present on the PCB, and you'll find the labelling is slightly different on earlier revisions of the Model B, although their functions are identical.

When you first connect the Pi to a power source, the red LED2 should light. This indicates the device is getting the correct amount of power, and this LED should remain lit for the entire time your Raspberry Pi remains powered. Even when there's no network connection, or if the SD card isn't connected, this LED should stay lit. If it flickers, or if it goes off, you've got a problem with the way your device is being powered, and the first thing you should check is the cable and the power supply unit. The Pi 2 added detection for poor power supplies was added. If the Pi detects an inadequate supply or a borderline one, the power LED remains unlit.

With the SD card connected, the edge-side LED should be the next to light. This is the LED that signals that data is being read from the SD card.

Boot sequence

Initially, it will flicker on then off, pause for a moment, then pulse on and off again as the Pi reads the boot code off the SD card. If it doesn't get that far, then the problem is going to be either that the boot code hasn't been correctly written to your storage card, or that your storage card isn't working with your Pi. Be sure to check the card is seated properly and that any micro SD adaptor you might be using is connected. You can also check to make sure the connectors on the Pi are springy and look identical, as there have been a few reported problems with these connectors.

It's also possible to decode which part of the boot process the Pi is stalling at. Here's a list of what the various flashing modes from the ACT/OK LED mean – although these rely on a firmware from at least the middle of 2012, and we've taken this data from the Raspberry Pi forums. In use, we've found it easier to ignore the flashing status unless you're prepared to delve into the complexities of the boot procedure, and while this can be fun, it's also frustrating if you just want to start playing with your new device.

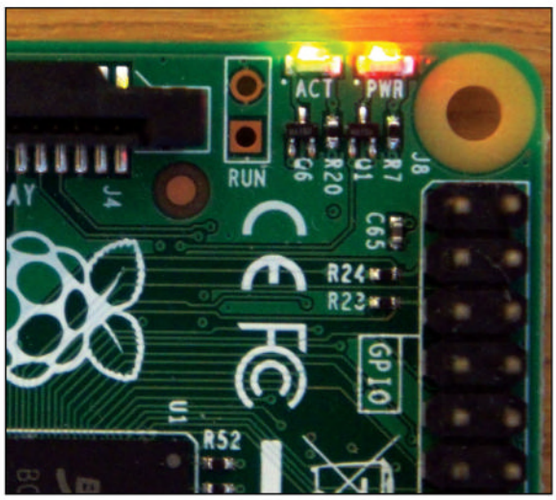
- **3 flashes:** start.elf not found
- **4 flashes:** start.elf not launched
- **7 flashes:** kernel.img not found
- **8 flashes:** SDRAM not recognized. You need newer bootcode.bin/start.elf firmware, or your SDRAM is damaged

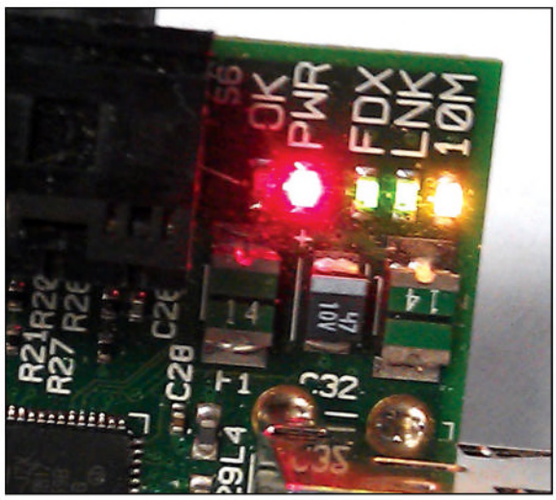
For devices made before October 2012 or old devices there have never been updated:

- **3 flashes:** loader.bin not found
- **4 flashes:** loader.bin not launched
- **5 flashes:** start.elf not found
- **6 flashes:** start.elf not launched

It is possible to troubleshoot these problems by looking for the files involved and making sure they're correct. Take a

The boot





On the left the LEDs from the latest Pi 2, on the right, the LEDs from the originial Model B.

look at something called a checksum and make sure the checksum for the files on the card is the same as the checksums for the original files.

As these read errors are more likely to be an indication of either the SD card not being read correctly, or the Raspberry Pi operating system not being written correctly, we'd recommend trying to get hold of a new card first and writing the image with a different method. Our installation guide on page 10 covers three different operating systems, so it might be worth trying a friend who uses something else. However, we have experienced problems with several SD card readers – especially those embedded within laptops and netbooks – and we'd suggest switching to a standalone reader first to see if that helps. These are available cheaply, but they can often be found bundled with the SD cards themselves. If you've got your Pi connected to a display, recent versions will also show a kaleidoscopic boot screen. If it makes it past this and no further, then the problem is once again in the hands of your power supply.

Even when booting sometimes you won't immediately get a video signal The raspberry PI (its Raspbian OS) is designed to output a HDMI signal, but if it doesn't detect a HDMI device connected it will default to generating a composite signal on the RCA port (or 4-pins 3.5mm A/V jack on model B+) That sometimes means you have to turn on your monitor before booting. It may even mean that the monitor must be switched to HDMI input. It depends on the monitor. Booting NOOBS works a bit differently, as it will always output a HDMI signal, (even if you have nothing connected to the HDMI port) unless you press one of the numerical keys 3 (PAL) or 4 (NTSC) to switch to a composite video output mode.

Networking

If your Raspberry Pi has been able to successfully negotiate the early boot procedure, the operating system will start being read off the card in earnest. Very soon after Linux starts to boot, there will also be a flicker of the fourth LED (LNK), followed by the remainder of the networking LEDs lighting up about half a second later – these LEDs on newer models are found built into the network Ethernet port. This happens as the networking stack is initiated and forms a link with your Ethernet network. The status of these LEDs is exactly the same as the LEDs on the rear of nearly any other Ethernet port, and are more likely to indicate problems with your network than with the configuration of the Pi itself. The orange LED indicates a full duplex connection.

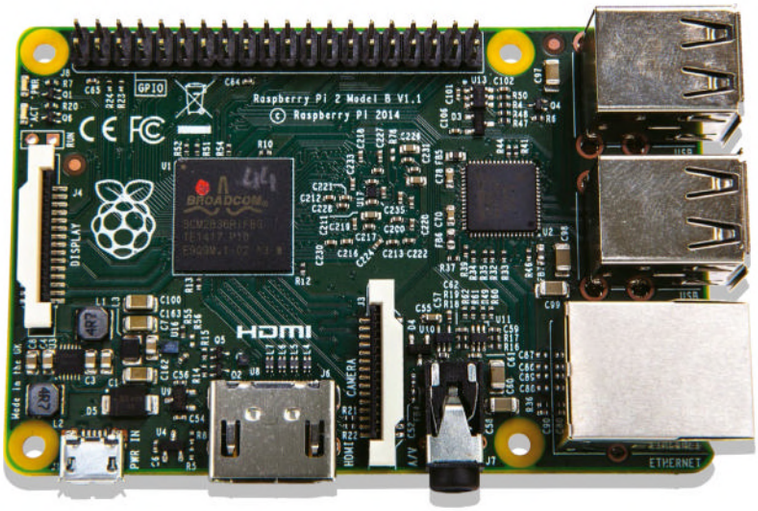
The green LNK LED is the networking equivalent to the ACT LED for SD card access, because it's this that flickers on and off as data is received and transmitted across the network. If the LED does light up, you can assume, at least at a low level, your hardware has been able to negotiate a connection between your hardware and the switch that manages your network. But this doesn't necessarily indicate a working network connection where you can browse the web or check your email.

If you've got activity but no working connection on your Pi, you should look at your local network configuration, first by making sure your Pi is getting an IP address from your router, and then secondly from the Raspbian operating system itself. It is able to configure itself under almost any circumstances, but there are sometimes problems if your network is a little weird – you could be using two domains, for example, or a couple of routers. The only way to solve these problems is to try to connect your Raspberry Pi to the simplest and most visible part of your network and then work your way back to the configuration you'd like to see.

The final LED is used to indicate the speed of your network. If it's on, then your Pi has negotiated a speed of 100Mbps. That's 1,000,000 bits of data per second, or 100 megabits, otherwise known as a Fast Ethernet connection. This is a step up from the very basic 10Mbps indicated if this LED doesn't light, but it's still some way from the current fastest standard, which is 1,000Mbps, also known as a Gigabit connection.

Command line:

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



As you have no doubt discovered, Raspbian has a graphical interface similar to that of Windows or Mac OS X. You can do most of your day-to-day tasks in this interface. There's a file manager, web browser, text editor and many other useful applications. However, sometimes you need an interface that's a bit more powerful, and this is where the command line interface (CLI) comes in. It's also known as the terminal or shell.

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

The first thing you need to do is open up a terminal. Click on 'LXTerminal' on the Raspbian desktop.

You should now see a line that looks like:

```
pi@raspberrypi ~ $
```

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

```
/home/pi
```

If you've changed your username, then you'll see a different line. The rather cryptically named **pwd** command stands for Print Working Directory, and the system simply outputs the directory you're currently in. When you start a terminal, it will go to your home directory.

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

```
cd ..
pwd
```

You should find that the system returns **/home**. This is because we've **cd**ed to '..' and two dots always points to the parent directory. To move back to your home directory, you can enter **cd pi**. There is also another way you can do it. The ~ (pronounced tilda) character always points to your home directory, so wherever you are in the filesystem, you can enter **cd ~** and you'll move home.

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

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

Knowing what commands to use

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

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

The second way of finding information on a command is using **man**. This is short for manual. It takes a single argument, that is, a word after the command that isn't preceded by a hyphen. It then displays information on the command given as an argument. To see the man page for **ls**, type **man ls**. You can navigate through the page using the up and down arrows, or the page up and page down keys to

Interactive programs

Most of the commands we're dealing with here are non-interactive. That means you set them running and then wait for them to finish. However, not all command line programs work like this. For example, when you first booted Raspbian, it started a config tool that ran in the terminal. There are a few other programs that work in a similar way. Traditionally, the most common has been text editors that allow you to work on files if you don't have a graphical connection. There are a few quite complicated ones that are great if you spend a lot of time working from the command line, but they can be hard to learn. There's also an easy-to-use terminal-based text editor called **nano**. Enter **nano** followed by a filename at the command prompt to start it. You can then navigate around the text file and make any changes you need. Press [Ctrl]+[X] to save your work and exit back to the prompt.

Learn the ropes

scroll faster. To search for a word or phrase inside the man page, type **/**, then the phrase. For example, **/-l** will find all occurrences of **-l**. You can use the [N] key and [Shift]+[N] to scroll forwards and backwards through the occurrences of **-l**.

As we introduce more commands, it's good to take a look at the help and the man page to familiarise yourself with what they do. Of course, you can always Google a command if you find the text-based help a little off-putting, but staying in the terminal will help you become more familiar with the command line interface.

How will I know?

Remember we said there's good news and bad news? Well, the bad news is that it can be quite tricky to find commands if you don't know what they're called. One helpful feature is the **man** keyword search. This is done with the **-k** flag.

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

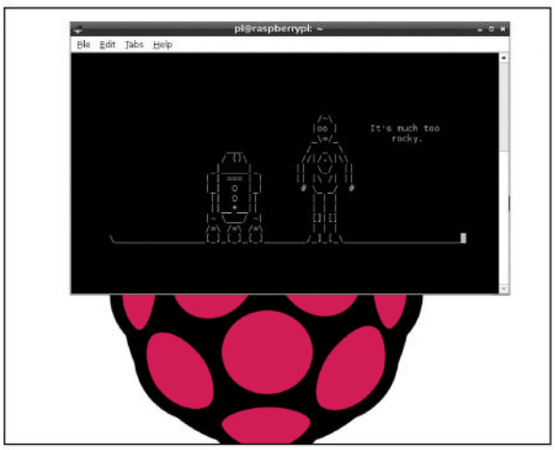
If you've got *Iceweasel* (a rebranded version of *Firefox*) installed on your Pi (it's not on there by default), you can open **TuxRadar.com** in a new tab in a currently running *Iceweasel* window with the command **iceweasel --new-tab www.tuxradar.com**.

We're now going to quickly introduce a few useful commands. **rm** deletes (ReMoves) a file. **mkdir** makes a new directory. **cp** copies a file from one place to another. This one takes two arguments, the original file and the new file. **cat** outputs the contents of one or more text files. This takes as many arguments as you want, each one a text file, and simply spits their contents out on to the terminal. **less** is a more friendly way of viewing text files. It lets you scroll up and down using the arrow keys. To exit the program back to the command line, press [Q.] We'll use all these commands in examples below, so we won't dwell on them for too long.

find is a useful command for finding files on your computer. You use it in the format **find location flags**. For example, to find every file on your computer that's changed in the last day, run

```
find / -mtime 1
```

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



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

Tab completion

When you're dealing with long filenames, it can be very annoying to have to type them out every time you want to run a command on them. To make life a bit easier, the terminal uses tab completion.

This means that if you start typing a filename and press the [Tab] key, the system will try to fill in the rest of the name. If there's only one file that fits what you've typed so far, it will fill in the rest of the name for you (try typing **cd /h** then pressing [Tab]).

If there are more than one, it will fill in as far as the two are the same, If you press [Tab] again, it will show the options (try typing **cd /m**, and then pressing [Tab] twice).

Power up

So far you're probably thinking to yourself, "I could have done all this in the graphical interface without having to remember obscure commands and flags." You'd be right, but that's because we haven't introduced the more powerful features yet. The first of them are wildcards. These are characters that you can put in that match different characters. This sounds a bit confusing, so we're going to introduce it with some examples.

First, we'll create a new directory, move into it and create a few empty files (we use the command **touch**, which creates an empty file with the name of each argument). Hint – you can use tab completion (see boxout) to avoid having to retype long names, such as in the second line.

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

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

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

```
ls *
```

This isn't particularly useful, but we can put it in the middle of other characters. What do you think ***.txt** will »

match? Have a think, then run:

```
ls *.txt
```

to see if you are right. How about **one*** ? Again, run

```
ls one*
```

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

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

We can then check they made it there correctly with:

```
ls text-files/
```

The second wildcard we'll look at is ?. This matches any single character. What do you think:

```
ls ???
```

will match? Have a guess, then run it to see if you're right.

We can also create our own wildcards to match just the characters we want. **[abc]** will match just a lower-case A, B and C. What do you think **ls [ot]*** will match? Now try

```
ls [!ot]*
```

What difference did the exclamation mark make? It should have returned everything that didn't start with a lower-case letter O or T.

Pipes and redirection

The commands that we've been looking at so far have all sent their output to be displayed in the terminal. Most of the time this is what we want, but sometimes it's useful to do other things with it. In Linux, you can do two other things with a command: send it to a file, or send it to another program. To send it to a file, use the > character followed by the filename. Run:

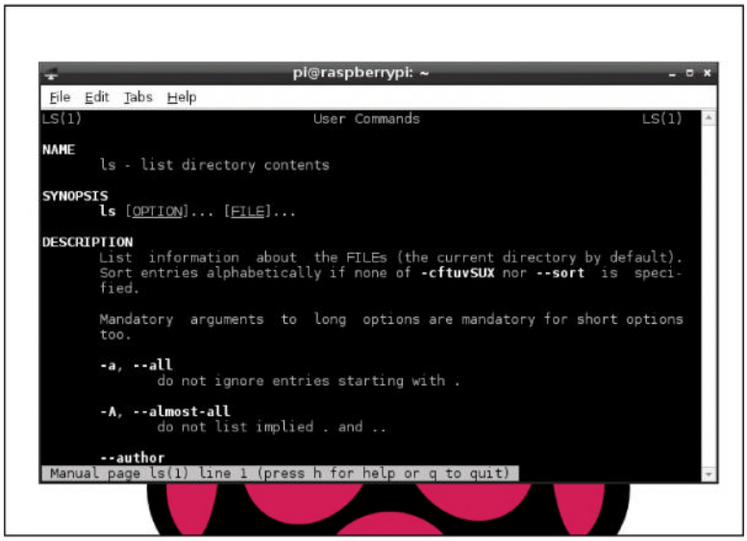
```
ls > files
cat files
```

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

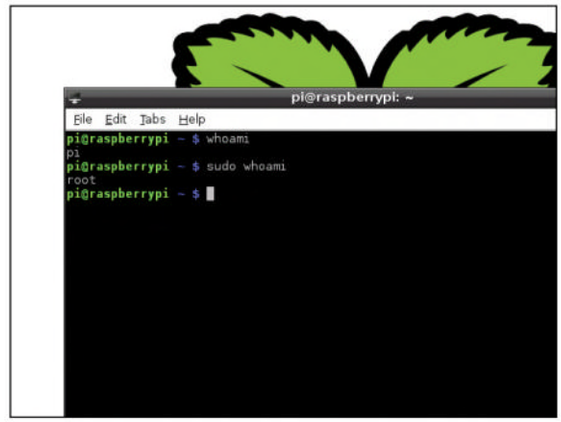
The second option, sending it to another program, is another of the really powerful features of the Linux command line, because it allows you to chain a series of commands together to make one super command. There are a lot of commands that work with text that are designed to be used in this way. They're beyond the scope of this tutorial, but as you continue to use the command line, you'll come across them and start to see how they can be linked together. We'll take a look at a simple example. If you run **find /** (don't do it just yet!) it will list every file on the system.

This will produce a reel of filenames that will quickly go off the screen. However, rather than direct it to the screen, we can send it to another command that makes the output easier to read. We can use the **less** command that we looked at earlier for this. Run:

```
find / | less
```



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



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

Take it further

We've only been able to touch on the basics of using the command line, but you should have enough knowledge now to get started, and hopefully you're beginning to see just how powerful the command line interface is once you get to know it. [*see page 92*, to take your knowledge further!]

If you want to know more (and you should!) there are loads of resources in print and online. **LinuxCommand.org** is a great place to start. Its book (*The Linux Command Line*) is available from bookshops, or for free online (**www.linuxcommand.org/lc3_learning_the_shell.php**).

sudo

When using the Raspberry Pi for normal use, you can work with files in your home directory (for example, **/home/pi**). You will also be able to view most other files on the system, but you won't be able to change them. You also won't be able to install software. This is because Linux has a permissions system that prevents ordinary users from changing system-wide settings. This is great for preventing you from accidentally breaking your settings. However, there are times when you need to do this. You can use **/sudo** to run a command as the super user (sometimes called root), which can do pretty much anything on the system. To use it, prefix the command with **sudo**. For example:

```
sudo apt-get install synaptic
```

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

Cut-out-and-keep command line reference

Navigation and files

» **cd** Changes directory. For example, **cd movies** moves to the **movies** folder. **cd ~** moves to your home directory, **cd /** moves to the root directory, **cd ..** moves back one directory.
» **ls** Lists files. By itself, it lists the files in the current directory. **ls movies** lists the files in the directory **movies**. **ls -a** lists all files (including hidden ones), and **ls -l** lists more information about each file.
» **cp** Copies files. **cp orig-file new-file** copies **orig-file** to **new-file**.
» **wget** Downloads a file from the internet. To download the Google homepage to the current directory, use **wget www.google.com**.
» **df -h** Displays the amount of space left on the device.
» **pwd** Displays the current directory.

Finding files

» **find <location> <tests>** useful flags include: **-mtime <number>** finds files modified in the last **<number>** days. **<number>** could be, for example, 2 (exactly two days ago), -2 (less than two days ago) or +2 (more than two days ago). **-name <filename>** finds files called **<filename>**. **-iname <filename>** matches files called **<filename>** but not case-sensitive. **-writable** finds files that are writable. There are many more options. See the man page for a detailed list. For example **find / -mtime -2 -writable** finds all files on the filesystem that were changed less than two days ago and are writable by the current user.

Remote working

» **ssh** Log in to a remote computer using Secure SHell (SSH protocol). **ssh pi@192.168.1.2** will log in as user pi on the computer at the IP address **192.168.1.2**. Note, this will only work if the remote computer has an SSH server running.
» **scp** Secure copy. **scp file pi@192.168.1.2 :/home/pi** will copy a file to the directory **home/pi** on the machine with **192.168.1.2**. **scp pi@192.168.1.2:/home/pi/file**. will copy **/home/pi/file** from the machine **192.168.1.2** to the current directory. Note, this will only work if the remote machine has an SCP server running.

Wildcards

» ***** Matches any string of characters, or no characters.
» **?** Matches any single character.
» **[abc]** Matches a, b or c.
» **[!abc]** Matches any character except a, b or c.
» **[A-Z]** Matches any character in the range A–Z (that is, any upper-case letter).
» **[A-z]** Matches any character in the range A–z (that is, any upper- or lower-case letter).
» **[one, two]** Matches the words one and two.

Information about the computer

» **top** Displays the programs that are currently using the most CPU time and memory.
» **uname** Displays information about the kernel. **uname -m** outputs the architecture it's running on.
» **lscpu** Lists information about the CPU.
» **dmesg** Displays the kernel messages (can be useful for finding problems with hardware).

Text files

» **head** Displays the first 10 lines of a text file. Change 10 to any number with the **-n** flag. For example, **dmesg | head -n 15** displays the first 15 lines of the kernel message log.
» **tail** Displays the last 10 lines of a text file. Can use the **-n** flag like **head**. Can also keep track of a file as it changes with the **-f** (follow) flag. For example, **tail -n15 -f /var/log/syslog** will display the final 15 lines of the system log file, and continue to do so as it changes.
» **less** Allows you to scroll through a text file.
» **cat** Dumps the contents of a text file to the terminal.
» **nano** A user-friendly command line text editor ([Ctrl]+[X] exits and gives you the option to save changes).

Special keys

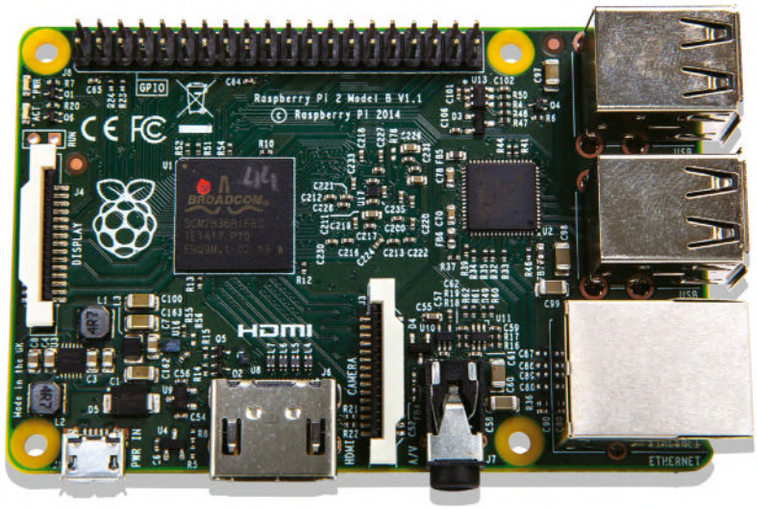
» **[Ctrl]+[C]** Kills whatever is running in the terminal.
» **[Ctrl]+[D]** Sends the end-of-file character to whatever program is running in the terminal.
» **[Ctrl]+[Shift]+[C]** Copies selected text to the clipboard.
» **[Ctrl]+[Shift]+[V]** Pastes text from the clipboard.

Installing software

» **tar zxvf file.tar.gz**
» **tar xjf file.tar.bz**
» **./configure** When you unzip a program's source code, it will usually create a new directory with the program in it. **cd** into that directory and run **./configure**. This will check that your system has everything it needs to compile the software.
» **make** This will compile the software.
» **make install (needs sudo)** This will move the newly compiled software into the appropriate place in your system so you can run it like a normal command.
» **apt-get** This can be used to install and remove software. For example, **sudo apt-get install iceweasel** will install the package **iceweasel** (a rebranded version of *Firefox*). **sudo apt-get purge iceweasel** will remove the package. **apt-get update** will grab an up-to-date list of packages from the repository (a good idea before doing anything). **apt-get upgrade** will upgrade all packages that have a newer version in the repository.
» **apt-cache search <keyword>** Will search the repository for all packages relating to keyword.

The filesystem:

Get to know how and where Linux keeps its files, and discover how to add additional storage to your Raspberry Pi.



The Linux filesystem is all based around **/**. If you think about the filesystem as a building complex, this denotes the front door. Everything else is based around it. Take a look by opening up a terminal and typing:

```
cd /
ls
```

This contains some files and directories. The directories are, to continue our metaphor, like doors to additional rooms. The most important room for most users is their home directory. We can take a look at this by entering

```
cd /home
ls
```

Did you notice the **/** before **home**? This means that home sits within the root directory (**/**). When you use a location that starts with a **/**, it's called an absolute location (or path). The **ls** will then show us all the files in **/home**. There should be one directory for every user. You can enter the directory for the user pi with:

```
cd pi
```

This time we didn't use a **/** before the directory name. This means that the directory sits in the current directory. This is called a relative location (or path). We could also have used the absolute location with **cd /home/pi**.

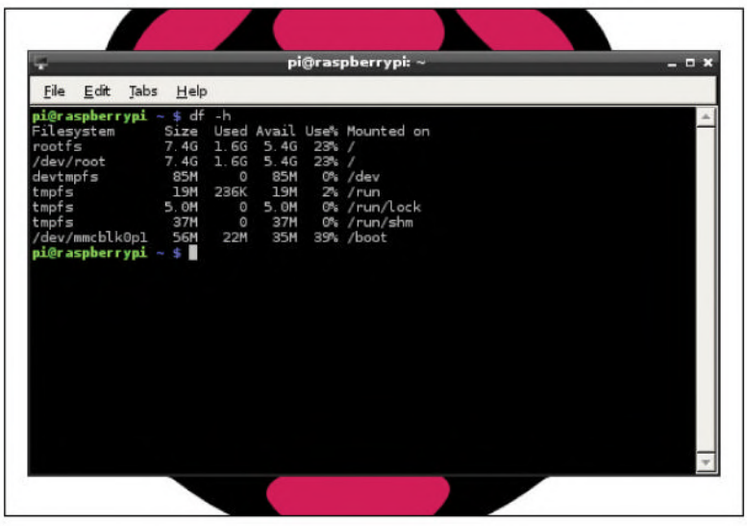
So, if we think about the filesystem as a building complex containing a load of rooms, it's important to realise that not every directory is on the same device. We can think of this a bit like the rooms being in different buildings connected via walkways. These are completely invisible to the casual user. So, in the previous case, the directory **pi** could have been on a different physical drive to **home** and we would never have known it. This may seem a bit odd to those of you used to the Windows method, where each drive is kept separate, but it makes for a much more flexible system.

To see what devices are attached to your system, type:

```
df -h
```

You should get a few lines. The first column tells you where the filesystem is (this isn't the location of the mountpoint – or door – but the location of the file for the filesystem). There are basically two types – those that start with **/dev/** and those that don't. The latter aren't real devices, but virtual devices created by the operating system that contain virtual files that have system information. For example, you should see one that's mounted at **/dev**. This contains a file for each of the devices on the system, including all the drives. That's why the lines in the output that relate to physical devices have **/dev/** in the first column.

For the devices that start with **/dev/**, there are again two categories. Those that start with **/dev/mmcblk** (the ones on the SD card), and the ones that start with **/dev/sd** (the ones on additional devices. Since we'll leave the SD card as Raspbian has set it up, we'll just look at the ones starting **/dev/sd** (note, you only see entries for these if you've added extra storage). Each drive is assigned a letter starting with A, so **/dev/sda** is the first drive, **/dev/sdb** is the second, and so on. Then each partition on the drive is given a number, so **/dev/sda1** is the first partition on the first drive, **/dev/sdb3** is the third partition on the second drive, and so on.

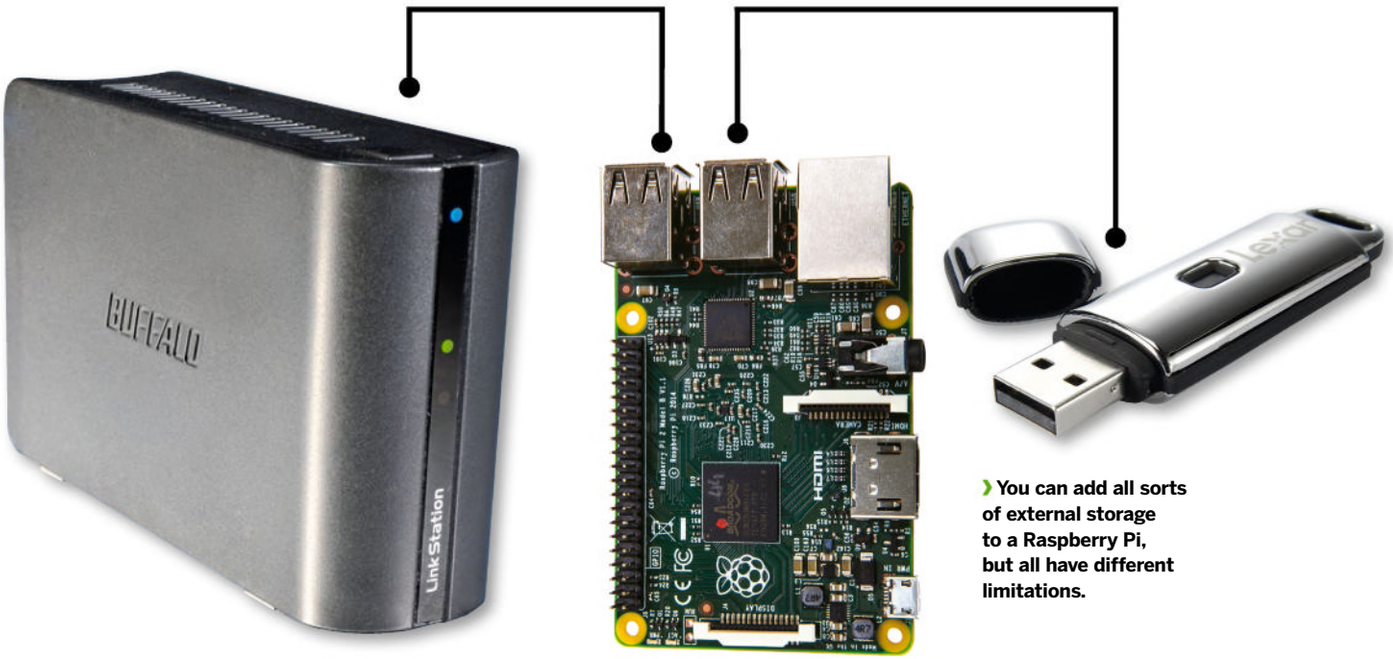


df -h is a really useful command for displaying the drives currently attached to your Raspberry Pi.

Adding storage

You can add storage to your Raspberry Pi in more or less any form as long as it has a USB connection. USB memory sticks and external hard drives are the most common. With external hard drives, you need to consider how to power them. The Raspberry Pi isn't designed to provide as much power over USB as an ordinary computer, so if the drive has only a USB connection and not an extra power adapter, it probably won't work if you just plug it in. You can use a powered USB hub to get around this. A memory stick draws much less power, and

Storage



You can add all sorts of external storage to a Raspberry Pi, but all have different limitations.

so should work if you plug it into the USB port on the Raspberry Pi, although you may need a hub if you also want a mouse and keyboard, because there are only two ports available.

It's also worth noting at this point that USB connections on the RPi are slower than on most computers, so don't expect the same transfer rate. This will be especially noticeable with USB 3.0 devices that will fall back to USB 2.0 when attached to the RPi, and so function far slower. This is due to the trade-offs essential for making the board so small and cheap.

Data layout

So far, we've used the word filesystem to mean the layout of the directories on the computer. This is correct, but it also has another meaning: the way data is physically stored on the disk. This is important because each drive needs to be formatted to a particular filesystem before it will work. There are a number of different filesystems available – some that Windows uses, some that Mac OS X uses, some the other systems use. The good news is that Linux works with almost every one, so if you want to share your drive between your Pi and another computer that doesn't run Linux, the easiest thing to do is format it on the other device. If you do it the other way around, then you have to make sure you pick the right filesystem type for the device. Most devices come formatted to FAT32, which is readable by almost everything. The only problem with it is that it can't handle large files (the limit is 4GB). If, on the other hand, you only want to use the device with your Raspberry Pi and other Linux devices, you'll get the best performance out of it if you format it to one of the Linux filesystems. The most popular is ext4. If you've formatted your drive, and put it in your Pi, open a terminal and type:

```
df -h
```

Did a line for it come up? Probably not. This is because we need to mount it.

We can do this via the terminal. Firstly we need a mount point. This is just the directory that we will use as a door to the drive. It can be absolutely anywhere in the filesystem. For example, it could be a directory called **data** in the user's home directory. To mount the first partition on the first device to a data directory in the user's home partition, enter the following:

```
cd ~
mkdir data
sudo mount /dev/sda1 data
```

Of course, we don't want to have to do this every time we restart our Pi, especially if we want to run it without a keyboard and monitor attached, so we can set it up to do this automatically. There's a file that tells the computer what to mount where. It's called **fstab**, and it's in **/etc**. To open it in a command line text editor, open a terminal and type:

```
sudo nano /etc/fstab
```

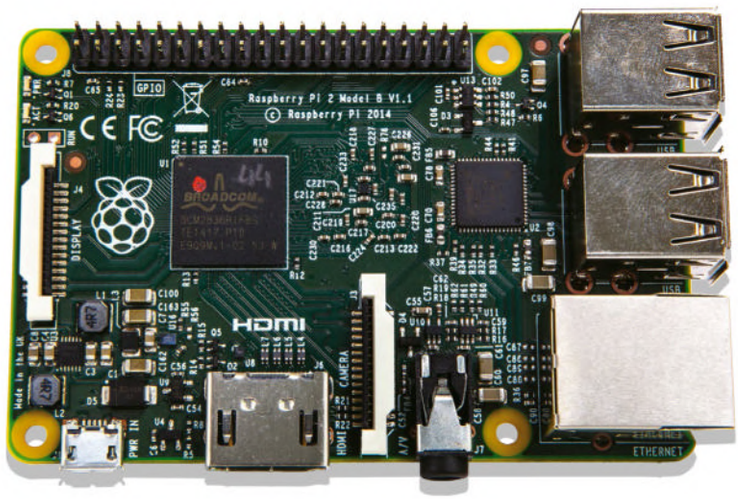
This file contains one line for each partition of each device it mounts. Each line is a series of tab separated values that tell the system what to do. In order to get our drive to mount automatically at startup, enter the following in a new line at the bottom of **fstab**:

```
/dev/sda1		/home/pi/data
```

(Note: the directory data must already have been created, like in the previous example.)

Beginnings: On

Before you get stuck in with your new wonder device, read on to get some background on why the Raspberry Pi exists.



Nowadays, computers keep their workings hidden. You drop down through a menu, click on an icon with a mouse, and the program you're using does what you've told it to do. Or at least, it does what the creator of the software wanted it to do. Sadly, for most people, using a computer amounts merely to using software that someone else has written for them, rather than learning how to create their own solutions to problems. It's true that modern graphical user interfaces have opened the usefulness of computers to a much wider audience than would have been possible without them, but it's also true that those users are getting a shallow experience of what their computers can do.

Another factor discouraging people from looking beneath the surface of their machines and finding out how they actually work is that Windows and Mac OS X both have clauses in their licence agreements forbidding you from modifying the code they're written in. Even though you paid good money for it, you're not allowed to take it to bits and put it back together again. Combined, these two factors mean that, although computers are all around us, most people are ignorant of what goes on inside them.

Enter the Raspberry Pi

This sets the stage for the entrance of our hero: Eben Upton, director of studies for the computer science program at St John's College, Cambridge. As part of his role, he's involved in the admissions process, and over the years he noticed a gradual decline in the standard of applicants.

The more time that universities waste in bringing below-par candidates up to scratch, the less time they're teaching them the cool stuff. So it's not just the calibre of students that's suffering, it's also about the calibre of employees that are going into the computing industry, and the extra time and effort that employers are having to go to in order to find employees who can do the job.

So Eben Upton, Robert Mullins (trustee of the Raspberry Pi Foundation) and others decided to redress this situation, by creating something hackable, cheap and intellectually free, which would be able to do everything that a desktop PC can do, at a fraction of the cost. What they came up with, as you'll no doubt have guessed, is the Raspberry Pi.

The BBC Micro

The Raspberry Pi isn't the first computer launched with the intention of improving British IT education. In 1981, Acorn Computers, in collaboration with the BBC's Computer Literacy Project, released the BBC Micro. It was available in two versions – Model A and Model B – and was designed to run programs that would help kids learn to use computers, along with the BBC's TV series *The Computer Programme*. The biggest difference between the two products is the price: in 1981, the Model A cost £235, while the more powerful Model B cost a whopping £335. The BBC Micro was a raging success, and a generation of British children grew up with an intimate knowledge of computer programming thanks in no small part to its influence.

As a historical footnote, the chip at the heart of the Raspberry Pi is based on a design by one of the companies that eventually span off from Acorn Computers – Cambridge's ARM Holdings.

Adjusted for inflation, the Model B BBC Micro would cost over £1,100 in today's money – a far cry from $35 for a Pi.

the origin of Pi

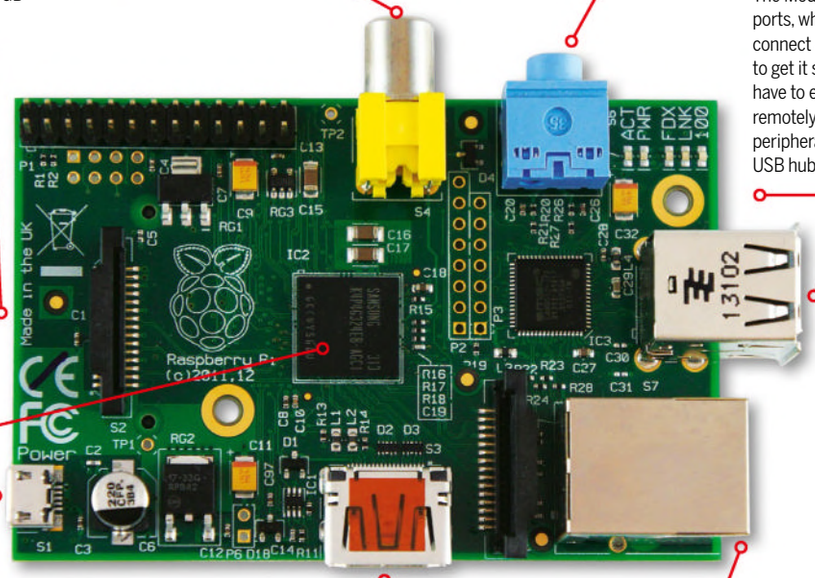
A look at the original Raspberry Pi

SD card slot
The Raspberry Pi doesn't come with any storage, so you'll have to procure your own SD card to store the operating system and your files. The beauty of this is that you only pay for what you're going to use – you can pay £8 for a 4GB card, £200 for a 128GB card or anything in between.

RCA video out
For older television sets, there's also an RCA video connector. If your screen doesn't accept RCA or HDMI inputs, there are adaptors available to convert HDMI to DVI sockets.

3.5mm audio jack
Use this to plug into a set of speakers and give yourself an instant, cheap hi-fi system.

USB ports
The Model B Pi has two USB ports, which is enough to connect a keyboard and mouse to get it set up. After that, you'll have to either connect to the Pi remotely to connect more peripherals, or use a powered USB hub.



System on Chip
This tiny unit is the really clever bit: it comprises a 700MHz ARM 11 processor and a Videocore 4 GPU. The RAM sits on top of the System on Chip; Model A owners get 256k, while the Model B comes with 512k.

Power connector
The power input for the Pi uses a standard 5V micro-USB connector, meaning that you can pick up a power supply pretty cheaply. And if you have an Android phone, you probably have a compatible power supply already.

HDMI out
The design team assumed that most people would be plugging into either a television or a screen, so there's an HDMI out for connecting to modern television sets and computer screens.

Ethernet
Plug straight into the internet – unless you have a Model A Raspberry Pi, which doesn't have an Ethernet connection, in which case you'll have to make do with a USB wireless adapter.

Raspberry Pi 2 or B+?

There's a whole range of Pis out there these days: Model A+, B+, Compute Module and the Pi 2. They're all essentially the same computer; the main differences are in the price, the RAM and the connectivity. Outside of this they all run the same software. Where the Model B+ has 512k of RAM, the Model A+ only has 256k and the Pi 2 has 1GB. The Model A+ has only one USB port, compared with the Model B+ and Pi 2's four; and the Model B+ has a built-in Ethernet connection, which the Model A+ lacks. The other difference is the price, with the Pi 2 costing $35 rather than $25 or $20. The Model B+ is a versatile unit, and was used while writing this guide. The new Pi 2 is the one to go for as it offers so much more power. The Model A+ is a good choice for minimal projects, and will help keep costs down if you're using lots of units. For an extra few pounds, though, we'd rather have the flexibility of the powerful Pi 2.

RASPBERRY Pi PROJECTS 2015

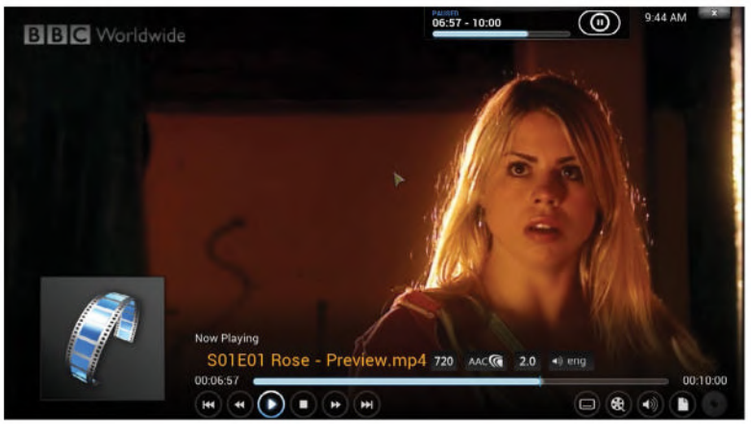
Projects

Kodi media centre 34
Whatsapp on the Pi 40
Install Ubuntu 15.04 44
Create a Tor box 48
Using OwnCloud 52
Create an internet radio 56
Make a Pi home server 60
Mathematica Pi 64
Multi-Pi processing 68
Minecraft Pi hacking 72
Build a Minecraft trebuchet 74
Make 2048 in Minecraft 78

Kodi: A HTPC

Take your home entertainment to the next level with a Pi media centre.

› ***Kodi* has a very informative and intuitive overlay menu for controlling playback.**



You can use the Raspberry Pi to do all kinds of geeky things but one of the most interesting uses for the puny little device is that of a home theatre PC (or HTPC). The small size of the hardware and the fact that it runs silently make it a really good choice for building your own entertainment centre.

One of the best open source apps for turning any computer into a fully functional media centre PC is the recently rechristened *Kodi* media player, formerly known as *XBMC*. *Kodi* uses a 10-foot user interface, which is ideal for connecting to large-screen displays and projectors. The interface has ergonomic display elements and can be easily navigated using a remote control. You can also control playback using your Android smartphone. Using *Kodi* you can view multimedia in virtually any format. Besides playing files from local and network storage devices, *Kodi* can also fetch files from online services, such as YouTube, Spotify, Pandora Radio and more.

While you can install *Kodi* on top of Raspbian, there are several projects that produce a dedicated media centre distro for the ARM-based device, including OpenELEC. The OpenELEC project produces streamlined builds based on *Kodi* for various platforms, including the Raspberry Pi. The advantage with OpenELEC is that you can use the distro without any knowledge of its base Linux OS.

Download OpenELEC

To start setting up your HTPC, grab the OpenELEC build for the Raspberry Pi. As with other projects, OpenELEC hosts different images for the older single-core Pis and the newer quad-core ones. Once you've downloaded the image, extract it and transfer it on to an SD card, either from Linux using the **dd** command, as follows

```
sudo dd if=OpenELEC-RPi2.arm-5.0.8.img of=/dev/sdd
```

or from Windows using *Win32 Disk Imager*.

Then insert the card into the Pi, hook it up to your TV via the HDMI port, and power it on. OpenELEC boots up pretty quickly and takes you straight into *Kodi*. If you've used the media player (or its predecessor) before on the desktop, you shouldn't have any issues navigating it on the Pi. However,

Optimise playback

Although the Raspberry Pi 2 packs quite a punch, there are some tweaks you can do in OpenELEC that result in smoother playback. For starters, you can turn down the video playback resolution to 720p, especially if your HTPC isn't connected to a Full HD TV.

Head to Settings > System > Video Output, and change the Resolution option to 720p. Another trick is to replace the default skin, which was designed for desktop computers, to a lightweight skin, such as Aeon Nox, which makes navigating the menus snappier. To change the skin, head to Settings > Skins. Also make sure that hardware acceleration is turned on. Go to System > Video > Acceleration and check that the Decoding Method is set to Hardware and not Software. While you're here, also reduce the GUI updates when playing video to 5fps. Another playback-related tweak involves matching the refresh rate of the screen to the video being watched, which results in smoother playback. You can enable it by going to Settings > System > Video > Playback, then toggle the Adjust Display Refresh Rate to Match Video option.

Another way to smooth out playback is to use audio passthrough for encoded audio, such as Dolby. To enable the option, navigate to Settings > System > Audio Output and toggle the Enable Passthrough option. Finally, if you are on a slow internet connection, you can cut down on the bandwidth usage by heading to Settings > Video > Library and disabling the Download Actor Thumbnails option.

Back up your video library

Seeing as you've spent a considerable amount of time setting up your HTPC, it would be a shame to lose it all because of a corrupted card. To prevent this happening, you can back up all your customisations and information about your library. *Kodi* includes a backup utility, but we'll use an add-on that enables us to back up files to a custom location, including Dropbox. Head to Programs > Get More and install the *Backup* add-on. Once installed, launch it from under Program > Backup. The program asks you to select one of two modes – Backup or Restore. When you select Backup, it throws an error because we haven't configured it yet. Click OK to bring up the Settings window.

If you wish to back up to Dropbox, use the Remote Path Type pull-down menu to select the Dropbox option, and enter the authentication details for your Dropbox account. Otherwise, click Browse Remote Path and select the location where you wish to store the backup files. Optionally, select the Compress Archive option to reduce the size of the backed-up files. Then switch over to the File Selection tab and customise the list of files you want to back up. Finally, switch over to the Scheduling tab and enable the scheduler to back up automatically, as per your defined schedule.

Once you've set it up, create an initial backup copy by launching the *Backup* program. This time when you hit the Backup button, the program saves the marked files to the specified destination. To restore the files, simply launch the program and click the Restore button. The program shows you a list of all the backups inside the configured backup location.

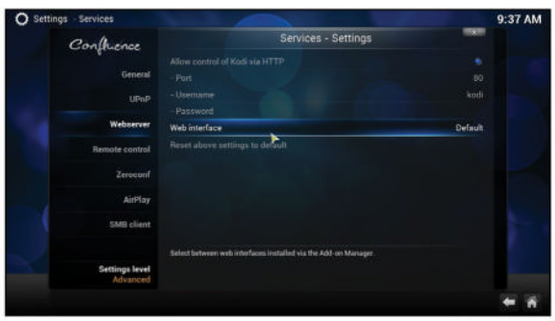
you do need to spend some time configuring it to take advantage of this specialised environment.

Use your keyboard to scroll through the *Kodi* menu and head to System > OpenELEC. This section lists settings and configuration options related to the distro under five different categories. From the System category, you can change the hostname of this OpenELEC installation. This helps you distinguish multiple instances, in case you have more than one on your network – one in the living room, say, and another in the bedroom or kitchen.

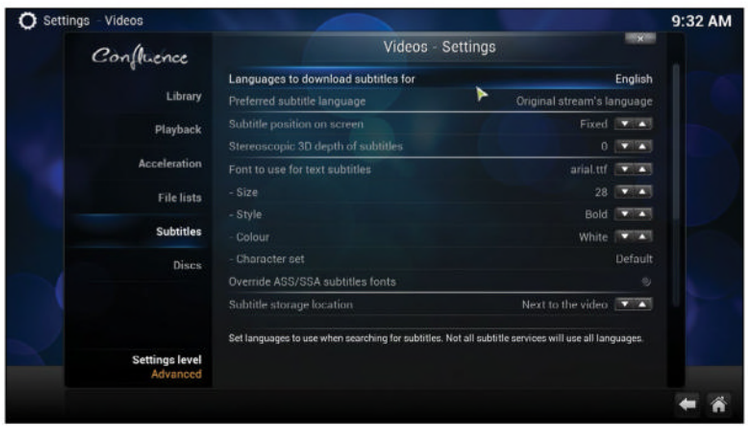
Also, by default, OpenELEC is set up to inform you when a new update is available. However, you can toggle the Automatic Updates option to ask the distro to fetch the update without asking for your approval.

For a better HTPC experience, you can use OpenELEC with a Raspberry Pi-compatible Wi-Fi adaptor. Connect the adaptor and head to System > OpenELEC > Network, and toggle the Active option under the Wireless Networks section. Then switch to the Connections section and select your Wi-Fi network from the list that appears there. Now click Connect and enter the relevant authentication details to connect to the Wi-Fi network.

Then there are *Kodi's* own settings, listed under Settings > System. Using these settings, you can configure audio output, calibrate the monitor, set up remote controls, set up *Kodi's* built-in PVR, and more. To watch TV on your Pi HTPC, head to Settings > System > Live TV, and toggle the Enabled option. You're then asked to enable one of the supported PVR add-ons. Select your PVR from the list and click the Configure button to enter the relevant configuration details, such as the IP address of the PVR host. When you're done, click the Enable button to activate it. Then head back to the System > Live TV section to set up other options, such as the length of the recording, alter the behaviour of the OSD, set up parental controls, and so on.



Your HTPC includes a small web server that lets you control Kodi from a skinnable web interface.



You can set up *Kodi* to search for and download subtitles in your preferred languages.

Add and stream content

Once you've set it up, it's time to add content to your HTPC. You can configure a bunch of media sources in *Kodi,* from where it can pull content. These media sources can be local media on the card, removable USB drives plugged into the Pi, and even various file shares on the local network.

To define a media source, enter any of the Videos, Music or Pictures entries on the screen, and click Add Sources. Use the dialog box that pops up to browse to a source that contains some media. Adding media on the card or plugged-in USB drives is pretty simple and straightforward. But if you wish to pull in content from another computer on the local network, you have to define these network shares first. *Kodi* supports several popular file-sharing protocols, including all the popular ones such as *Samba,* NFS, AFP, FTP and more. To view media on a shared *Samba* drive, head to System > OpenELEC > Services, and toggle the Enable Samba option. If the source requires authentication, toggle the Use Samba Password Authentication option and enter the username and password. When you've added a source, you can tell *Kodi* about the type of content it houses. In return, *Kodi* lets you choose a scraper – a special plugin that fetches metadata about a media file from the internet.

If you've set up multiple OpenELEC HTPCs on the same network, they can also share libraries between them, using the UPnP protocol. On the HTPC that houses the content you wish to share with the other HTPC, head to Settings > Services > UPnP and toggle the Share Video and Music »

Quick tip

You can power the Pi from the USB port on any computer, similar to Google's Chromecast.

Quick tip

Use NFS instead of SMB to access media quicker.

» Libraries Through UPnP option. Now jump over to the other HTPC where you wish to view the content, and add a source, as described earlier. When you browse for a media source, select the UPnP Devices option from the list of sources, which then displays the other HTPC that houses the content.

Your HTPC now enables you to watch content either on locally connected drives or on any other computer or HTPC on the network, and even from your DVR. To further enhance the experience, you can enable the web interface to remotely control playback. *Kodi* includes a web server which allows you to control the player via a web browser. To enable it, head to Settings > Services > Webserver and toggle the Allow Control of Kodi Via HTTP option. You can optionally lock access behind a password. Once enabled, fire up a web browser on any computer on the network and navigate to the IP address of the HTPC to control playback.

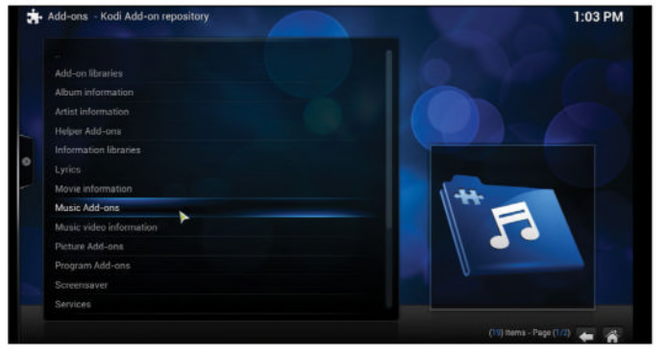
Kodi also produces official remote control apps for Apple, Android and iOS devices, and you can find several third-party ones for the Windows Phone as well. Before you can use them, head to Settings > Services > Remote Control, and toggle the Allow Program on Other Systems to Control Kodi option. Now head to your device's app store and grab a remote control app. The official app is called *Kore* on the Android Play Store.

» You can keep tabs on shows that are being recorded or scheduled for recording on your DVR.

Extend with add-ons

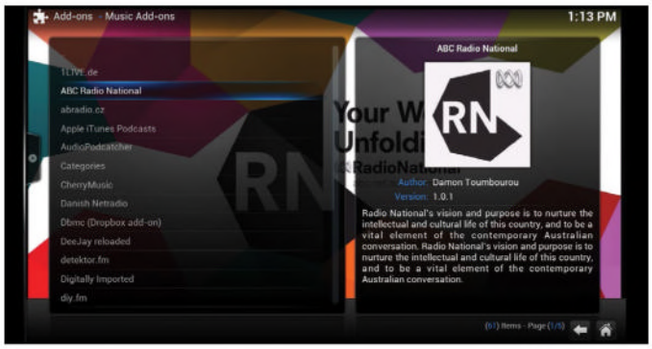
1 Select a repository

You can extend virtually every aspect of your HTPC by adding a number of plugins and extensions. To do this, head to System > Settings > Add-ons > Get Add-ons. This displays a list of repositories, including the official OpenELEC and *Kodi* repositories. Select the repository you wish to install from. There's also the All Add-ons option, which displays plugins from both these repositories.



2 Select category

Once you've selected a repository, you're shown a list of add-on categories. The *Kodi* repository includes a lot more categories than the OpenELEC one. The OpenELEC repo mostly includes drivers for various devices, while the *Kodi* repository includes well over a dozen categories. The Programs Add-ons category is particularly interesting and houses plugins that turn your HTPC into a seed box.



3 Select the add-on

When you select a category, you're shown a list of related add-ons. For example, if you select the Music Add-ons category, you're shown plugins for various online radio stations. Similarly, the Video Add-ons category houses plugins for popular video streaming websites, including YouTube, Vimeo and TED Talks. When you find a plugin you wish to use, select it and click the Install button.

4 Configure the add-on

You're returned to the list of plugins while *Kodi* downloads the one you selected. *Kodi* also installs and enables the plugin with the default options. Some plugins have optional configurable elements. To view these, click the Configure button associated with the plugin. The installed plugin is accessible from under the category it belonged to. For example, video plugins install under Video on the main page.

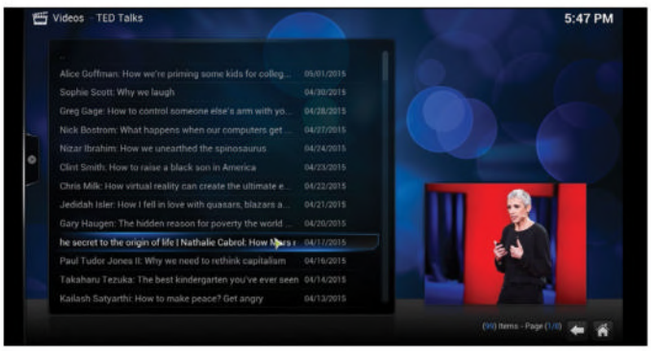
Must-have add-ons

1 BBC iPlayer

You can use this video add-on to watch content from the BBC iPlayer service on your HTPC. The add-on enables you to watch all BBC channels live or find content by browsing genres. The add-on also enables you to browse through popular and newest content, and brief highlights of the shows. Because BBC iPlayer is available only to UK residents, the plugin doesn't work on non-UK IP addresses.

2 Browse images

You can find lots of interesting pictures on websites such as Flickr and PicasaWeb. Head to the Pictures Add-ons section, which lists plugins that let you pull in images from these sites, and others such as the Hubble Space Telescope. Once enabled, different websites give you different options. For example, Flickr includes a list of interesting images of the day and runs a slideshow of them.

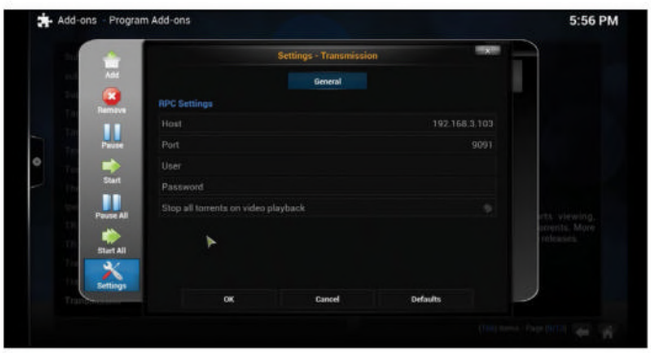


3 Watch online video

Similarly, there are lots of video streaming websites such as YouTube, Vimeo, NASA TV, TED Talks and so on. You can find add-ons for each of them and a lot more inside the Video Add-ons section. Each add-on lists videos according to the service it supports. For example, TED Talks lets you browse talks by topics or speakers, while both NASA TV and YouTube show live streams in addition to recorded videos.

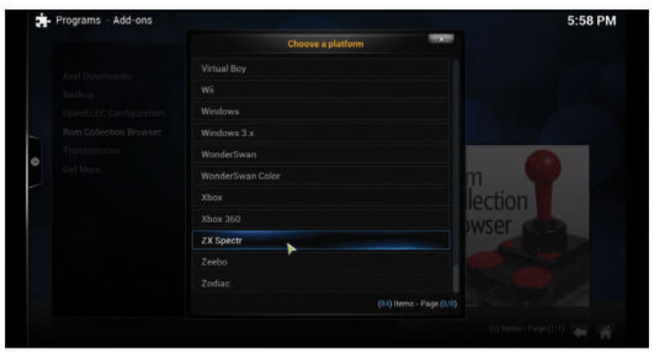
4 Control MPD

Besides remote-controlling content on the HTPC, you can control playback on other machines as well. So if you have an MPD server running on a computer (or a Raspberry Pi) somewhere on the local network, you can install the *MPD Client Audio* add-on on your HTPC. Once installed, configure it to point to your MPD server and you get a nice interface to browse through the music and control playback.



5 Monitor downloads

Talking of remote-controlling, you can also use the HTPC to connect to, monitor and control downloads on a remote computer. The *Transmission Client* add-on available under Programs can connect to another *Transmission* client on the network. For this to work, you need to enable the Remote Control feature in the *Transmission* client on the desktop, under the Edit > Preferences > Remote tab.



6 Play classic games

Another interesting add-on under the Program is the *ROM Collection Browser*. Once enabled, the add-on takes you through a wizard to set itself up according to your environment. It then imports your ROMs, scrapes metadata about them from online sources, then lets you play them. Setting it up is rather involved, so go to **https://code.google.com/p/romcollectionbrowser/wiki/FirstUse** to get it to work.

Whatsapp: Build a PiBot monitor

Interact with and control your Pi via popular messaging platforms.

Do you use the Raspberry Pi for a headless project, such as a media player, NAS server, seed box or security camera? If you do, the Pi is probably tucked away somewhere that's not easily accessible. You can always log in to it remotely but how do you monitor it in real time? How do you know whether it's overheating? Or running out of disk space? In this project, we'll play God and make your Pi self-aware, and give it the ability to communicate.

In more earthly terms, we'll install the *sendxmpp* tool on the Raspberry Pi, which allows it to communicate via the popular XMPP messaging protocol. We will then use it to send notifications to us via instant messages whenever a predetermined event is triggered.

First up, get a XMPP IM account for the Pi. If you aren't using a XMPP server already, you can register with any of the publicly listed XMPP servers (**https://xmpp.net/directory.php**). We're using the **Jabber.hot-chilli.net** service, which gets a top-notch security rating from XMPP.net and allows you to register an account on the website itself. Once you've registered an account for your Raspberry Pi, make sure you add it as a friend on your regular account, on which you want to receive notifications.

Now log into the Pi, update the repos and then download the *sendxmpp* tool with **sudo apt-get install sendxmpp**. It's a Perl script and will pull in the required Perl dependencies. When it's installed, create a file named **.sendxmpprc** under

```
pi@raspberrypi: ~/yowsup
pi@raspberrypi ~/yowsup $ yowsup-cli demos --yowsup --config config
Yowsup Cli client
==================
Type /help for available commands

[offline]:/L
Auth: Logged in!
[connected]:/message send 919968139981 "This is a test message sent from the Raspberry Pi"
[connected]:Sent: 1432009220-1
[connected]:
Iq:
ID: 2
Type: result
from: 918375972443@s.whatsapp.net


[919968139981@s.whatsapp.net(19-05-2015 04:21)]:[1432005783-1]   Got it!
Message 1432005783-1: Sent delivered receipt
[connected]:
[connected]:
```

There's not much documentation for the Yowsup library, but it does ship with some useful example apps, which make for an interesting study.

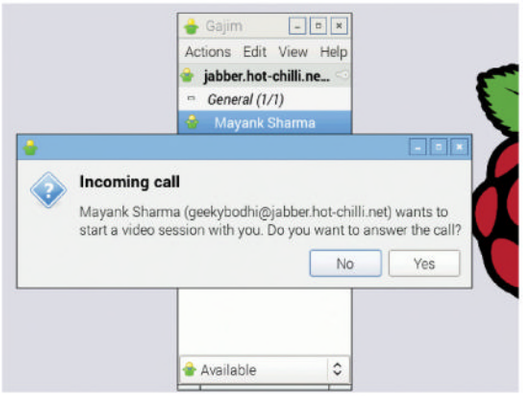
Video chats

If you're using the Raspberry Pi 2 as a regular desktop, you can install an IM client and converse with your friends either via text or video. The XMPP protocol we've discussed in this tutorial is one of the most popular protocols for exchanging instant messages, and is used by services such as Google Chat. One of the best XMPP clients for the Raspberry Pi, with the right balance of size and features, is *Gajim*. It's available in the official Raspbian repos, and you can install it with a simple command:

```
$ sudo apt-get install gajim
```

Once installed, you can hook it up to your XMPP account and exchange text messages with your friends. You can also hook up a webcam to the Pi and make video calls to your buddies. But for that you'll first have to make sure you've got the **gstreamer1.0-plugins-bad** and the **python-farstream** packages. Both are available in the official Raspbian repos and can be installed with the *apt-get* package manager. Once you have installed the webcam and the required components, your chat window now features a button that enables you to start a video session.

If you have trouble transmitting audio or video, head to Edit > Preferences > Audio/Video and make sure you're using the correct audio output device. If you're on a limited connection, you can also reduce the video size by picking a lower resolution from the Video Size pull-down.



› ***Gajim*** **has several interesting plugins, such as OTR, which lets you encrypt all your IM communications.**

your home directory with the credentials of the Pi's XMPP account, such as:

```
$ nano ~/.sendxmpprc
rpibot@jabber.hot-chilli.net    my-secret-password
```

Remember to replace the username and password with the credentials for the account you registered for the Pi. After saving the file, you can send a message to the Pi:

```
$ echo "Hi, this is Pi!" | sendxmpp -t geekybodhi@jabber.hot-chilli.net
```

The above command sends a message from the Pi to the XMPP ID specified with the **-t** option. Swap out the ID in the example above with your own XMPP ID. If you're signed into your regular IM account, you receive the greeting as a regular message from the Pi's XMPP account.

You can also pass output of *Bash* commands, such as:

```
$ echo "It is" $(date) | sendxmpp -t geekybodhi@jabber.hot-chilli.net
```

This command sends the output of the date command. Here's another example that's a little more useful:

```
$ echo $(/opt/vc/bin/vcgencmd measure_temp) | sendxmpp -t geekybodhi@jabber.hot-chilli.net
```

This command queries the temperature sensors on the Pi using the utilities installed by the **raspberrypi-firmware-tools** package, which we then pipe to our regular IM user. You can use this statement to monitor the Pi and send you an alert over IM when the temperature crosses a preset threshold. Copy the contents of Listing 1 or nab it from **http://pastebin.com/NdQw5frt** [*see page 43*] in a file called **status.sh**. Then set a crontab entry by running **crontab -e** and entering the following line:

```
*/5 * * * * ~/status.sh
```

Here we are asking the Pi to run the **status.sh** script every five minutes. Remember to change the location of the **status.sh** file to the location on your Pi. So what's in the **status.sh** script? The script stores the temperature of the Pi in a variable named **temp** after stripping out the verbose text and the decimal, because *Bash* can only handle integers. The script then checks whether the value is greater than 40°C, and if it is, alerts us with a message. You can extend this script to keep track of the goings-on in the Raspberry Pi. For example, you can ask it to send you alerts whenever it finds a particular message in a log file, or whenever the status of a daemon changes. The *sendxmpp* script helps you keep track of the activities on the Pi – however, you can't act on them without logging in to the Pi. But what if this isn't possible? What if you get a temperature warning from the Pi monitoring your home while you're away at work? Wouldn't it be great if you could control the Pi via messages as well?

Your best buddy

WhatsApp is one of the most popular messaging apps and you can use it with the Raspberry Pi. The Yowsup Python library enables you to use your *WhatsApp* account to exchange messages with your contacts. After the novelty of messaging your friends from the Pi wears off, you can use the Yowsup library to monitor and control the Pi by sending messages via *WhatsApp*.

Before you can install the Yowsup library, fetch its dependencies with **sudo apt-get install git python-dev libncurses5-dev**. Then use **git clone git://github.com/tgalal/yowsup.git** to download the library under the current directory, and install it with:

```
$ cd yowsup
$ sudo python setup.py install
```

Once the library has been installed, it's time to register your mobile number with *WhatsApp*. In the **yowsup** directory, create a file called **mydetails** with the following:

```
$ nano mydetails
cc=44
phone=447712345678
```

The **cc** option points to the country code, which is 44 for the UK. Replace it and the phone number with your particulars. Make sure you don't enter the + symbol.

Then save the file and use the following to ask *WhatsApp* for a registration code:

```
$ python yowsup-cli registration --config mydetails --requestcode sms
```

After a few seconds, you should receive an SMS on the mobile phone with the SIM card for the mobile number that you've entered in the **mydetails** file. The message contains a six-digit code. Use this code to register the mobile phone number with *WhatsApp:*

```
$ python yowsup-cli registration --config mydetails --register xxx-xxx
```

»

» Replace **xxx-xxx** with your code. After a second or two, you receive a response from *WhatsApp* on the Pi that looks something like this:

```
status: ok
kind: free
pw: jK0zdPJ9zz0BBC3CwmnLqmxuhBk=
price: 0.89
price_expiration: 1434674993
currency: EUR
cost: 0.89
expiration: 1463544490
login: 448375972334
type: new
```

The only bit of information we're interested in is the password mentioned with the **pw** variable. Copy it and paste it in the **mydetails** file, which should now read:

```
cc=44
phone=447712345678
password=jK0zdPJ9zz0M8G3CwmnLqmxuhBk=
```

That's all there's to it. The Yowsup library includes a demo app, which you can use to send and receive messages. Bring it up with:

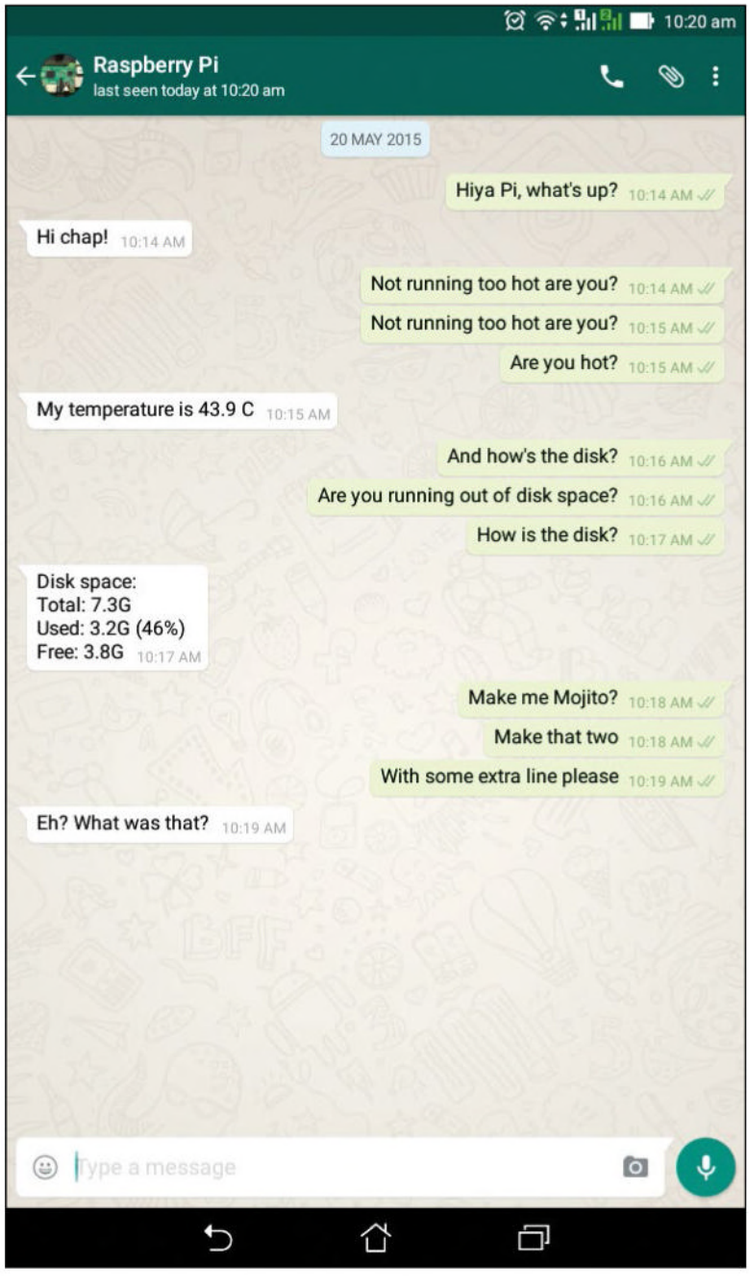
```
$ yowsup-cli demos --yowsup --config mydetails
```

This brings up the Yowsup CLI client. Type **/help** to see all the available commands. The **[offline]** prompt indicates that you aren't connected to the *WhatsApp* servers. Use the **/L** command, which picks up the authentication information from the **mydetails** file and connects to the server. The prompt now changes to **[connected]**.

You can now send messages to other *WhatsApp* users. To send a message to 449988776655 enter:

```
/message send 449988776655 "Hiya mate, I'm sending this from the Raspberry Pi!"
```

If the recipient responds with a message, it is displayed on the console on the Raspberry Pi. To end the session, use the **/disconnect** command to quit.



› The Raspberry Pi doesn't always receive your messages and, unfortunately, will not make you a mojito, even if you ask nicely.

What's up Pi!

The real advantage of the Yowsup library is that it can be used to invoke actions on the Pi. For example, you can send a *WhatsApp* message to check certain details on the Pi, such as its disk space or temperature, then maybe update it or shut it down. You can also use it to influence the GPIO pins and control any connected peripherals – a door, for example.

You can use the Python script in Listing 1 *(opposite)* to interact with the Pi. The script listens to messages from a certain predefined number, recognises certain keywords and responds accordingly. So if you send something like "Hiya Pi", it greets you back. If it receives a message that begins with "memory", the Pi executes the **df -h .** command and messages you the results.

The script uses classes created by Italian blogger Carlo Mascellani. They are housed with two files, named **wasend.py** and **warecieve,py**, which you can download with:

```
$ wget http://www.mascal.it/public/wasend.py
$ wget http://www.mascal.it/public/wareceive.py
```

In the same directory, create a file called **pitalk.py** with the contents of Listing 2 *(opposite)*. Now create a shell script called **talktome.sh** that calls the **pitalk.oy** Python script:

```
$ nano talktome.sh

#!/bin/bash
while :
do
sudo python /home/pi/yowsup/pitalk.py
done
```

Now make it executable with **chmod +x talktome.sh** and make sure it runs automatically whenever the Pi boots up by pointing to it in the **/etc/rc.local** file:

```
$ sudo nano /etc/rc.local

/home/pi/yowsup/talktome.sh
```

Save the file and reboot the Raspberry Pi, and the script starts automatically.

Parsing the script

Let's break down the script to understand it better. The **credential()** function at the top helps connect the script to the *WhatsApp* server by using the credentials for your

account. Make sure you edit both the parameters in this function. The **Answer()** function specifies the *WhatsApp* number our Pi communicates with. This is important because we don't want just anybody to control our Pi.

Then we define the functions that do the actual task we query the Pi for via the *WhatsApp* messages. For example, the **Refresh()** function refreshes the repository list and **Restart()** reboots the Pi. The **Temp()** and **Disk()** functions are a little more complex. The former fetches and truncates the temperature information, as illustrated earlier in the tutorial. Similarly, **Disk()** formats and rearranges the output of the **df -h** command for easier reading.

In the main part of the program (the **while** loop), the script waits for a message, and when it gets one, it raises a **MessageReceived** exception. The received message begins with a phone number followed by a message, such as "449876543210Message".

When it raises the exception, the script first converts the whole string to lowercase with the **value.lower()** method. It then checks whether the message is from the number it's supposed to respond to. If it isn't, the script appends it to a log file and doesn't respond.

If, however, the phone number is correct, the script then strips the number from the message and just leaves the textual bit. The **If** conditions then parse the message to decide how to respond. We've used different types of matching to give you an idea of what's possible. The first two look for matching characters at the start of the text. For example, **if received[:4]=="hiya": Answer("Hi chap!")** is triggered if the first four characters of the message are **h**, **i**, **y** and **a**, and responds with "Hi chap!" This condition is met even if the message it receives is, "Hiya Raspberry Pi, are you online?" The second also looks for matching characters at the beginning of the message but is triggered if it finds either of the two strings (**restart** or **reboot**).

The next three do a different kind of matching. They're triggered if their corresponding text is in any part of the message and not just at the beginning. So if you send a "What's the status of your disk?" message, the script picks up the "disk" keyword and triggers the **Disk()** function. Similarly, if you send a "You're not running too hot, are you?" message, the script picks up the "hot" keyword and responds with a readout from its temperature sensor. If it fails to pick up any keywords, the scripts just responds with the "Eh? What was that?" message.

You can extend this script for a whole variety of home automation tasks. You can even hook up the Pi camera module and use the Python Picam library to take pictures or videos and send them to you via a *WhatsApp* message. Check the Yowsup library's wiki page (**https://github.com/tgalal/yowsup/wiki**) for some examples of rolling the script into usable apps.

Listings

Listing 1: status.sh

```
#!/bin/bash

temp=$(/opt/vc/bin/vcgencmd measure_temp | cut -c6-7)

if [ "$temp" -gt 40 ]; then
   echo Whoa! My temperature is up to $(/opt/vc/bin/vcgencmd measure_temp). Power me down for a bit! | sendxmpp -t geekybodhi@jabber.hot-chilli.net
fi
```

Listing 2: pitalk.py

```
import os, subprocess, yowsup, logging

from wasend import YowsupSendStack
from wareceive import YowsupReceiveStack, MessageReceived

def credential():
    return "447712345678","jK0zdPJ9zz0BBC3CwmnLqmxuhBk="

def Answer(risp):
    try:
       stack=YowsupSendStack(credential(), [(["447668139981", risp])])
       stack.start()
    except: pass
    return

def Refresh():
    Answer("Refreshing the repos.")
    os.system("sudo apt-get -y update")
    Answer("Repos updated.")
    return

def Restart():
    Answer("Rebooting")
    os.system("sudo reboot")
    return

def Temp():
    t=float(subprocess.check_output(["/opt/vc/bin/vcgencmd measure_temp | cut -c6-9"], shell=True)[:-1])
    ts=str(t)
    Answer("My temperature is "+ts+" C")
    return

def Disk():
   result=subprocess.check_output("df -h .", shell=True)
      output=result.split()
    Answer("Disk space:\nTotal: "+output[8]+"\nUsed: "+output[9]+" ("+output[11]+")\nFree: "+output[10])
   return
while True:
    try:
          stack=YowsupReceiveStack(credential())
          stack.start()
    except MessageReceived as rcvd:
          received=rcvd.value.lower()
          if received[:len("447668139981")]=="447668139981":
                received=received[len("447668139981"):]
                if received[:4]=="hiya": Answer("Hi chap!")
    elif received[:7]=="restart" or received[:6]=="reboot": Restart()
      elif "disk" in received: Disk()
      elif "hot" in received: Temp()
elif "refresh" in received: Refresh()
          else: Answer("Eh? What was that?")
      else: #message from wrong sender
                with open("/home/pi/whatsapp.log","a") as mf: mf.write("Unauthorised access from: "+received[:len("919968139981")]+"\n")
```

Ubuntu 15.04: On the Pi 2

Can you use Ubuntu on the Raspberry Pi as an everyday desktop? Follow these tips and you certainly will be able to.

The release of the Raspberry Pi 2, equipped with a new quad-core ARMv7-based processor, brought with it a host of new supported distributions, including the popular desktop distro, Ubuntu. Soon after the announcement, third-party Ubuntu images, which were compiled to exploit the new architecture, began popping up on forum boards across the internet. While the official Ubuntu image *(see box on page 46)* is a lightweight command-line version of the distro, the unofficial images were much fuller, complete with a desktop and a host of applications. One of the best Ubuntu images currently available is put out by the Ubuntu Mate project. The image is based on the latest Ubuntu 15.04 release and uses the Mate desktop v1.8.2. Additionally, the desktop includes a large number of apps and can be easily fleshed out using the bundled Ubuntu Software Center. What makes it even more special is the fact that this image isn't just for demonstration purposes but is very usable on the rejuvenated Pi 2.

You can grab the image as a compressed archive from Sourceforge at **http://sourceforge.net/projects/ubuntu-mate/files/15.04/armhf/**. The latest version available at the time of writing is **ubuntu-mate-15.04-desktop-armhf-raspberry-pi-2.img.bz2**. Once you've downloaded the archive, you can extract the .img file out of it with:

```
$ bunzip2 ubuntu-mate-15.04-desktop-armhf-raspberry-pi-2.img.bz2.
```

The developers suggest using the *ddrescue* tool to transfer the image on to your card. Install this tool with:

```
$ sudo apt-get install gddrescue
```

Now insert the card – which needs to be at least 4GB in size – and use the **lsblk** command to find its mount point. Assuming that it's mounted on **/dev/sdb**, transfer the image file to the card with:

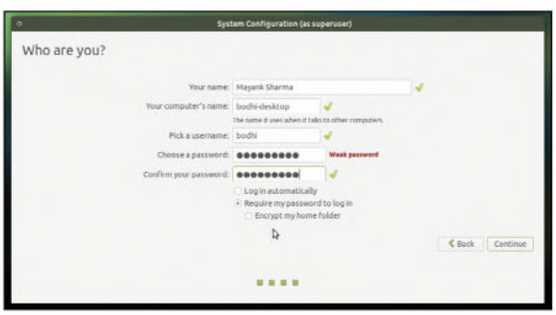
```
$ sudo ddrescue -d -D --force ubuntu-mate-15.04-desktop-armhf-raspberry-pi-2.img  /dev/sdb
```

If you are on a Windows computer, you can use *7-Zip* to extract the image file from the archive, which you can then transfer on to the card with *Win32 Disk Imager*. The Raspberry Pi lacks a BIOS, so if you wish to edit its hardware configuration, you have to manually edit the **config.txt** file *(see box opposite)*. Now plug the card into your Raspberry Pi 2 and power it up.

Initial configuration

The first time the distro boots up, it brings up a four-step installation and configuration wizard. In the first step, you are asked to select the default language for the distro, followed by the time and the keyboard layout. In the final step, you are asked to create a user account. This screen should be familiar to anyone who's previously installed Ubuntu on the desktop. You can also choose to toggle the option that enables you to log directly into the desktop. There's also the option to encrypt your home folder but we advise you against enabling this because the option adds a processing overhead on the limited resources.

Once you've configured the distro, it is installed on to the card with these settings. When it's done, you're dropped at



Ubuntu Mate doesn't ship with a preconfigured user account so it asks you to set one up as soon it boots up.

Quick tip

You can select which audio device *Omxplayer* should output audio to. For HDMI: **omxplayer -o hdmi video.mp4** For 3.5mm audio jack: **omxplayer -o local video.mp4**

Editing BIOS settings

The Pi doesn't include a BIOS setup utility, but you can tweak the various system configuration parameters by manually editing a text file. On a running Ubuntu Mate installation, you'll find this file under the boot directory (specifically **/boot/config.txt**). Open the file in a text editor with **sudo nano /boot/config.txt**. You can also edit the file while the distro isn't running. For this, remove the card from the Pi and connect it to a PC running Linux, Windows or Mac OS X. Mount the card and use the file manager to look for the **config.txt** text file. Irrespective of how you access the file, its contents are the same. The file is divided into various sections. The hash symbol (**#**) in front of the options is used to comment out or disable that option. You can only specify one option per line.

If you plug in the Pi to your HDMI monitor or TV and get no picture, scroll down to the **#hdmi_safe=1** option and uncomment it by removing the hash symbol. Then save the file and restart the Pi. When you do get the display but it doesn't stretch across the screen, find **#disable_overscan=1** and uncomment it.

Adventurous users can also overclock the Pi from this file. By default, the Pi runs at 700MHz but you can safely run it at 800MHz. While many people run their Pi at over 1GHz, make sure you take the necessary precautions, including installing a heatsink. To overclock the Pi, scroll down to the bottom of the file and uncomment the **arm_freq=800** parameter. Upon restart the Pi is now running at 800MHz.

the Mate desktop login screen, from where you can log in to the desktop. Before going any further, however, you should resize your installation image to take over all the remaining free space on the card. Unlike Raspbian, which includes a CLI configuration tool, which does this for you, on the Ubuntu Mate image you have to resize the partitions manually. Fire up the Mate terminal and launch the *Fdisk* command-line disk partitioning utility:

```
$ sudo fdisk /dev/mmcblk0
```

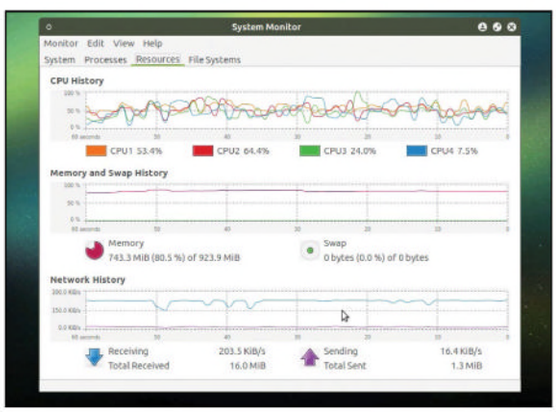
Use the P key to print the current partition table. We'll now delete the second partition and recreate it to take over the entire card. Press D and, when asked, enter 2 to delete the second partition. Now press N, followed by P and 2, to create a new primary partition. Now *Fdisk* asks you to enter the physical dimensions of the partition. Don't sweat – just press Enter twice to select the default options. When you're back at the main menu, press P again to view the two partitions. Taken together, the size of both these partitions should equal the size of the card. When you've verified this, press W to save the partition table and exit the *Fdisk* utility. Now close the terminal and reboot the device.

When you're back up again, head to the terminal and enter the following command to extend the filesystem:

```
$ sudo resize2fs /dev/mmcblk0p2
```

The command takes some time to complete, depending on the size and the speed of the card. Unlike the desktop version, Mate for Raspberry Pi doesn't include a swap space. If you are using a large SD card greater than 8GB, install the following to automatically create a 2GB swap space:

```
$ sudo apt-get install dphys-swapfile
```



The distro handles load surprisingly well. It's still impressively responsive with 20 *Firefox* tabs, *LibreOffice Writer* and *Transmission* all running at once.

That's a fully-fledged Ubuntu Mate 15.04 running on the Raspberry Pi 2.

Again, this step takes some time as the utility earmarks space for use as a swap space. The final bit of configuration is to check whether the module for the Raspberry Pi's audio hardware has been enabled or not. To do this, enter the following in a terminal:

```
$ lsmod | grep snd_bcm2835
```

If the command prints no output, you have to manually load the module with:

```
$ sudo modprobe snd_bcm2835
```

To make sure the module is loaded on subsequent boots, enter the following in a terminal:

```
$ echo "snd_bcm2835" | sudo tee -a /etc/modules
```

Using the desktop

If you're used to the Raspbian distro, the Mate edition will be a pleasant surprise. Not only does the desktop look different, but it's also flush with applications. The Mate desktop is the continuation of the Gnome 2 desktop, which also makes it lighter than mainstream desktops, such as Gnome, KDE and Unity. To complement its lightweight disposition, the distro uses resource-friendly apps including the *Caja* file manager, *Pluma* text editor, *Eye of MATE* graphics viewer, *Atril* document viewer, and more.

Surprisingly, the distro also includes popular feature-rich apps you'd find on a regular desktop distro, such as the *Firefox* browser, *Thunderbird* email client, *Rhythmbox* audio player and the *LibreOffice* office suite. Even more surprising is that these apps make good use of the extended processing power of the new Raspberry Pi 2 and perform rather well. Although we haven't stress-tested the distro or the device, Mate 15.04 on Raspberry Pi maintained its impressive performance through several hours of usage. One app that isn't of much use, though, is the *VLC Media Player*. Despite »

Quick tip

Use Ubuntu's meta desktop installation packages to install new desktops. For example, **sudo apt-get install ubuntu-desktop** or **sudo apt-get install xubuntu-desktop** to install the LXDE and Xfce-based Ubuntu desktops respectively.

adding support for hardware acceleration with the GPU on the Pi, playback with the popular media player isn't very impressive, especially with high-res videos. Instead you should use the command-line *Omxplayer,* which works very well and is also installed on the distro.

If you want a graphical client for *Omxplayer,* you can install the browser-based *OmxWebGUI.* The advantage of this client over others is that you can use it to control playback from any computer on the network. Although it's still in the early stages of development, it works perfectly and hasn't ever misbehaved on any of our installations.

Before you can install the client, you need to fetch its dependencies with:

```
$ sudo apt-get install git php5-cli
```

Now fire up a terminal and fetch the client with:

```
$ git clone https://github.com/brainfoolong/omxwebgui
```

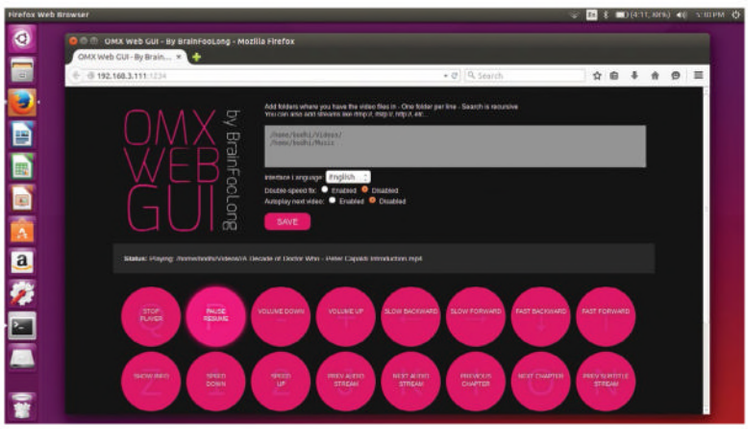
This creates a directory called **omxwebgui** under your home directory. Then enter the following command:

```
$ php -S 0.0.0.0:1234 -t ~/omxwebgui  >  /dev/null 2>&1  &
```

This command creates a simple PHP web server that's listening for connections on port 1234 of the Pi. Assuming the IP address of the Pi is **192.168.3.111**, fire up a web browser on the Pi or on any computer on the network and navigate to **192.168.3.111:1234**. This brings up the graphical interface for *Omxplayer.* Use the text box at the top of the page to specify the path to the folder that houses your media files and click Save. Then click on the status bar and select the file you wish to play. You can control playback using the buttons on the interface.

Quick tip

The Ubuntu Mate image uses the EXT4 filesystem instead of the Flash-friendly f2fs because the latter can't be resized. Follow the instructions on the project's website to build an image that uses f2fs.



Even if you aren't turned off by the CLI, use the *Omxplayer* web client to control playback from a remote device.

To further enhance your multimedia experience, you can also equip the web browser on the Pi to handle Flash content. Because Adobe doesn't release new versions of the Flash player for Linux, we'll use Google's *Pepper* Flash player instead. For this to work, though, you have to install the *Chromium* web browser first. Fire up a terminal and type:

```
$ sudo apt-get install chromium-browser  chromium-codecs-ffmpeg-extra
```

Then download the *Pepper* plugin ripped by a Raspberry Pi forum board member. The latest version at this time is v15.0.0.2.152, and you can download it with:

```
$ wget -c http://odroidxu.leeharris.me.uk/PepperFlash-15.0.0.152.r2-armv7h.tar.gz
```

Now you need to extract the plugin into *Chromium's* plugin directory with:

```
$ sudo tar zxf PepperFlash-15.0.0.152.r2-armv7h.tar.gz -C /usr/lib/chromium-browser/plugins/
```

Finally, edit the browser's configuration file to make it aware of the plugin:

```
$ sudo nano /etc/chromium-browser/default
```

Add the following line:

```
CHROMIUM_FLAGS="--ppapi-flash-path=/usr/lib/chromium-browser/plugins/libpepflashplayer.so --ppapi-flash-version=15.0.0.152 -password-store=detect -user-data-dir"
```

That should do it. Now launch *Chromium* and enter **chrome://plugins** in the address bar, which lists the enabled Flash plugin next to the others. While you can now browse Flash content on the Raspberry Pi, you can't use it to connect to streaming services such as Netflix.

Set up for remote access

The Ubuntu Mate image doesn't include the *OpenSSH* server, so you can't manage it from a remote machine out of the box. To rectify this, head to the terminal and enter:

```
sudo apt-get install openssh-server
```

You can now securely log into the Pi from any machine on the network. The command **sudo ssh mayank@192.168.2.111** asks for authentication information for the user account named mayank before logging you in. Modify the command to change the username and the IP address of the Pi according to your installation and network.

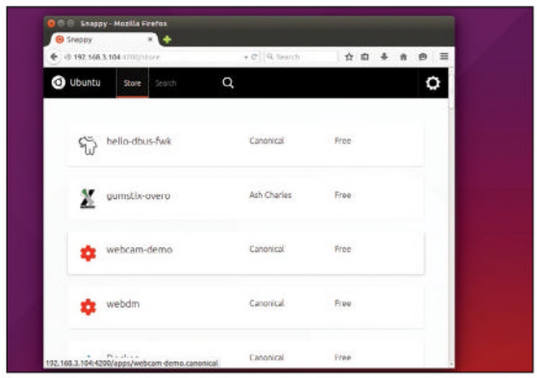
The new Pi also has enough resources to pipe your desktop to a remote machine via the VNC protocol. If you

Official Ubuntu for Raspberry Pi 2

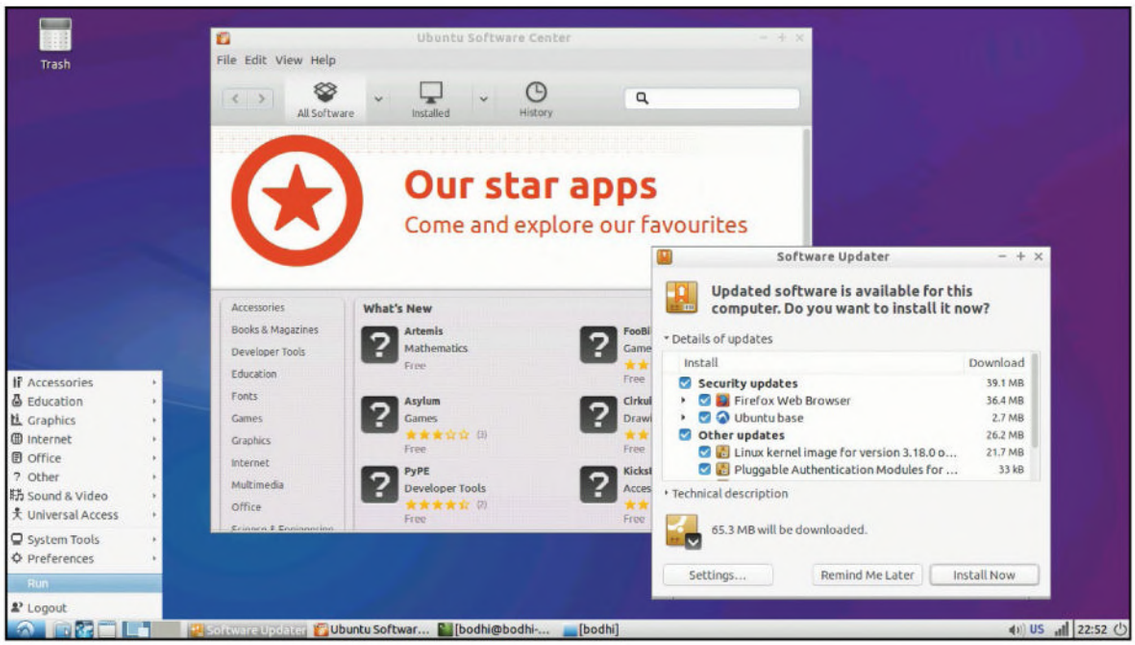
The official Ubuntu release for the Pi 2, dubbed Ubuntu Snappy, is a minimal server image that uses the same libraries as the mainstream release. Besides missing a graphical desktop, the biggest difference in Ubuntu Snappy is that it doesn't use the *apt-get* package management system. Instead it manages packages with the new containerised system, which Canonical claims to be more "snappier" and gives the distro its name.

Although it's meant for developers, you can take Snappy for a spin by downloading it from **www.raspberrypi.org** and transferring it to a card like any other Pi distro. After booting it up, type **snappy info** to get details about the installed packages, which you can update with **snappy update-versions**. If you get a certificate error, set the date and time using the format mmddhhmmyyyy.ss. For example, **sudo date 050411342015.00** sets the date and time to 4 May 2015, 11.34pm.

The distro is still under development and supports a limited number of packages. You can search through them with **snappy search <package-name>**. To simplify the task, install the *WebDM* web-based package manager with **snappy install webdm**. Once installed, navigate to port 4200 on the RPi IP address to view Snappy's app store.



You can find various guides on the internet that explain how to turn this basic image into something more useful.



You can use the distro to install individual pieces of software from the Ubuntu Software Center, and even complete desktop environments.

wish to enable remote desktop on the Ubuntu Pi, first install a desktop that's even lighter than Mate. While we recommend the LXDE desktop, you can also use Xfce. To install the LXDE desktop, enter the following in a terminal:

```
$ sudo apt-get install lxde
```

This command installs the vanilla LXDE desktop. You could instead install a customised version of the LXDE desktop produced by the Lubuntu distro with:

```
$ sudo apt-get install lubuntu-desktop
```

This command installs both the vanilla and the customised flavours of the LXDE desktop. After you've installed the desktop, you can install the *VNC* server with:

```
$ sudo apt-get install tightvncserver autocutsel
```

The command installs the *TightVNC* server and the *Autocutsel* package, which lets you share the clipboard between the local and the remote machine. After installing the *VNC* server, start it and then kill it, like so:

```
$ tightvncserver :1
$ tightvncserver -kill :1
```

Doing this creates the **~/.vnc/xstartup** configuration file. Open the file in a text editor and edit it so it looks like this:

```
#!/bin/sh

xrdb $HOME/.Xresources
xsetroot -solid grey
#x-terminal-emulator -geometry 80x24+10+10 -ls -title
"$VNCDESKTOP Desktop" &
#x-window-manager &
# Fix to make GNOME work
export XKL_XMODMAP_DISABLE=1
#/etc/X11/Xsession

autocutsel -fork

openbox &

/usr/bin/lxsession -s Lubuntu -e LXDE &
```

Now save the file and then start a fresh instance of the VNC server with:

```
$ tightvncserver :1
```

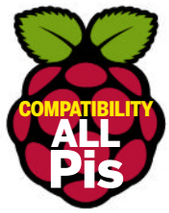
Now that your VNC server is online on the Pi, head to another machine on the same network and install a *VNC* viewer on it, such as *Xtightvncviewer*, with **sudo apt-get install xtightvncviewer**. Launch the *VNC* viewer client and enter the IP address of the *VNC* server on the Pi, such as **192.168.3.111:1**. This should bring up the remote LXDE desktop running on the Pi. In our tests, the Ubuntu Mate image could easily host two VNC sessions on the Raspberry Pi 2 without any noticeable dip in performance.

The Raspberry Pi was conceived as a low-cost computer. Thanks to the efforts of distributions such as Raspbian, the Pi managed to fulfil its maker's vision to quite an extent. But it's usage was limited to running lightweight apps and for specialised hobbyist applications, such as an HTPC. However, the enhanced juiced-up version 2 has raised the game to a whole new level. If the Ubuntu Mate image, which runs effortlessly, is any indication, the Raspberry Pi can now finally be classified as a true mini PC.

Here you can see the Lubuntu desktop on the Raspberry Pi 2 being piped to a remote Linux Mint desktop.

Tor: Set up a Wi-Fi hotspot

Configure a Raspberry Pi as an access point that routes all your private traffic over the anonymous Tor network.



Do you use *Tor* to prevent big brother from tracking you online? Although it is pretty straightforward to use, it can be quite a hassle to configure *Tor* on all your Internet-enabled devices. You can save yourself a lot of hassle by using a Raspberry Pi as an anonymised wireless access point. The Pi will dole out an IP address and any device that's connected to it will be able to access the Internet via the *Tor* network. To get this project up and running, you'll need a Raspberry Pi along with an SD card with the Raspbian distro. If you haven't done this before, follow the walkthrough to get Raspbian up and running. You'll also need an Ethernet cable. Hook one end into the Pi's Ethernet port and the other into your wireless router. This is how the Pi will connect to the Internet. You'll also need a USB Wi-Fi adaptor that's compatible with the Raspberry Pi. If you haven't got one yet, check the list of compatible adapters that are known to work on the Pi (**http://elinux.org/RPi_USB_Wi-Fi_Adapters**).

Access Point Pi

Once you've setup the Pi, you can configure the Pi from a remote machine via SSH. For the rest of the tutorial, we'll assume the IP address of your Pi is **192.168.2.100**. Fire up a terminal that's connected to the same router as the Pi and enter

```
ssh pi@192.168.2.100
```

to connect to it. After authenticating yourself into the Pi, use

```
iwconfig
```

to make sure the wireless adaptor is recognised by the device. Now refresh its package list with

```
sudo apt-get update
```

and install the software that will make it act as an access point with:

```
sudo apt-get install hostapd isc-dhcp-server
```

When it's installed, it's time to set it up. Begin by editing the **/etc/dhcp/dhcpd.conf** file that controls the DHCP and automatically assigns IP addresses to all connected devices. Open it in the *nano* text editor with

```
sudo nano /etc/dhcp/dhcpd.conf
```

and comment out the following two lines by adding a # in front of them, so that they read:

```
#option domain-name "example.org";
#option domain-name-servers ns1.example.org, ns2.example.org;
```

In the same file, scroll down and uncomment the word authoritative; by removing the **#** in front.

Then scroll down to the end of the file and add the following lines:

```
subnet 192.168.12.0 netmask 255.255.255.0 {
range 192.168.12.5 192.168.12.50;
option broadcast-address 192.168.12.255;
option routers 192.168.12.1;
default-lease-time 600;
max-lease-time 7200;
option domain-name "local";
option domain-name-servers 8.8.8.8, 8.8.4.4;
}
```

In these lines we define the IP address of our Pi access point (192.168.12.1), the range of the IP addresses it'll hand out to connected devices (from 192.168.12.5 to 192.168.12.50) as well as the address of the domain name servers (8.8.8.8 and 8.8.4.4). You can change any of these values as per your preference. Save the file (Ctrl+X) once you're done.

Setting up a static IP

We'll now edit the **/etc/default/isc-dhcp-server** to specify the interfaces that our new DHCP server should listen to. Open the file and scroll down to the line that reads **INTERFACES=""**. Insert **wlan0** between the quotes so that it now reads **INTERFACES="wlan0"**, and save the file.

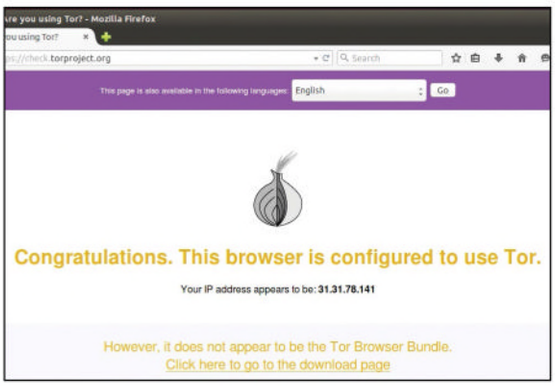
Now we'll setup the wireless adaptor (wlan0) and give it a static IP address. First, deactivate the wireless adaptor with:

```
sudo ifdown wlan0
```

command and then open the **/etc/network/interfaces** file. In the file, comment out every existing entry associated with

Quick tip

If you get Locale errors when connected to the Pi remotely, make sure you don't forward your locale by editing **/etc/ssh/ssh_config** and commenting out the SendEnv LANG LC_* line.



It takes more than *Tor* to stay anonymous. Make sure you read the documentation on the Tor Project's website.

wlan0, such as:

```
# iface wlan0 inet manual
# wpa-roam /etc/wpa_supplicant/wpa_supplicant.conf
# iface default inet dhcp
```

Then add the following lines below the line that reads **allow-hotplug wlan0** to set the static IP address for the new access point:

```
iface wlan0 inet static
address 192.168.12.1
netmask 255.255.255.0
```

Save the file and activate the interface with

```
sudo ifconfig wlan0 192.168.12.1
```

Make your point

Now that we've defined the wireless access point it's time to configure it. Create a new file called **/etc/hostapd/hostapd.conf** with the following contents:

```
interface=wlan0
ssid=TorSpot
hw_mode=g
channel=6
macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0
wpa=2
wpa_passphrase=$$Your_Passphrase$$
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
rsn_pairwise=CCMP
```

We've setup a password-protected network called TorSpot. You can specify a different name for the access point by specifying it in the **ssid=** string. Also change the **wpa_passphrase=** string to specify a custom password. You'll need to enter this password to authenticate yourself to the Pi's access point.

Next up, we'll tell the Pi where to find this configuration file by pointing to it in the **/etc/default/hostapd** file. Open the file, find the commented out line that reads **#DAEMON_CONF=""** and uncomment and edit it to read **DAEMON_CONF="/etc/hostapd/hostapd.conf"**.

NAT setup

We now need to set up NAT to allow multiple clients to connect to the Pi's access point and route all their traffic through the single Ethernet IP. Edit the **/etc/sysctl.conf** file and at the bottom add the following line:

```
net.ipv4.ip_forward=1
```

Save the file and then enter

```
sudo sh -c "echo 1 > /proc/sys/net/ipv4/ip_forward"
```

to activate the forwarding. You'll now have to specify the routing rules that will connect the Ethernet port (eth0) that's connected to the internet and the Wi-Fi access point (wlan0) which is exposed to the devices within your network:

```
sudo iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
sudo iptables -A FORWARD -i eth0 -o wlan0 -m state --state RELATED,ESTABLISHED -j ACCEPT
sudo iptables -A FORWARD -i wlan0 -o eth0 -j ACCEPT
```

By default, these rules will be flushed when you restart the Pi. To make them permanent, first run:

```
sudo sh -c "iptables-save > /etc/iptables.ipv4.nat"
```

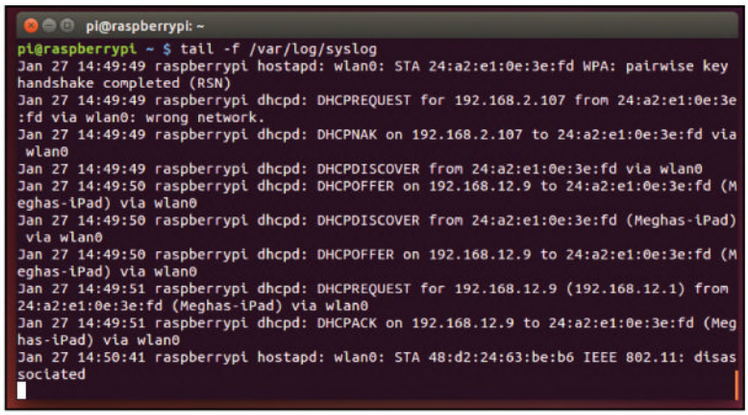
Then edit the **/etc/network/interfaces** file, scroll down to the very end and add

```
up iptables-restore < /etc/iptables.ipv4.nat
```

what this does is loads the rules when the devices are activated on boot.

Your Pi access point is now all set. To test it restart the DHCP server with

```
sudo service isc-dhcp-server restart
```



Use the tail -f /var/log/syslog common to keep an eye on the devices connected to your *Tor* hotspot.

Quick tip

Use the **tail -f /var/log/syslog** command to keep an eye on all system messages. This might come in handy if you are unable to connect to the Pi hotspot.

Your own hostapd

Sometimes even though a wireless adaptor works out of the box on the Raspberry Pi, it might throw errors when it's asked to serve as an access point. This is especially true of cards that use Realtek chipsets, like the one we've used – MicroNext MN-WD152B – which uses the RTL8192CU chipset. While it works right off the bat for browsing the web, it doesn't work with the *hostapd* client in Raspbian's repository. It turns out Realtek has its own version of *hostapd* client which you'll have to use in case you are in the same predicament as us.

To download the file, head to Realtek's download section (**http://bit.ly/RealtekWiFiDrivers**) and select your chipset from the ones listed. This takes you to a page that lists the drivers for your chipsets. From this page grab the driver for Linux, which will download a compressed zip file with a long-windy name. In our case this was called **RTL8188C_8192C_USB_linux_v4.0.2_9000.20130911.zip**. We'll just refer to it as **driver.zip**.

Copy this file to the Raspberry Pi using *scp* using something like:

```
scp driver.zip  pi@192.168.2.100:/home/pi
```

This copies the file to the Pi's home directory. Now extract the file with

```
unzip driver.zip
```

and **cd** into the **wpa_supplicant_hostapd** directory. It'll list several compressed tarballs. Use the **tar zxvf** command to extract the file beginning with **wpa_supplicant_hostapd**. Now **cd** into the **hostapd** directory under the extract directory. This directory has a file named Makefile. Open it in a text editor and replace the

```
CFLAGS = -MMD -O2 -Wall -g
```

line towards the top of the file with

```
CFLAGS=-MMD -Os -Wall -g
```

Save the file and enter *make* to compile the *hostapd* client. It'll take quite some time and when it's complete it'll replace the *hostapd* binary in this directory.

Before using this new version, move out the old version with:

```
sudo mv /usr/sbin/hostapd /usr/sbin/hostapd.orig
```

Then copy over the newly compiled version with the following:

```
sudo cp hostapd /usr/sbin/
```

And give it the right permissions with:

```
sudo chmod 755 /usr/sbin/hostapd
```

You should now be able to get your access point online without any issues.

» and manually enable the access point with our configuration with the following command [Read the *'Your Own Hostapd' box, p72*, if you get an unknown driver error]:

```
sudo /usr/sbin/hostapd /etc/hostapd/hostapd.conf
```

If everything goes well, the wireless access point (TorSpot) is listed in the list of available Wi-Fi hotspots. You can connect to it from another computer or a smartphone and authenticate using the password you specified in the **hostapd.conf** file. When connected, you should be able to browse the Internet normally.

Once you have tested the new access point, let's cement the settings so that they are activated as soon as the Pi boots up. Start the hostapd and DHCP services with the command:

```
sudo service hostapd start
```

and then use:

```
sudo service isc-dhcp-server start
```

commands and then update the init scripts with:

```
sudo update-rc.d hostapd enable
```

and finally:

```
sudo update-rc.d isc-dhcp-server enable
```

Now restart the Pi with

```
sudo shutdown -r now
```

When the Pi is back up again, you'll be able to connect to the new access point and browse normally.

Quick tip

We've used Google's DNS service in this tutorial, but you can use another service like OpenDNS or your ISPs DNS servers by pointing to them in the **/etc/dhcp/dhcpd.conf** file.

Torify access

Your Raspberry Pi is now fully functional as a wireless hotspot. However, the data is still not anonymised. So let's add *Tor* to the mix. SSH back into the Pi and install *Tor* with

```
sudo apt-get install tor
```

When it's installed, edit *Tor*'s config file **/etc/tor/torrc** and add the following at the top:

```
Log notice file /var/log/tor/notices.log
VirtualAddrNetwork 10.192.0.0/10
AutomapHostsSuffixes .onion,.exit
AutomapHostsOnResolve 1
TransPort 9040
TransListenAddress 192.168.12.1
DNSPort 53
DNSListenAddress 192.168.12.1
```

These settings inform *Tor* about the IP address of our access point and asks that it anonymises any traffic that flows over it. Next up, we'll change the routing tables so that connections via the Wi-Fi adaptor (wlan0) are routed through *Tor*. First, flush the existing redirection and NAT rules with:

```
sudo iptables -F
```

go on to:

```
sudo iptables -t nat -F
```

Since, we'll still want to be able to SSH into the Pi, we'll add an exception for SSH's Port 22 with:

```
sudo iptables -t nat -A PREROUTING -i wlan0 -p tcp --dport 22 -j REDIRECT --to-ports 22
```

We'll now add two rules. The first is a passthrough rule for DNS lookups and the second directs all TCP traffic to *Tor*'s port 9040:

```
sudo iptables -t nat -A PREROUTING -i wlan0 -p udp --dport 53 -j REDIRECT --to-ports 53
sudo iptables -t nat -A PREROUTING -i wlan0 -p tcp --syn -j REDIRECT --to-ports 9040
```

Like before, these rules won't be carried on to the next session. To load them on reboot, all you have to do is save them to the NAT save file like before with:

```
sudo sh -c "iptables-save > /etc/iptables.ipv4.nat"
```

In the previous section, we've already configured the **/etc/network/interfaces** file to load the contents of this file when the interfaces are activated.

You can now enable the Tor service with:

```
sudo service tor start
```

and update the relevant boot scripts with:

```
sudo update-rc.d tor enable.
```

That's it. Now restart the Pi. When it's back up again, you'll be able to connect to the Pi hotspot, TorSpot, as before. However, unlike as before all your traffic will now be routed through the *Tor* network.

You can verify that this is happening by heading to check **https://torproject.org** from any device that's connected to TorSpot. The page will also list your IP address which will not be that of your ISP. Visit this page from another device connected to TorSpot and it'll show a different address. Congratulations, you can now anonymously browse the web on all your devices!

```
Setting up torsocks (1.2-3) ...
Setting up tor-geoipdb (0.2.4.24-1) ...
pi@raspberrypi ~ $ sudo nano /etc/tor/torrc
pi@raspberrypi ~ $ sudo iptables -F
pi@raspberrypi ~ $ sudo iptables -t nat -F
pi@raspberrypi ~ $ sudo iptables -t nat -A PREROUTING -i wlan0 -p tcp --dport 22 -j REDIRECT
pi@raspberrypi ~ $ sudo iptables -t nat -A PREROUTING -i wlan0 -p udp --dport 53 -j REDIRECT
pi@raspberrypi ~ $ sudo iptables -t nat -A PREROUTING -i wlan0 -p tcp --syn -j REDIRECT --to-
pi@raspberrypi ~ $ sudo iptables -t nat -L
Chain PREROUTING (policy ACCEPT)
target     prot opt source               destination
REDIRECT   tcp  --  anywhere             anywhere            tcp dpt:ssh redir ports 22
REDIRECT   udp  --  anywhere             anywhere            udp dpt:domain redir ports 53
REDIRECT   tcp  --  anywhere             anywhere            tcpflags: FIN,SYN,RST,ACK/SYN r

Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination

Chain POSTROUTING (policy ACCEPT)
target     prot opt source               destination
pi@raspberrypi ~ $
```

› **Verify the traffic redirection rules with the sudo iptables -t nat -L command.**

Tor-in-a-box options

If you find this tutorial too cumbersome, or want to set up something for a non-technical friend or relative, there are several ready-made hardware solutions that can anonymise all their web traffic in a similar fashion.

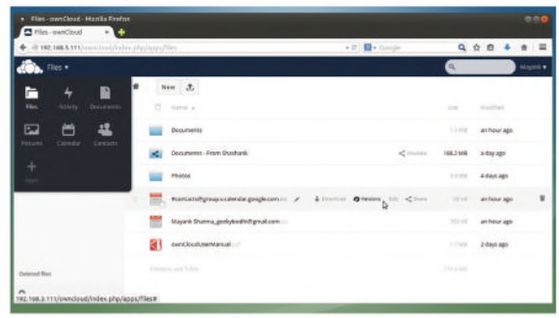
There's the OnionPi Pack from AdaFruit (**http://bit.ly/AdaOnionPi**) which includes a Raspberry Pi B+ and a compatible USB Wi-Fi adaptor along with a case for the Pi, cables, SD card and everything else you need to setup your Torrified Wi-Fi hitspot. The bundle costs $80. However, you'll still have to follow the instructions and set it yourself.

If you'd rather have something more plug and play, there's the SafePlug from the guys who bought us PogoPlug. It's a $49 device that plugs into your wireless router and once activated routes all traffic over the Tor network. A neater and smaller alternative is the Anonabox (**www.anonabox.com**). It initially launched on Kickstarter but after its funding was suspended it relaunched on Indiegogo. Here it was listed at $51 and surpassed its funding target in early January 2015 and will begin shipping in February 2015. Anonabox is a router that you can directly connect to via Wi-Fi or Ethernet.

Another router-based option is Portal which stands for Personal Onion Router To Assure Liberty. The project produces a pre-built software image for several TP-Link routers. You can simply flash the Portal firmware image onto these router following the instructions on the project's website (**https://github.com/grugq/portal**).

OwnCloud: Share and sync

Take charge of your data by creating your own cloud service.



› ***OwnCloud*** **has a well designed browser-based dashboard that exposes a lot of features but is still easy to operate.**

Do you want the convenience of an omnipresent Dropbox-like storage service without doling out wads of cash and your data to a third party? *OwnCloud* is one of the best pieces of open source software to help you create your own private and protected cloud-sharing service. Using *OwnCloud*, you can sync and share your private data, and access it from any device connected to the internet. For added security, *OwnCloud* can also encrypt your files. The software can handle files in a variety of formats and you can extend its usability by adding a number of other apps.

As with other online cloud storage services, you can sync files on *OwnCloud* either using the web browser or a desktop client on Windows, Mac and Linux, as well as mobile clients for Android and iOS devices. Furthermore, your *OwnCloud* server keeps older versions of all changed files and enables you to revert to an older version without much effort.

In this tutorial, we're setting up the *OwnCloud* server on top of the Raspbian distribution for the Raspberry Pi. The server software has modest requirements and it performs well even on the Raspberry Pi Model B in certain small and controlled environments, such as your house. You also need a USB portable disk for storing the data. For maximum reliability and performance, it's best to use a self-powered disk that doesn't draw power from the Raspberry Pi. Before you begin setting up the server, make sure the Raspberry Pi has a static IP address. The easiest way to do this is to tie an IP address to your Pi's unique MAC address in your router's admin page. We're assuming the Pi is at **192.168.3.111**.

Quick tip

To view some stats on the APC cache, copy its script to your **DocumentRoot** with **cp /usr/share/doc/php-apc/apc.php /var/www/** and then view it in the web browser on **http://localhost/apc.php**.

Install OwnCloud

Because Raspbian is based on Debian, we can pull in packages from *OwnCloud's* Debian repository. Fire up a terminal and add the *OwnCloud* repositories with:

```
$ wget http://download.opensuse.org/repositories/
isv:OwnCloud:community/Debian_7.0/Release.key
$ sudo apt-key add - < Release.key
```

You can now refresh the repositories with:

```
$ sudo apt-get update
```

Now install the *OwnCloud* server and all its required dependencies as follows:

```
$ sudo apt-get install owncloud.
```

This also pulls in and sets up the *MySQL* database, and you're asked to set up a root password.

In addition to installing the required components, the above command automatically configures the *Apache* web server to talk to the *OwnCloud* installation. You need to enable certain *Apache* modules for *OwnCloud* to work correctly. In a terminal, enter:

```
$ sudo a2enmod headers rewrite env
```

Then restart *Apache* using:

```
$ sudo apachectl restart.
```

You have to tweak the configuration file of PHP if you wish to upload files that are greater than 2MB in size. To do that, open the PHP configuration file, **php.ini**, housed under **/etc/php5/apache2**, in a text editor. Look for the **upload_max_filesize** and **post_max_size variables** and change their value from **2M** to something like **1024M** or even **2G**.

Optionally, on larger installations, you can also install the APC PHP accelerator to make the *OwnCloud* installation

snappier. Pull in the components with **sudo apt-get install php-apc** and then open APC's configuration file and add:

```
$ sudo nano /etc/php5/conf.d/20-apc.ini
extension=apc.so
apc.enabled=1
apc.shm_size=12M
```

Then bring the cache online by restarting *Apache* with:

```
$ sudo apachectl restart.
```

Now that the server is set up and configured, it's time to prepare the storage medium. Plug the USB disk into the Pi and enter **sudo blkid** in a terminal. The USB disk is probably be mounted as **/dev/sda1** if you don't have any other USB disks attached. Make a note of the corresponding UUID, which looks something like **6154-F660**. Now create a directory to mount this drive using:

```
$ sudo mkdir /media/owncloud
```

Then mount the drive with:

```
$ sudo mount -t vfat -o umask=007,auto,uid=33,gid=33 /dev/sda1 /media/owncloud
```

The above command assumes your drive has a FAT32 filesystem and is mounted at **/dev/sda1**. Once the drive is mounted correctly, you can edit the **fstab** file to make sure it's automatically mounted:

```
$ sudo nano /etc/fstab
UUID=6154-F660  /media/owncloud/  vfat
rw,umask=007,auto,uid=33,gid=33 0 0
```

Configure the cloud

That's all there is to installing the server components. You're now all set to configure your cloud. Launch a web browser and navigate to the *OwnCloud* installation instance at **192.168.3.111/owncloud**. Because this is a brand new installation, you are asked to create a new user account for the *OwnCloud* administrator.

Next, we need to ask *OwnCloud* to use the *MySQL* database and store files under the mounted USB drive. For this, click on the Storage & Database pull-down menu. Then enter **/media/owncloud/data** in the text box corresponding to the Data Folder entry and select the MySQL/MariaDB option in the Database section. You're asked to enter the connection details of the database serve, so just enter **localhost** as the host and **root** as the username, along with the password you configured when the database was pulled in along with *OwnCloud*.

That's it – you've set up *OwnCloud*. You can now log into your cloud server as the administrator using the credentials you have just created for *OwnCloud*. While you can start using the server to upload and download files straight away, let's take a moment to get the house in order. For starters, when you log into the *OwnCloud* server, click on the pull-down menu next to your username and click on Personal. Here you can change the settings for your account, such as the login password and display name. You can also add a profile picture and configure how you would like to be notified about certain actions. Also, if your cloud is going to be used by multiple people, it's advisable to add users and organise them into different groups. To do this, select the Users option from the pull-down menu. While adding users, you can restrict their storage space and even share your admin responsibilities with other users, and mark certain users as admins for a particular group.

Upload and share files

You're now all set to upload data into your *OwnCloud* server. After you've logged in, you are in the Files section. To upload a file, click on the arrow button. To organise files into folders, click on the button labelled New, and select the Folder option from the drop-down menu to create a new folder.

If you've uploaded a file in a format that *OwnCloud* understands, you can click on its name to view and edit the file. *OwnCloud* can visualise the data it houses in different views. For example, click on the Files pull-down menu in the top-left corner of the interface, and select the Pictures option. This view helps you view images in your cloud by filtering out all other types of content.

Another way to upload files to the server is by using the WebDAV protocol, with which you can access your cloud server from your file manager. For example, in the Files file manager, press Ctrl+L to enable the location area. Here you can point to your *OwnCloud* server, such as **dav://192.168.3.111/owncloud/remote.php/webdav**. Once authenticated, the *OwnCloud* storage is mounted and you can interact with it just like a regular folder.

To share uploaded files, go to the Files section in the web interface and hover over the file or folder you wish to share. This displays several options, including Share, which enables you to select which users or groups you want to share the item with and whether you want to give them permission to edit and delete the files. You can also share with someone who isn't registered with your *OwnCloud* server. Click on Share with Link, and *OwnCloud* displays a link to the item that you can share with anybody on the internet. You can also password-protect the link and set an expiration date.

While you can interact with the cloud using the web interface, it's far easier to use one of its official clients. *OwnCloud* has clients for all the major desktop and mobile »

Quick tip

If your drive has an NTFS partition, install the NTFS driver with **sudo apt-get install ntfs-3g** and use **-t ntfs-3g** in the **mount** command.

Ready-made solutions

Although it doesn't take too much effort to install and configure the *OwnCloud* server from scratch, there's a couple of ways to save time and effort. The chaps behind PetRockBlog have written a script that automates the whole installation process.

The script downloads and sets up an OwnCloud installation on top of a Raspbian distribution. However, unlike the tutorial, the script uses the *Nginx* web server instead of the *Apache* web server. To use the script, install the required components with:

```
$ sudo apt-get install git dialog
```

Then download the script with:

```
$ git clone git://github.com/petrockblog/OwncloudPie.git
```

which creates a directory called **OwncloudPie**. Move into this directory:

```
$ cd OwncloudPie
```

Make the script executable:

```
$ chmod +x owncloudpie_setup.sh
```

Then execute it:

```
$ sudo ./owncloudpie_setup.sh
```

Now give it some time to download all the components and configure your server. Once you've installed *OwnCloud* from the script, you can run it again to update the installation whenever a new *OwnCloud* version is released.

If you are the adventurous sort, you can install arkOS (**https://arkos.io/**) on your Raspberry Pi. In addition to *OwnCloud,* the distro has other apps to keep you in charge of your data.

Head to the downloads page, then download and extract the installer for the Raspberry Pi. Then insert an SD card and run the installer with:

```
$ sudo ./arkos-install
```

Follow the steps in the installer to download the image from the internet and install it on to your SD card. Once it's done, boot your Raspberry Pi from it and head to **http://arkos:8000** to set up your server.

» platforms. These clients also help you synchronise folders from the desktop to your *OwnCloud* server with ease.

Set up clients

Most desktop distros host the Linux client in their official repos. You can also grab the latest version by adding the corresponding repo for your distro from here: **http://bit.ly/1HZxhOy**.

The page has instructions for popular distros including Debian, Fedora, Ubuntu, OpenSUSE, and more. To download clients for other platforms, head to the downloads page on *OwnCloud's* website (**https://owncloud.org/install/#install-clients**). Mobile clients are best fetched from either Apple's App Store or Google's Play Store.

Once the client is installed, it prompts you for your login credentials in order to connect to the *OwnCloud* installation. Once connected, the Linux clients create a local sync folder named **owncloud** under the home directory, such as **/home/bodhi/owncloud**. Any files you move into this directory are automatically synced to the server. You can also specify one or more directories on a local machine to sync with the OwnCloud server. If a directory is shared with several users, when anyone makes a change to a file on one computer, it automatically flows across to the others.

When collaborating with other users, you'll appreciate *OwnCloud's* version control system, which creates backups of files before modifying them. These backups are accessible via the Versions pull-down option corresponding to each file, along with a Restore button to revert to an older version.

In addition to files, you can also sync your calendar and address book with your *OwnCloud* server. Follow the walkthrough opposite to enable the *Calendar* and *Contacts* apps. Once you've enabled both, the top-left pull-down menu now includes the Calendar and Contacts option.

Now you need to import your contacts and calendar from your existing apps into your cloud server. *OwnCloud* supports the popular vCard file format (which has the .vcf file extension) and almost every popular email app, including online ones such as Gmail, export their address books in this format. Similarly, calendars can be imported in the popular iCal format. Before proceeding further, make sure you download both the .vcf and .ical files from your existing contacts and calendar apps.

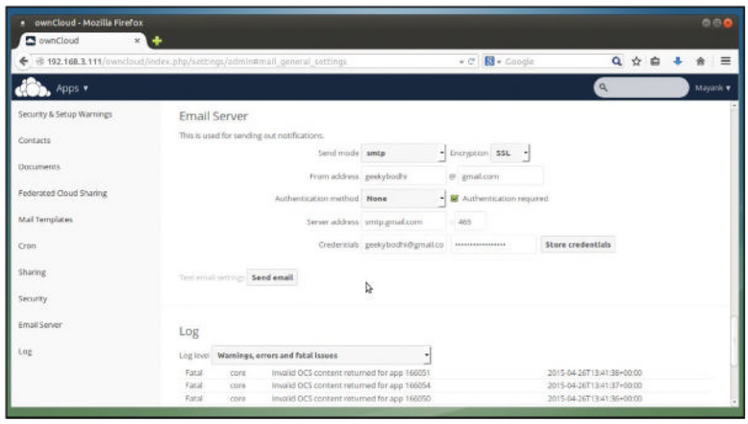
Now head to *Contacts* in *OwnCloud* and click on Import Contacts. In the pop-up window, click on Upload File and point it to the .vcf file. Once the contacts have been imported, you can sync them with your email clients using CardDAV links. Head to the Contacts section in *OwnCloud*, click on the gears icon at the bottom, hover over the name of the address book you imported and click on the Chain icon. This spits out a CardDAV link for this address book that you can feed to your desktop or mobile address book client.

Sync and share your calendar

Similarly, you can use *OwnCloud* to manage your calendar and tasks. To create an event in your calendar, head over to the *Calendar* app. You can view the calendar for the entire month or for the current week. To add a new event, click on the appropriate date in the calendar. This brings up a window, which gives you several options to configure the event. To import an existing client, simply upload the .ical file to your cloud server. When you click on the file in *OwnCloud's* web interface, the server recognises the file and offers to import it into an existing calendar or into a new one. Select the option that best suits you.

After you've imported the calendar, you can use *OwnCloud* to share it with other users. Click on the Share Calendar icon corresponding to the calendar you wish to share. This brings up a pull-down menu, which enables you to select the users or the group of users you wish to share the calendar with. Furthermore, just like address books, *OwnCloud* can also sync your calendars with desktop and mobile apps that can read this information from CalDAV links. To get the CalDAV link for your calendar, click on the Gears button and then on the Chain icon corresponding to the calendar you wish to sync. This displays the link that you can pass on to the clients to keep them in sync with the *OwnCloud* calendar.

There's a lot more you can do with *OwnCloud.* Follow the walkthrough opposite to flesh out the default installation with new apps to extend the functionality of your cloud.



› **Scroll down the Admin options to hook *OwnCloud* up with an email server.**

Universal access

The real advantage of commercial cloud services such as Dropbox is that you can access data stored within them from any computer connected to the internet. However, by default, a self-hosted *OwnCloud* installation is only accessible from computers and devices within the local network,

That's not to say that you can't access your private cloud from the internet, though. The trickier and expensive solution is to get a static IP address from your ISP and then poke holes in your router's firewall. Or, you can set up Dynamic DNS in your router or local machine. The smarter way, however, is to use a tunnelling service, such as PageKite. The service uses a pay-what-you-want model. As a non-commercial user, you can use the service for free by filling out a form once a month, telling PageKite how you use the service. But it's definitely worth more than the $3 per month minimum they request from individuals.

First you need to install *PageKite.* Launch a terminal and enter:

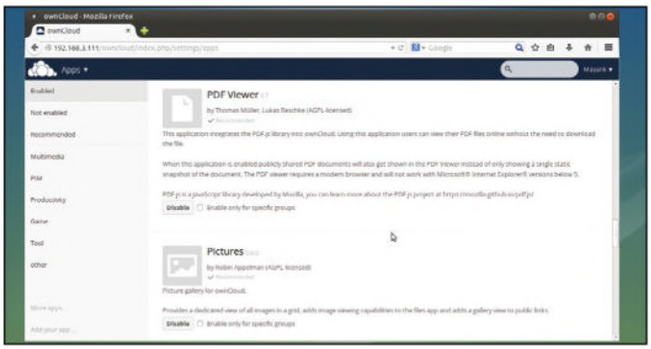
```
$ curl -s https://pagekite.net/pk/ | sudo bash
```

When it's done, make your local web server public with the following command:

```
$ pagekite.py 80 mycloudserver.pagekite.me
```

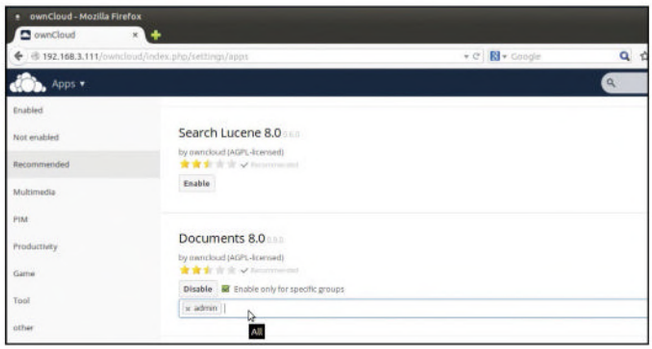
Remember to replace **mycloudserver** with the name you want for your *OwnCloud* server. Now you can access your own personal *OwnCloud* instance by heading over to **http://mycloudserver.pagekite.me** from any computer anywhere in the world. The first time you run this command, PageKite runs you through its brief sign-up process and asks for your email address.

Install and enable apps



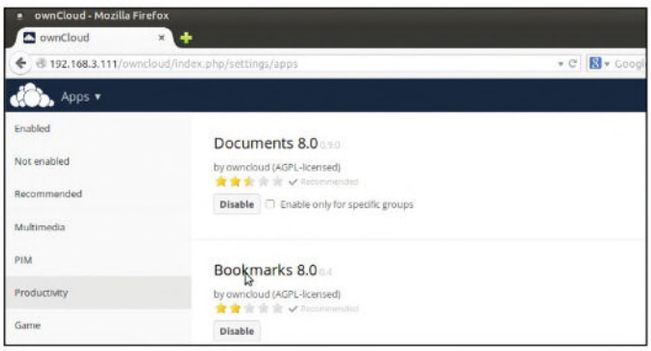
1 Enabled apps

You can extend your default *OwnCloud* installation by adding (or removing) a bunch of apps. Bring up the pull-down menu in the top-left of the interface and click on Apps. By default, you are shown a list of apps that are already enabled on your installation. You can browse through this list and read their descriptions to understand them better. You can also disable any enabled app from this section.



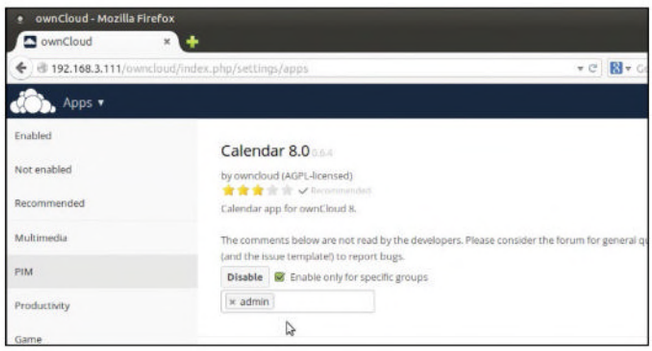
2 Documents app

Once you've browsed through this list, head to the Recommended section, which lists a number of apps that are recommended by the *OwnCloud* developers. These haven't been enabled by default as they might not be of use on all *OwnCloud* deployments. The *Documents* app enables you to edit and collaborate on documents in various formats within *OwnCloud* itself, without the need for an external app.



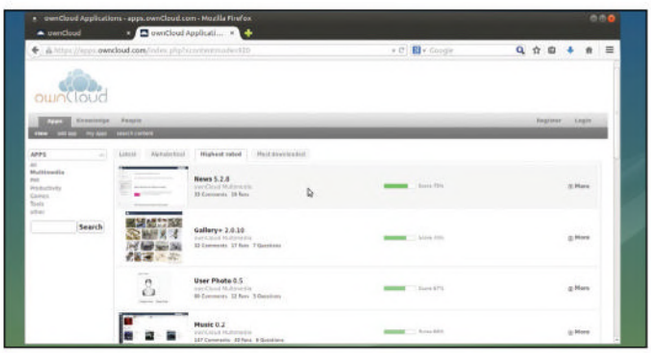
3 Bookmarks app

The other productivity app that you should enable is *Bookmarks*. This app lets you store and manage bookmarks in your *OwnCloud* server. You can add bookmarks by adding them directly, or by importing a bookmark file from your web browser. The app also has a bookmarklet that you can add to your browser's bookmarks. Press the bookmarklet to add a website to *OwnCloud's* list of bookmarks.



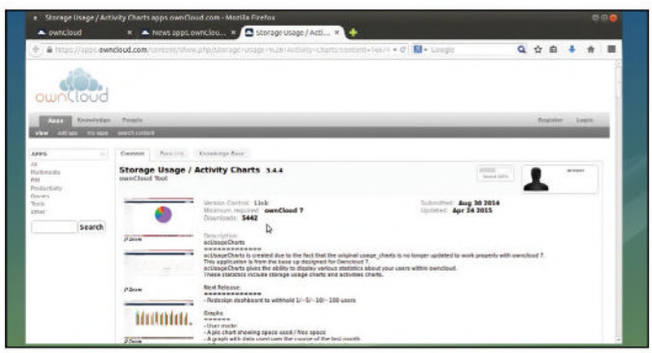
4 Calendar and Contacts

Scroll down and click on the PIM tab on the left. This section lists two apps. You can enable either or both the *Calendar* and *Contacts* apps. Once enabled, the apps let you pull in your existing contacts and calendars, which you can sync with the PIM apps from your *OwnCloud* installation, as explained in the tutorial. Some *OwnCloud* apps also have the option to enable them for specific users.



5 More apps

In addition to the apps listed in the Apps section on your *OwnCloud* installation, there are others that you can install from the *OwnCloud* website. Scroll down the Apps section and click on the More Apps... link. This takes you to the *OwnCloud* app store at **http://apps.owncloud.com**. You can download any app from here and extract it under the **/var/www/owncloud/apps** folder inside the Pi.

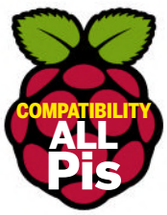


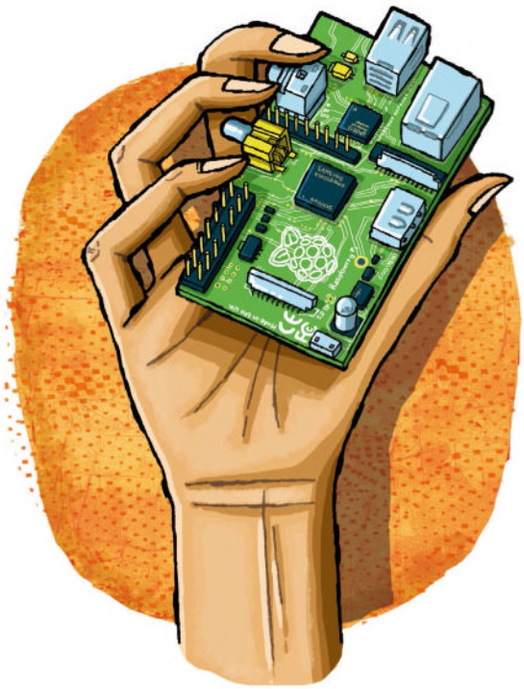
6 News and storage usage

Two other useful apps are the *News* and *Storage Usage* apps. The former is an RSS reader and the latter visualises the storage space on your cloud with a variety of charts. Once you've downloaded them from the website and extracted them under the **apps** folder, head back to the Apps section in your *OwnCloud* installation. These and any other downloaded apps are listed under the Not Enabled section.

PiMusicBox: Stream music

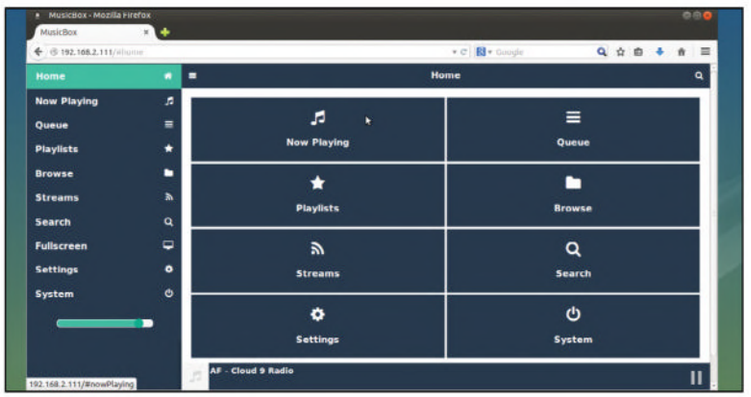
Stream music from the internet and across your network.





We've got music everywhere. In addition to DRM-free tracks on the hard disk, you've probably got hundreds of tracks on Spotify or Google Play. Bring them all together with the Pi MusicBox distro, which transforms the Raspberry Pi into the ultimate music player.

The secret sauce that powers Pi MusicBox is the *Mopidy* music server, which can fetch music from a variety of sources. The distro couples its powerful underpinnings with a nicely designed browser-based user interface to control playback and configure other aspects of the distro.



Pi MusicBox has a simple but intuitive browser-based interface.

In Pi MusicBox, you get a headless media server that can play music from a variety of sources, in all the popular audio formats, from a connected USB or a NAS device, as well as streaming internet radio. The distro is designed for audiophiles and is fully equipped to work with your hi-fi gear. It can output music through speakers attached to the headphone jack of the Pi, and also through the HDMI and USB ports. So all you have to do is install the distro, hook up some speakers to the Pi, plug in your account credentials and let it rip. You can then control your juiced-up Pi from any computer on the network and even from any Android device.

Begin by downloading the compressed image for the Pi MusicBox distro from **www.pimusicbox.com**. Extract the downloaded ZIP file and then put the IMG image file on your SD card with the **dd** command, such as:

```
$ sudo dd if=musicbox0.6.img of=/dev/sdd
```

Remember to replace **/d/ev/sdd** with the location of your SD card.

If you use the Ethernet port to connect the Pi to the internet, you can boot the Pi from the newly-created SD card. However, if you use a wireless card, you need to edit the distro's configuration file and manually point it to your wireless router. Access the newly-created SD card from a regular distro, navigate to the **config/** folder and open the **settings.ini** file in a text editor. Near the top you'll notice two variables: WIFI_NETWORK and WIFI_PASSWORD. Insert the corresponding values of your network next to these two variables and save the file. The caveat is that Pi MusicBox only works with WPA2-protected wireless networks.

Once that's done, boot the Pi with the configured SD card. If you have a monitor attached to the Pi, you can follow the booting process, otherwise wait a minute or two, then fire up a browser on any computer on your network and head to **http://musicbox.local**. If that doesn't take you anywhere, point your browser to the IP address of the Pi.

Configure MusicBox

The default interface of Pi MusicBox is rather bland because you haven't configured any music source yet. To fix that, click on the Settings link in the navigation bar on the left. This takes you to a page from where you can individually enable and configure all the supported streaming services, as well as tweak other settings.

Expand the Network setting to customise the distro's network-related parameters. For example, you can change the name of the workgroup the distro is listed under on the

The DNA of Pi MusicBox

The Pi MusicBox distro is based on the *Mopidy* music server, which in turn is based on a customised version of the *MPD* server. *MPD* stands for *Music Player Daemon*. It's a music player but unlike typical desktop music players, *MPD* uses a client-server model. Separating the music player into two components has a couple of advantages – it means that *MPD* uses few system resources, and it introduces interesting possibilities, such as remote playback and control and the ability to use different interfaces.

MPD can also handle audio files in a variety of formats, including Ogg Vorbis, Flac, MP3 and other formats supported by the FFmpeg library. It can also play Ogg and MP3 HTTP streams, can read and cache metadata, has native zeroconf support, and a built-in HTTP streaming server. This is why *MPD* is ideal for running on low-powered headless servers, while the client can run on any machine in the network. The *Mopidy* server's implementation of the *MPD* server inherits many features of the original *MPD* server, so you can control it remotely from any *MPD* client and search through your music.

Windows machines in the network. Also, if you want to move the Pi from being connected via an Ethernet cable to a Wi-Fi network, this section enables you to add the SSID and its password without tweaking the configuration file. Furthermore, you can also enable the option to connect to the distro via SSH. Unlike regular Raspberry Pi distros, you don't really need SSH access to the insides of Pi MusicBox. However, if you do decide to enable SSH access, make sure that you change the default root password by specifying a new one in the next section.

Next up is the MusicBox section, which houses some interesting settings. In the Device Name field, you can customise the network identity of the distro. This enables you to have several MusicBox instances running on the same network, such as kitchen.local, bedroom.local, garage.local, and so on. If there's an online music station you'd like the distro to play automatically upon boot, you can enter its link in the Autoplay URL field. The section also has two toggles that enable you to stream music from other devices to MusicBox *(refer to the walkthrough below to set this up)*.

Audiophiles should pay attention to the settings in the Audio section. The Audio Output option offers a pull-down menu to help you select the device from which the Pi should output audio. The default Automatic setting pipes audio to USB if USB audio is detected. If not, the audio is sent to HDMI, and then finally to the analogue output. The pull-down menu also lists several DACs (digital audio converters), including ones from HiFiBerry and IQ Audio. If you are using a compatible DAC, disable the Downsample USB setting, otherwise the distro turns down all audio to 44K.

By default, Pi MusicBox scans for new music files on the SD card it's installed on and on any attached USB disks. You can also ask it to scan for files under shared *Samba* folders on the network, by entering their location (such as **//192.168.2.10/Shared/Music**) under the Network Drive setting in the Music Files section. However, scanning for music could take a while, depending on the number of configured locations and the size of the USB disks. The distro also has the option to disable such scanning under this section. Finally, there's the Resize Filesystem option, which is in beta and might corrupt your installation. But if it works as intended, it resizes Pi MusicBox to take over the entire SD card, so you can use the extra space for music.

Quick tip

For better quality audio, use a Linux-compatible USB soundcard, also known as a digital audio converter, or DAC.

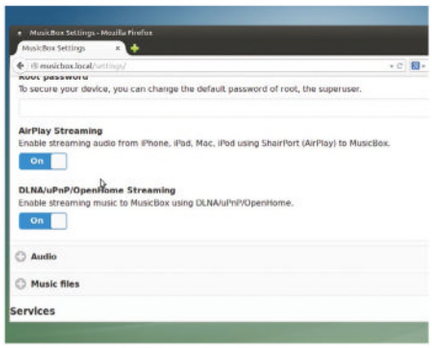
Quick tip

No matter what kind of speakers you are using, for best performance make sure they come with their own power supply and don't drain the Pi.

Configure services

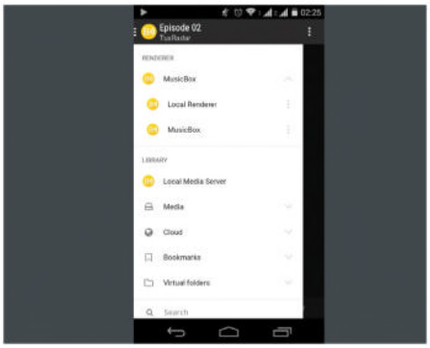
There are several ways to add music that you own to the Pi MusicBox distro, which you can then play via the Browse > Local Media option in the web frontend. You can fetch music from a shared *Samba* folder on the network by pointing to its location, as mentioned earlier. You can also attach one or several USB disks to the Raspberry Pi, and the distro adds music stored on them automatically. If you've expanded the »

Stream from mobile devices



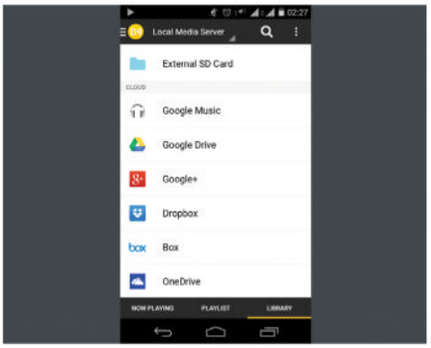
1 Enable streaming

Pi MusicBox also supports playback via the DLNA protocol. It's disabled by default, but once enabled, you can send music from devices such as your mobile phone to the Pi, which plays it from the attached speakers. To enable it, bring up the Pi MusicBox web interface, head to Settings > MusicBox and enable the DLNA Streaming option.



2 Install BubbleUPnP

This app is available from the Google Play Store as an ad-supported free download. Install the app and tap on the menu icon on the right. Your Pi MusicBox server should be listed under the Renderer section. Tap on it to set it up as the default renderer. From here on, any music you select to play on this mobile device is sent to the Pi for playback.



3 Stream music

You can now play any music stored on your mobile phone and it streams through the speakers connected to the Raspberry Pi. *BubbleUPnP* can also stream music stored on online services including Google Music, Dropbox, OneDrive and so on. Head to the Libraries section and tap on the service to equip it with your account settings.

SD card using the beta resizing option mentioned in the previous section, you can copy music directly on to the SD card as well.

To load music to the SD card, you can either connect it to your computer or access it via the network while it's attached to the Pi. The distro has a working *Samba* configuration and should show up in the Network section inside the file manager of all OSes. The distro's *Samba* share has a folder called Music. You can put any audio file inside this folder and it is transferred to the SD card.

In addition to local music, there's quite a handful of services from which Pi MusicBox can get audio files. Each of them can be configured separately from under the Services section. First up is the popular Spotify service. All you need to do is equip the distro with the authentication information for your Spotify Premium account, and it takes care of the rest.

The distro fetches all your playlists, which you can then access from the Playlists section in the main interface. If you are on a slower network or if your data usage is capped, use the Music Quality pull-down menu in the Spotify settings section and select the Lowest option, which consumes the least amount of bandwidth.

Pi MusicBox also has beta support for the SoundCloud service. Before you can enable the service, however, you need to fetch an authentication token by visiting **www.mopidy.com/authenticate** and logging in to your SoundCloud account. This spits out a long string of numbers that you then need to paste into the Token field under the SoundCloud settings.

Similarly, you can enable Google Play Music All Access by adding the login credentials for your account under the Google Music settings section. If the music from Google Play doesn't show up in the main interface, you need to add an alphanumeric Android Device ID to the configuration as well. You can fetch this by dialling ***#*#8255#*#*** on your Android phone or by installing the *Device ID* app (**http://bit.ly/LXFowncloud**) from the Google Play Store.

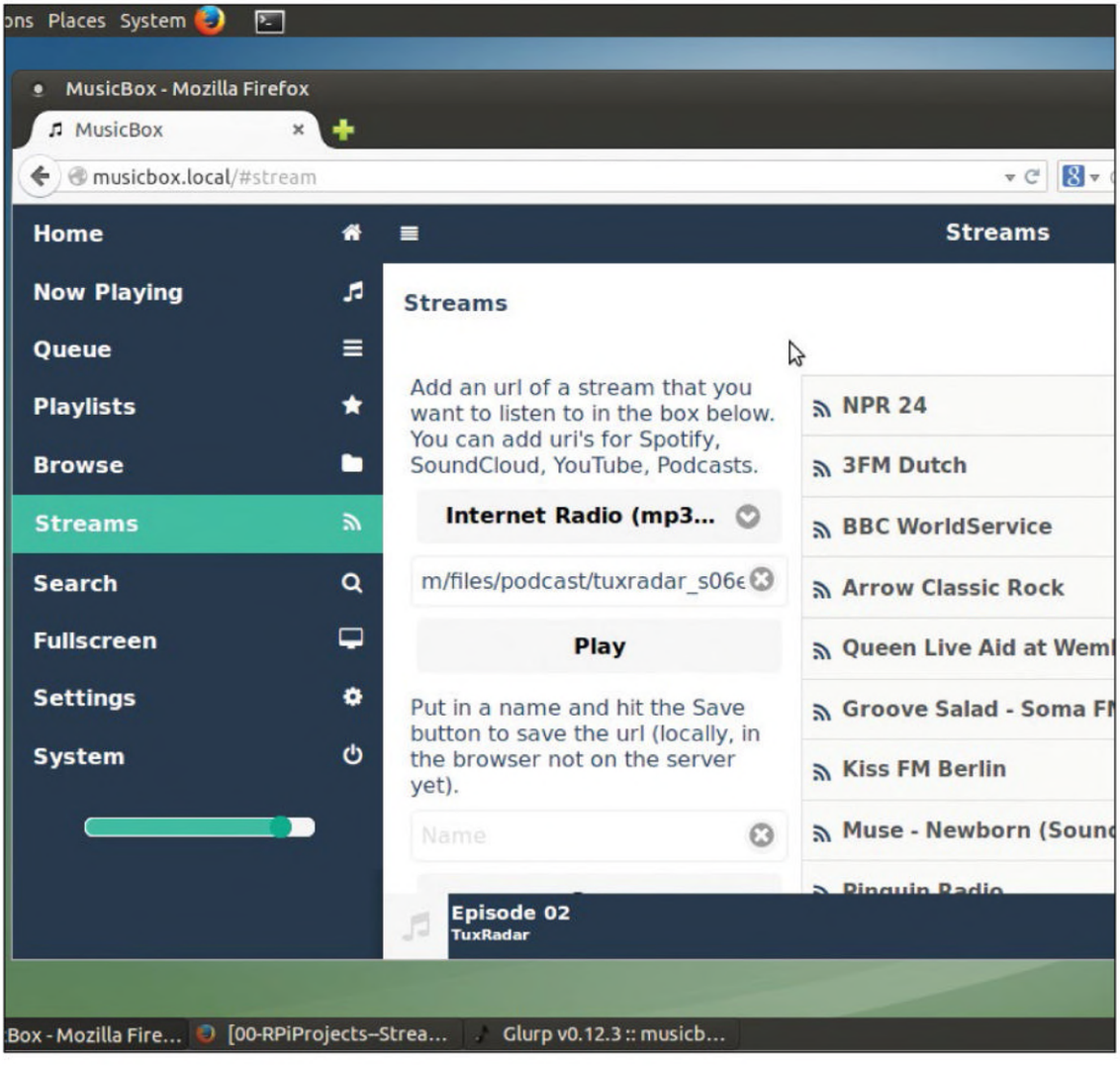
The distro also supports scrobbling via the Last.FM service and this is a wonderful way to discover new music. To enable it, just enter your login credentials in the Last.FM section. The distro can also play podcasts from iTunes and Gpodder.net, as well as online radio streams from Dirble, Soma FM and AudioAddict. To listen to them, all you need to do is enable them individually in the Settings page.

Playing audio

Once you have configured all the sources of music, you can head back to the main interface to listen to them. Switch to the Browse tab, which lists all the configured sources of music. Select any one and drill down through the genres or categories until you find the track you wish to play. When you click on it, the track or the radio station starts playing through

Quick tip

Audiophiles should consider using an add-on audio card from **www.hifiberry.com**.



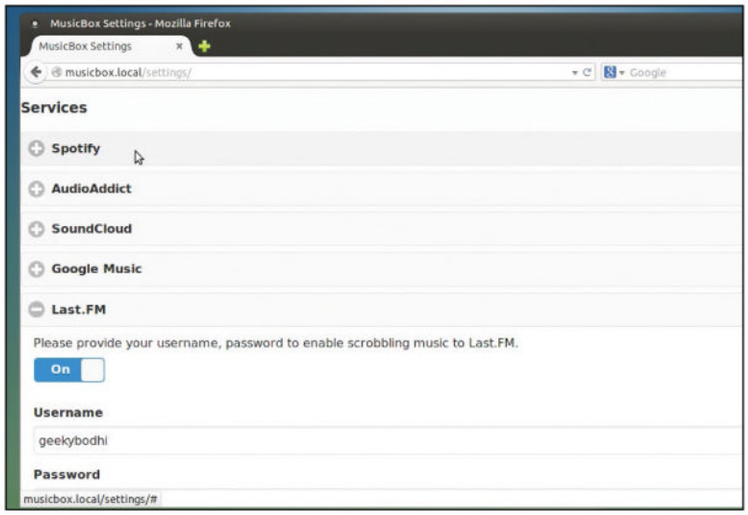
The distro ships with a large collection of online radio stations and also lets you add your favourites.

the attached speakers. Switch to the Now Playing tab to control playback.

The Playlists tab lists all your playlists from the Spotify service. To find specific music, switch to the Search tab and enter a string of text. By default, the distro looks for music matching the text in all the configured services. However, for faster results, you can instead ask Pi MusicBox to search inside a particular service only.

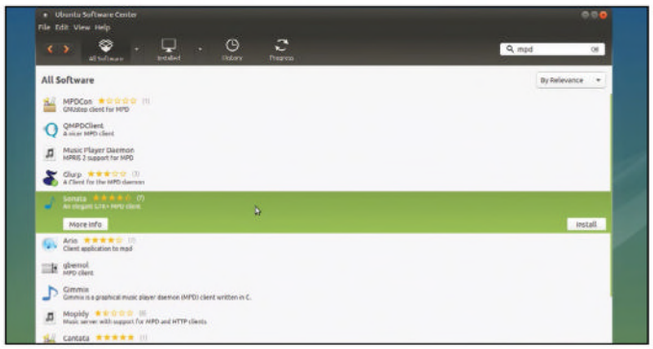
If there's a particular online stream you wish to play back, switch to the Stream tab and paste its URL into the Internet Radio box. When adding URLs, remember that you can't use links that point to container files, such as M3U, XSPF or PLS. You instead have to use links that point to a read stream, such as **http://www.tuxradar.com/files/podcast/tuxradar_s06e02.ogg** which is the URL for an episode of the TuxRadar podcast.

There are many more ways of interacting with the Pi MusicBox server besides the web-based interface. You can play music via any software that supports the *Music Player Daemon (MPD),* such as the *MPDroid* app for Android.



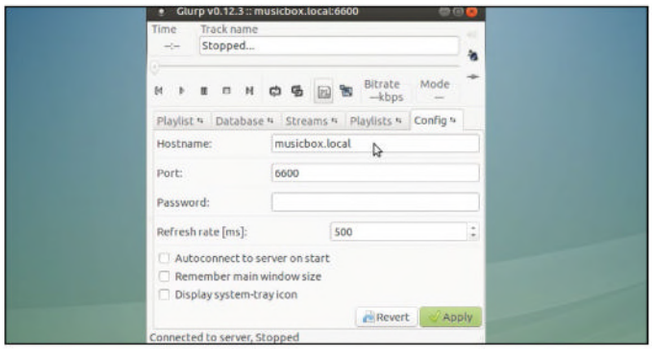
Every new version of the Pi MusicBox distro adds new services.

Connect with clients



1 Get a desktop client

Linux distros have several clients in their repos, including *Ario, Glurp, Sonata, QMPDClient, Cantata* and more. All apps are equally intuitive, so use the one that integrates with your desktop environment. Some clients, such as *Cantata*, are cross-platform and available for Mac OS X and Windows, which also has native clients, such as *Chimney.*

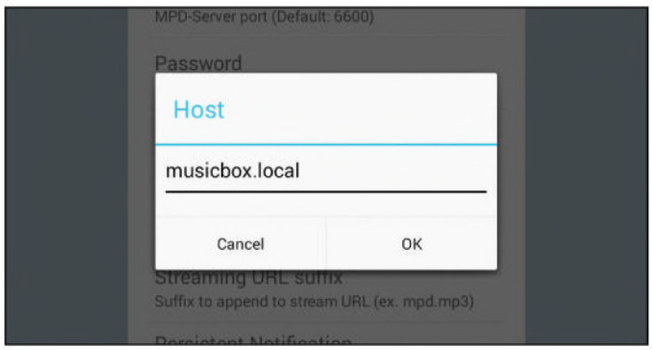


2 Control from desktop

All apps need to be hooked to your Pi MusicBox installation. Some ask you for details about your *MPD* server when you launch them for the first time, while others detect Pi MusicBox automatically thanks to zeroconf. When asked, enter **musicbox.local** as the hostname of the server and leave the other settings untouched.

3 Get an Android client

To control the *Mopidy* server from an Android device, you need an *MPD* client for it. The Google Play Store lists a bunch of *MPD* clients, including the popular *MPDroid* client. The app is available for free and has user interfaces for both phones and tablets. Using *MPDroid,* you can search your music collection and sort it according to categories such as Artists, Albums, Genres and so on.



4 Control from mobile

When you launch the app for the first time, you're asked to set it up. You have to first select a preferred WLAN connection and select your wireless network. In the following screen, tap on the Host parameter and then enter the IP address of the Pi MusicBox server. Now head back to the main screen, from where you can browse through and control your music collection.

Raspbian: Build a Pi 2 server

Uses the a fancy new Raspberry Pi 2 board to build a tiny but power-efficient server for your growing digital library.

Traditionally, servers are great big devices that consume vast amounts of power and generate a considerable amount of heat, and that is why data centres are huge air conditioned rooms full of racks housing multiple units. But for home or small business use, they don't have to be that way anymore, and with the Raspberry Pi we have a small Linux computer that consumes very little power and generates only a small amount of heat. In this tutorial, we will be using a Raspberry Pi 2 to build a small, powerful and energy efficient server that will serve files over SSH and be a handy remote backup location.

Aside from the Pi, you'll need to collect together a blank Micro SD card, a good-quality Raspberry Pi power supply, an Ethernet connection to router and an externally powered USB hard drive. We'll start by creating an SD card containing the operating system. Technically, this is a bundled collection of software based on the Linux kernel that's called a distribution or 'distro' for short. Our distro of choice is Raspbian, which is the default distro for the Raspberry Pi.

There are two ways of installing Raspbian to your SD card. First, the *NOOBS* method requires that you download the *NOOBS* ZIP archive and then extract the contents to a blank, FAT formatted card. Once this card is inserted into your Raspberry Pi, a menu system will ask you which distro to install; this is where you should choose Raspbian, and then grab a cup of tea while it installs itself. This is the easiest way of installing Raspbian, but for a smaller install footprint the best method is to download the Raspbian image and then extracting it from the archive, copying that to a blank SD card and using the command **dd**.

The **dd** command is powerful but dangerous. If used incorrectly, it can destroy all the data on a drive, so proceed with care. Before employing **dd**, we need to identify the SD card inserted into the computer. Open a *LXTerminal* (click on the icon on the desktop) and type the following:

```
$ sudo fdisk -l
```

You will now be able to see a list of the volumes/disks that **fdisk** can find. Typically, anything starting with '**/dev/sdX**' is an internal hard drive of your computer. Ignore those entries and look for '**/dev/mmcblk0XX**', where XX can be p1 or p2. You can ignore p1/p2, as they are partitions on the card – we will be writing data to the whole card, not just a partition – If you can see those, your SD card has been identified and make a note of the location.

With the SD card found, we use the **dd** command to copy the Raspbian image to the blank card. In the *LXTerminal,* navigate to the location where you extracted the Raspbian image from the ZIP archive; typically, this is **Downloads**.

```
$ cd ~/Downloads
```

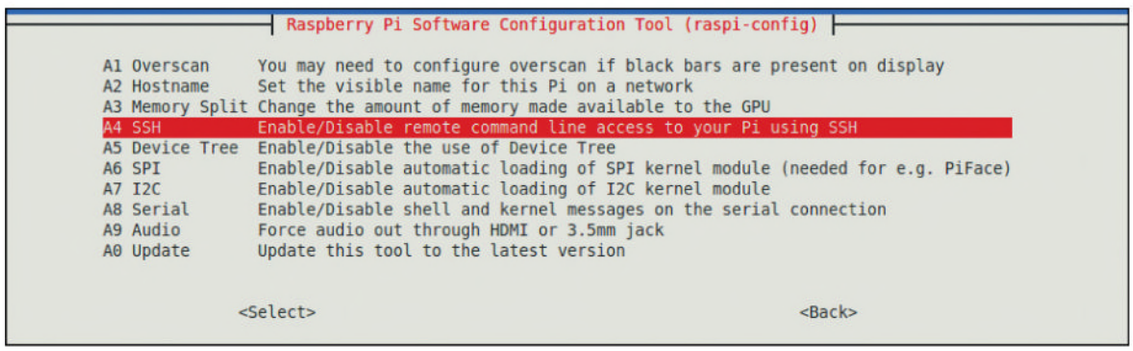
Now issue the **dd** command. Please double check everything before pressing enter, as once you start **dd** it won't give you a chance to stop the process. In the *LXTerminal*, issue the following command, replacing **raspbian.img** and **/dev/mmcblk0** to match what your system reported via the **fdisk -l** command.

```
$ sudo dd if=./raspbian.img of=/dev/mmcblk0 bs=4M
```

You will be prompted for your password and then **dd** will get to work, but you will see no output while it works; rather, it will do its job and then report after five to 10 minutes. Now is the time for a cup of tea.

Setting up Raspiconfig

With your SD card ready, lets assemble your Raspberry Pi setup. For this initial configuration step you will need to plug your Pi into a monitor and have a keyboard and mouse connected. Assemble your kit, insert the SD card and then



SSH is a secure protocol that provides a level of encryption to your connection, giving you a more secure method of transporting data over the internet.

Create a network printer

Typically, we have multiple computers in our home but when we want to print we only have one printer necessitating us to plug in to print a document. What if we could create a central network printer using our Pi server? Well, we can using CUPS, Common Unix Printing System a printing service created by Apple and used on OS X and Unix systems around the world.

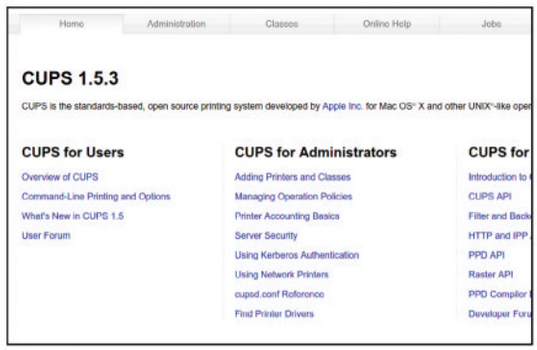
Typically we run a version of CUPS on our computer but we can also instruct our computer to print to a network printer. To install CUPS you will need to SSH into your Pi server and type into the *LXTerminal* the following.

```
$ sudo apt-get install cups
```

You will then need to complete a few configuration changes. We found a great guide for this on *How-To Geek* (**http://bit.ly/LXF198-cups**) that illustrates how to complete this task on your Raspberry Pi. To administer a remote CUPS printer all you need to do is open a browser on your computer and type in the IP address for your Pi server and append **:631** to it:

```
192.168.0.6:631
```

You can then add new printers, check the status of print jobs and cancel any erroneous prints. You can even set up an extreme remote print job by using port forwarding on your router to forward print jobs from outside your network to the CUPS printer.



CUPS comes with an easy to use admin web page that enables you to remotely administer your network printer and manage/install new printers via a wizard.

power up your Pi. The boot process on the Pi 2 will take less than 15 seconds, and when it's finished you'll be transferred to the raspi-config screen. Your first task is to expand the filesystem on the SD card so that you get the maximum amount of space. To do this, navigate to option 1 and press Enter to start an automated process. With that complete, navigate to option 8 – Advanced options. Now navigate to A3, Memory Split and press Enter. Change the value to 16 and press Enter. This gives you 16MB of RAM for the GPU. As you'll be using this headless, you won't need a lot of video RAM. You'll be returned to the main menu, but you need to navigate back to the Advanced menu – option 8. In the Advanced menu, navigate to A4, SSH, move the cursor to Enable and press Enter. That's our configuration done. Return to the main menu, navigate to Finish and press Enter. Reboot your Raspberry Pi to finish the configuration.

With the Pi rebooted, go ahead and log in. The default username is **pi** and the password is **raspberry**. Next, make sure that your software is up to date. At the *LXTerminal*, type in the following and press Enter.

```
$ sudo apt-get update
```

This will compare your software repository lists against what is on the Raspbian servers and updating where it needs to. If there is a lot to download, Raspbian will ask before it does so. Depending on your internet speed, this may take a while, but once it's done control will be returned to you.

Getting a fixed IP

With the updates complete, you can now create a fixed IP address for the Pi server. To do this, you will need to edit the **/etc/network/interfaces** file. You will need to know the current IP address of the Pi, which you can find by running:

```
$ ifconfig
```

This will spit out a lot of text. As you're using the Ethernet connection, look for the eth0 line, typically at the top of the text. In this section, look for the inet addr: Ours looks like this:

```
eth0      Link encap:Ethernet  HWaddr b8:27:eb:b8:d2:c8
inet addr:192.168.0.6  Bcast:192.168.0.255
Mask:255.255.255.0
```

We also need to run **netstat** to get more details:

```
$ netstat -nr
```

And pick out two numbers from here for the gateway and the destination. Make a note of both.

So with this information noted, edit the **/etc/network/interfaces** config file.

```
$ sudo nano /etc/network/interfaces
```

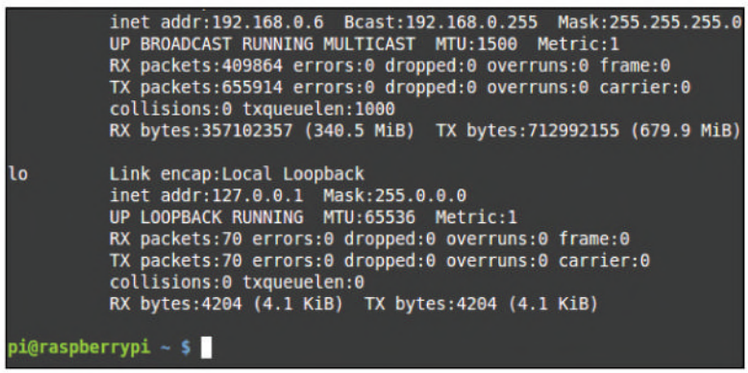
The default configuration will show that interface eth0, Ethernet, is using DHCP (Dynamic Host Configuration Protocol). In other words, it will get an IP address each time it connects to the router:

```
iface eth0 inet dhcp
```

In order to keep track of where our Pi server is on the network, we will fix its IP to what we found in **ifconfig**, which in our case was 192.168.0.6, but yours will differ. We will make changes only to the **iface eth0** section; the other sections must be left as they are.

Our new section looks like this, yours will reflect the setup of your network:

```
iface eth0 inet static
#The IP address that we wish to use
address 192.168.0.6
#This is used to divide IP addresses into subnets, this is the
standard and will work for you
netmask 255.255.255.0
#This is the IP address structure for our network, yours
might be 192.168.1.0
network 192.168.0.0
#We found this via ifconfig earlier it was labelled BCAST
broadcast 192.168.0.255
#This is the gateway address that we found in netstat -nr
  gateway 192.168.0.1
```



The ifconfig command is powerful and can be used to view the configuration of interfaces, but it can also be used to reconfigure them on the fly.

Quick tip

To make this project a little special you could add a Unicorn HAT board (**http://pimoroni.com**) to provide visual feedback for CPU usage, successful backup or to highlight issues. This is all possible using a Python script which can be run on boot.

» When you're ready, save the changes by pressing CTRL+O, then exit using CTRL+X. Reboot your Raspberry Pi by typing:

```
$ sudo reboot
```

Your Raspberry Pi will reboot and return you to the login prompt. Log back in and type:

```
$ ifconfig
```

to check that your IP address is now static; if you want to check that you have internet access, type the following:

```
$ ping google.com
```

If the pings return properly, you've configured the network settings correctly. At this stage, you can carry on using the keyboard and monitor or you can go headless and connect to your Pi server over the network. We chose to log in via SSH from our Linux Mint laptop. If you are going to do the same, you will need to open a terminal on your computer and type:

```
$ ssh pi@IP ADDRESS
```

Remember to replace the IP address with the static address that we used previously. You will need to provide your password before you can log in, but once successful any command issued will go through on the Raspberry Pi.

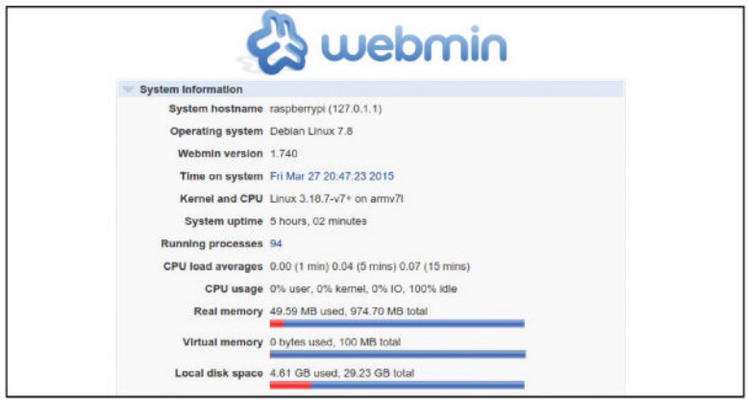
With your system built, you now need to configure it to act as a server, and the first task is to enable Raspbian to read and write our USB 2.0 external hard drive. If a drive is formatted as a typical Linux filesystem – ext3, ext4 or btrfs – you can read the drive with no configuration necessary. But because this drive is formatted NTFS – a Windows filesystem – you need to install a tool to enable read and write access to the drive, and this is called NTFS-3G. To install, type the following into the *LXTerminal*:

```
$ sudo apt-get install ntfs-3g
```

It will take a few seconds to install, and when complete, control will be returned to you.

Quick tip

The Raspberry Pi B+ and Raspberry Pi 2 Model B work better with external USB drives than their predecessors. Inserting a USB drive on the original Pi would trigger a reboot, but this was fixed for all models from the B+ onwards.



› ***Webmin*** **provides a graphical interface for common server administration tasks. You can even use it to update and upgrade the software installed on your Raspberry Pi and set up *Cron* jobs.**

Install Webmin

Administrating a server is traditionally undertaken via the terminal, in our case *LXTerminal*. In fact, we've done quite a lot of this already. However, to make administrating a server a little easier, there's a great tool called *Webmin*. This is a web interface for common administration tasks, such as user management and updating software.

To install *Webmin* on your Pi server, you will need to use the APT package manager to download *Webmin* and its dependencies.

```
$ sudo apt-get install webmin
```

With *Webmin* installed, open a new browser window on your computer and navigate to the IP address of your Pi and use the port number **10000**. Our IP address looked like this: **https://192.168.0.6:10000/**.

You may receive a warning that the certificate for the website isn't to be trusted. If so, don't worry, just click on Advanced Options and proceed onwards. You will next see a login screen – enter the username and password that you have used to log in to your Pi previously.

Once logged in, you will see the main menu split into eight sections. For this project, you need to refer to the System section, which contains controls for managing hard disks and for managing users and groups.

The next task is to mount the external USB hard drive on boot, so you will need the USB drive plugged in to the USB port of the Raspberry Pi. Before you manage the drive with *Webmin*, you will need to return to the *LXTerminal* and create a directory that will act as a share point. In the terminal, navigate to **/media/** and create a new directory called **drive**, as follows:

```
$ cd /media
$ sudo mkdir drive
```

Accessing your Pi server

In this project we've created a file server inside our home network that's not accessible to anyone outside. If you would like to enable your Pi server to be accessed by external users then there are a number of things you'll need to do and to help we have compiled a handy checklist to follow:

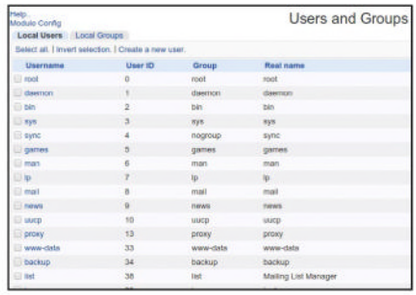
» Change the default password for the Pi user you want to give acess to. You can do this using the **passwd** command in *LXTerminal*.

» Create a new user with no sudo or root privileges, the easiest way to do this is via the System > Users and Groups menu.

» Set up port forwarding on your router so that requests to your external IP address are forwarded to the Pi Server. This is a little different for every router, so please consult your router manual.

» It's likely that your ISP does not provide a static external IP address. If that's the case then install *noipclient*, which you can download from **http://bit.ly/No-ipLinux** and also create an account on its website (**www.noip.com**). The client software will update your IP address to a URL that you create on the website, which will enable you to easily connect to your server without using an IP address.

» Last, it's really important to set up a firewall that will protect your server from external interference. The easiest way to do this is install *fwbuilder* and *VNC* on the server as *fwbuilder* is a GUI application to set up the firewall rules.



› **Webmin can also manage users and groups via its graphical interface, and is a lot easier than the terminal for those new to server administration.**

With the directory created, you now have an issue: only users with sudo access, or root, can use the directory. You need to change the permissions so that users can read and write to the hard disk. To do this in the *LXTerminal*, type the following:

```
$ sudo chmod 770 ./drive
```

That's it for now in the terminal, so let's return to *Webmin*.

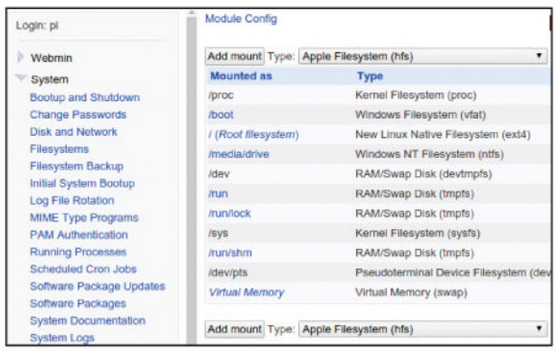
Mount drive on boot

To manage USB drives with *Webmin*, click on the System drop-down menu on the left of the screen, then click on Disk and Network Filesystems. After a few seconds, the screen will change and a list of all the mounted filesystems will be displayed. Our drive will not be listed, so we need to create a new mount using Add Mount; but before you click on the button, look to the right and you will see a drop-down labelled Apple Filesystem (HFS), click on the drop-down and select Windows NT Filesystem NTFS if your drive is formatted using NTFS. If not, select the format of your drive accordingly. Once selected, click on the Add Mount button to open a new menu.

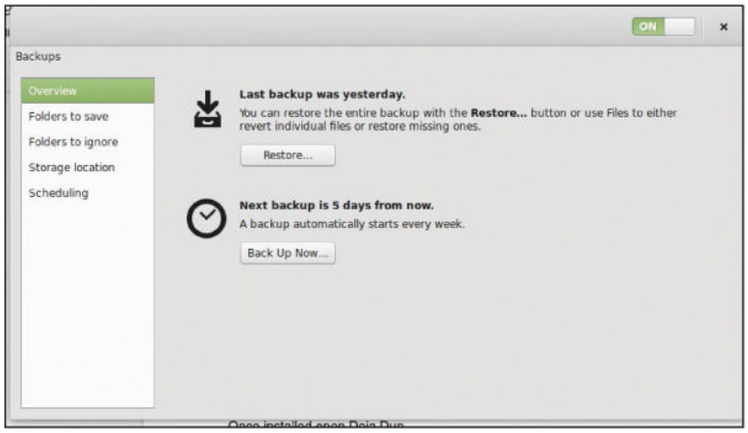
In the new menu, start by clicking on Mounted As and its ... button to the right. This will open a dialog box where you can navigate to the **/media/drive directory** that we created earlier. For the rest of the configuration, just follow this simple checklist:

- Save Mount? Save and Mount at Boot.
- Mount Now? Mount.
- Windows NT Filesystem: Disk. Look for your device in the dropdown menu.
- Mount Options.
- Read-only? No
- Allow users to mount this filesystem? Yes
- Disallow execution of binaries? If mountable by users
- Avoid updating last access time? No
- Buffer writes to filesystem? Yes
- Disallow device files? If mountable by users
- Disallow setuid programs? If mountable by users
- Wait until network interfaces are up? No
- User files are owned by: Leave blank
- Group files are owned by: Leave blank

When you're ready, click on Save to write these configuration settings and mount the drive. The drive will now be available for you to connect to over the network. If you are using Ubuntu or Linux Mint, you can use the file manager to connect to the server, but you can also use *Gigolo*. (A rather blunt name, we note. It's tagline is "It mounts what it is told to." Classy.) This is a graphical front-end for the virtual filesystem GIO/GVFs, which mounts remote filesystems and opens them in your file manager. This will be available via your software repositories.



Creating a mount point can be daunting for new users but using *Webmin* the process can be simplified. The tricky point to remember is the Add Mount button – you must specify the disk format *before* clicking on it.



***Déjà Dup* is a powerful backup and restore tool that is built on the Linux stalwart *rsync*. It can backup multiple directories to local and remote locations and is designed to make backing up easy.**

Back up using Déjà Dup

For the final part of this project we will use our Pi server as a remote backup device, with a great piece of open source software called *Déjà Dup* (**https://launchpad.net/deja-dup**). This is actually a graphical front-end for the flexible, speedy and scriptable *rysnc*, which has grown into a standard Linux utility since its first release back in 1996. It makes backing up really easy to do. To start with, you will need to install *Déjà Dup* on our computer. You don't need to install anything on our Pi server.

On your computer, open a terminal and type:

```
$ sudo apt-get install deja-dup
```

Once installed, open *Déjà Dup* and you will be presented with the Overview screen. You will need to instruct *Déjà Dup* what folders you wish to back up, and that's in the Folders To Save option. Once you have done that, move to the Storage Location menu and fill in the details for your Pi server, including the location to store your backup. We used **/media/drive/Documents**, which was an existing folder on the drive. With that complete, you are ready to navigate back to the Overview and click on Back Up Now.

Your first backup will take longer than subsequent backups, due to the way *Déjà Dup* works. The first backup is a full backup of the directories that you have selected, whereas future backups will be partial to cover changes to the files. To restore from a backup, go to the Overview menu and click on Restore... and follow the wizard.

So there we have it, we have built a server that uses very little power but centralises our files and provides a remote back up solution for our ever-growing digital lives – all thanks to the Raspberry Pi and a little Linux know-how.

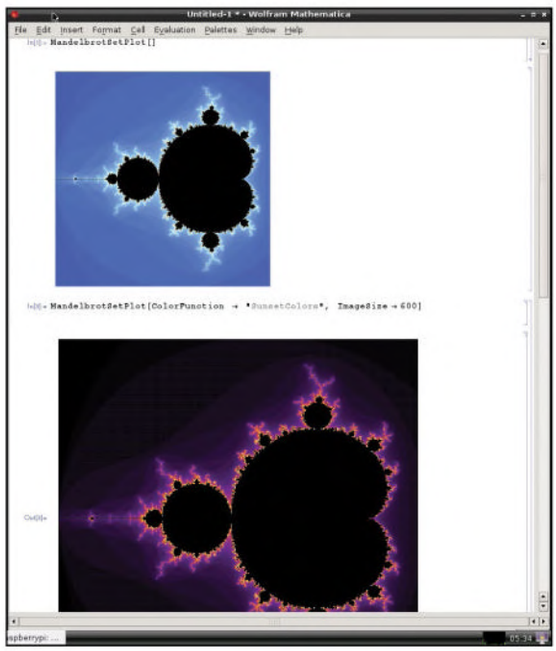
If you would like to know more about *Webmin*, head over to **www.webmin.com**, follow the official documentation and use the wiki for any questions.

Quick tip

You can also use the Raspberry Pi remotely using *VNC*. To install a *VNC* server on your Pi, SSH in to it and type **sudo apt-get install tightvncserver** and then type **sudo tightvncserver** to start. Install a *VNC* viewer and then navigate to your IP address, but add **:1**, for the port, to the end.

Mathematica: Visualise data

Do some serious calculating with the Pi.



Everybody's favourite fractal is easy to draw and colourise in Mathematica.

Mathematica is a fearfully powerful symbolic computation package. It's published by Wolfram Research and has been on the scene for over 25 years, during which time it seen heavy adoption by both academia and industry. Powered by the general-purpose Wolfram language, it provides a simple platform which can solve, simulate, approximate or decorate pretty much anything you can throw at it.

While we're not usually inclined towards proprietary software (we much prefer free and open source), but we make an exception in this case because Wolfram made the decision, in November 2013, to release a free version of Mathematica (and indeed the Wolfram Language) for the Raspberry Pi. If your views on free software are sufficiently austere, then consider yourself free to not use it.

Still here? Okay then, if you have a reasonably new release of Raspbian then good news: You already have Mathematica installed. If not you can get it with a simple

```
$ sudo apt-get update
$ sudo apt-get install wolfram-engine
```

Make sure you have enough space though as the whole install weighs in at about 600MB.

Up and calculating

The package will install two programs, Mathematica and Wolfram Language. Mathematica will start a notebook style graphical interface, and Wolfram Language will start a terminal based one. The Wolfram Language is instrumental in powering the Wolfram Alpha knowledge engine as well as the new Wolfram Programming Cloud. It strives to maximise automation and unification with the goal that, in Stephen Wolfram's own words "once a human can express what they want to do with sufficient clarity, all the details of how it is done should be handled automatically". If you're familiar with the package then beware, to quote Stephen again: "the Raspberry Pi is perhaps 10 to 20 times slower at running the Wolfram Language than a typical current-model laptop and sometimes even slower when it's lacking architecture-specific internal libraries". In sum, prepare to be patient.

We'll begin with showing the basics of Mathematica. In its most simplest form, you can use it as a calculator: Click on the worksheet and type **3 + 2** (or some other suitably complicated expression) at the **In[1]:=** prompt, press Enter, or select Evaluate Cell from the Cell menu, and you should see something like **Out[1]=** followed by the correct answer. Naturally, the program is capable of much more taxing calculation. Try **2014 ^ 2013**, and be amazed at how quickly the little computer spits out a big answer. It would also be

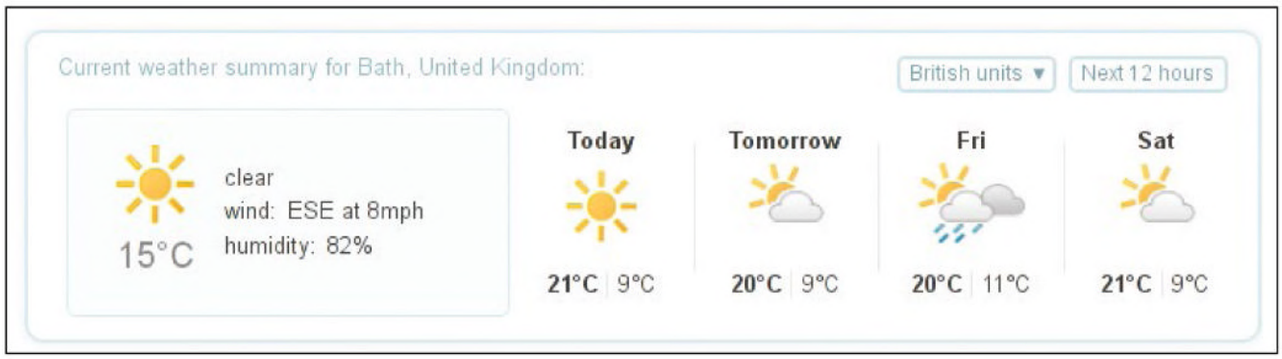
Calculate in the cloud

If the things you want to compute start grinding the Pi to a halt or saturating its memory, then it's possible to send certain queries to the Wolfram Alpha knowledge engine. Where possible, responses are sent back in a form that you can continue to work with in Mathematica. Naturally, this requires your Pi to be connected to web.

Wolfram Alpha is capable of understanding natural language queries as well as those in the Wolfram Language (eg Mathematica input), so you can ask it about anything you like. For example, rather than risk looking out of the window to see external conditions, we can simply type:

```
WolframAlpha["weather bath uk"]
```

Bear in mind that there's a limit to how much free compute time you're allowed, so you won't be able to calculate the answer to life. But the cloud technique is nonetheless useful for calculations which intermediately use a lot of memory, but return an easy to swallow answer.



remiss of us not to do something pi-related here, so lets calculate the first one million decimal digits of that transcendental:

```
pi = N[Pi, 1000000];
```

On our Raspberry Pi this took all of 12 seconds (Note that the semicolon suppresses outputting this rather lengthy result, which would significantly increase the time taken). We can define our own functions too, for example we could make a very naive Fibonacci number implementation like so:

```
F[0] = 0;
F[1] = 1;
F[x_] = F[x -1] + F[x -2];
```

We use the underscore to indicate that **x** is a user-supplied argument. This function works – we can quickly work out the first terms of the sequence as 0, 1, 1, 2, 3, 5, 8 etc – but things rapidly grind to a halt when we want to find, say, the 1,000th Fibonacci number. You could write a better function, but no need to reinvent the wheel:

```
Fibonacci[1000]
```

will swiftly answer your query.

Now for equations

Remember simultaneous equations from school? Something like solving 2x + 3y = 11 and 3x - y = 0? While this example is child's play, if we have more variables then the situation becomes harder. We form a matrix of coefficients and if possible invert it. This is a tedious process to do by hand (using the method of Gaussian Elimination, and that's got nothing to do with classic Bullfrog title, *Syndicate*), commonly suffered by hungover undergrads and involving lots of scribbling out. It is also the very same task that a significant proportion of the world's supercomputing time is devoted to, since so many models are based on linear systems.

We can solve our simple linear system above (if you haven't already done so) with a simple:

```
m = {{2,3},{3,-1}}
minv = Inverse[m]
minv * {{11},{0}}
```

Mathematica will return the vector {{1},{3}}, meaning x = 1 and y = 3.

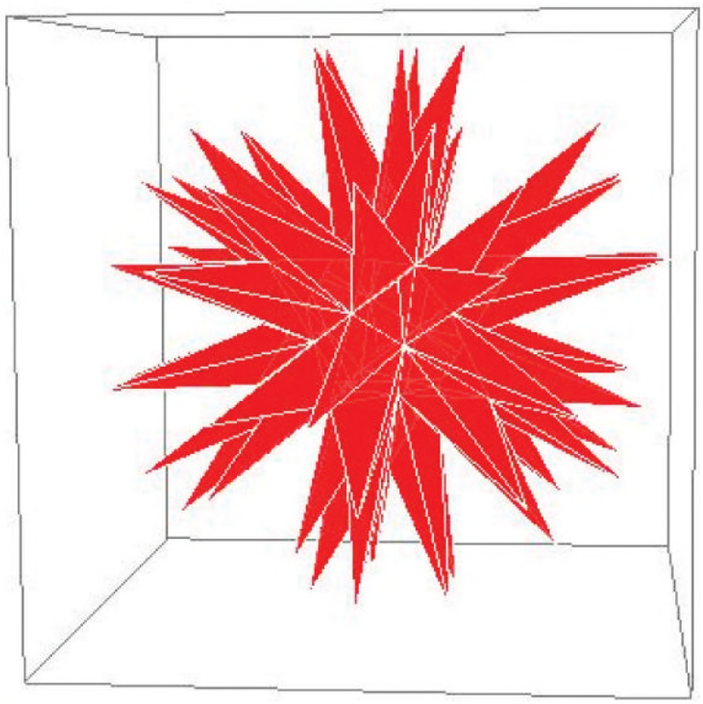
For no real reason, let's try and make Mathematica invert a 20x20 matrix of random floating point values.

```
m = RandomReal(1,{20,20})
Inverse[m]
```

It's nice to visualise the matrix as a rectangular array, rather than a list full of curly brackets. Doing this is a simple question of adding:

```
m // MatrixForm
```

Besides linear algebra, Mathematica can help you with your calculus homework. In particular it's very good at »



This is the Echidnahedron, which the Mathematica logo (also known as Spikey) is based on, albeit with some hyperbolic jazz thrown in.

integrating and differentiating things. We use the function call **D[f,x]** to differentiate the function **f** with respect to the variable **x**, so you can do something simple like:

```
D[cos[x] + x^2, x]
```

which will obtain the solution **-sin[x] + 2x**. Or you can try something a little harder, such as:

```
D[tan^-1[x^x],x]
```

You can also find the (lengthy) second and third derivatives of this function using:

```
D[tan^-1[x^x],{x,2}]
```

and then:

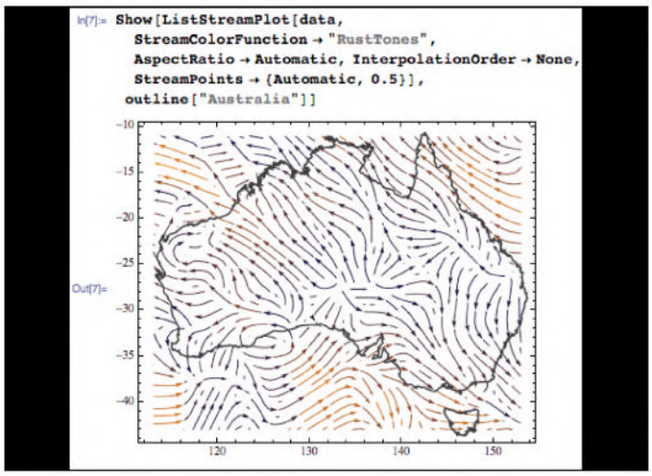
```
D[tan^-1[x^x],{x,3}]
```

You can also use the **D[ ]** function to do partial derivatives, or even implicit differentiation:

```
D[x^2 + (y[x])^3,x]
```

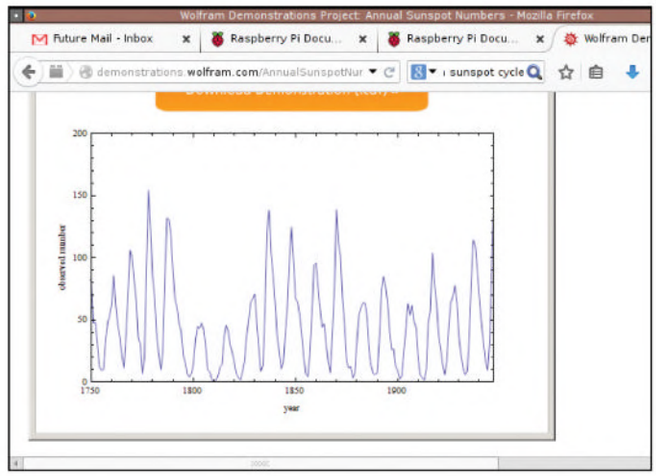
Students often find integration harder than differentation, and for many years now have been using the online integrator at **http://integrals.wolfram.com** to do their homework. As you would expect, Mathematica can symbolically integrate pretty much any function where this makes sense (there are

Things to try with Mathematica



1 Stream plot

Here is a stream plot that shows the directions of wind across Australia. The higher wind speeds are represented by brighter colours. To do this yourself, the **outline** function needs to be manually entered, but you can find all the details on how to do this on the Wolfram blog. Stream plots are commonly used to visualise differential equations (**http://bit.ly/WeatherPatterns**).

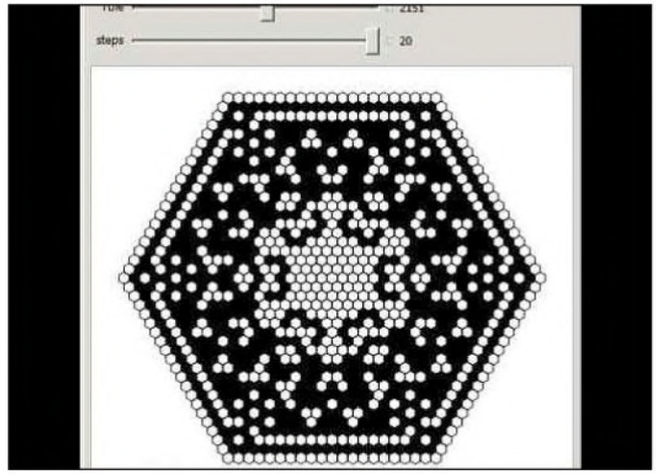


2 Sunspot Cycle

Witness the 11-year sunspot cycle using publicly available data. Between 1640 and 1710 there were abnormally few sunspots, which coincided with the European 'mini ice age'. In general, if you're looking for frequencies in noisy data, a good trick is to convolve your data with a moving average kernel to smooth it out before using Fourier analysis (**http://bit.ly/SunspotNumbers**).

3 Celestial Navigation

There is an old Maori proverb: "Before you embark on a journey, make sure that you know the stars". In this demo you can learn to get your bearings by measuring the angles of altitude to planets, stars or the moon at a specific time. The celestial sphere is approximated by the Earth moving through a circular orbit at constant speed, otherwise things get ugly (**http://bit.ly/CelestialNavigation**).



4 Snowflakes

Cold outside? Have a play with some different kinds of snowflakes. This one is generated using hexagonal cellular automata. All snowflakes exhibit hexagonal symmetry due to hydrogen bonding in water molecules. When they freeze, the crystals are formed into a hexagonal arrangement due to the layout of the charges (**http://bit.ly/SnowflakeLikePatterns**).

exceptions; such as things like **x^x** whose integral has no symbolic representation). However, all the elementary functions are handled as you would expect:

```
Integrate[x^2,x]
Integrate[sin^-1[x],x]
Integrate[log[x],x]
```

Remember: Don't forget the constant of integration. Furthermore, you can even get complicated expressions for things which don't integrate so nicely. For example

```
Integrate[ln[cos[x]]]
```

evaluates to a rather ugly complex expression involving polylogarithmic functions. One of Mathematica's most impressive capabilities is its ability to produce graphics. Graphs, surfaces, networks, maps are all just a couple of lines away. We can plot a period of the sine function with a simple:

```
Plot[Sin[x],{x,0,2 * Pi}]
```

You can also make nice weather charts Mathematica's built in **WeatherData[]** function. For example, to plot the daily mean temperatures for this year's great British Summer:

```
DateListPlot[WeatherData["Bath (United Kingdom)","MeanTemperature",{{2014,6,1},{2014,9,1},"Day"}],Joined->True]
```

We can even venture into three dimensions and plot the following function:

```
Plot3D[Sin[x]+Cos[y],{x,0,2 * Pi},{y,0,2*Pi}]
```

One can draw two (or more) lines or surfaces on the same plot by adding them to the relevant Plot function. For example, if you need reminding about how the sine and cosine functions are related:

```
Plot[Sin[x],Cos[x],{x,0,2 * Pi}]
```

Shapes are easy too, thanks to Mathematica's extensive library of polyhedra. To draw a red Echidnahedron (an icosahedron stellation having 92 vertices, 270 edges, and 180 faces) do:

```
Graphics3D[{Opacity[.8], Glow[RGBColor[1,0,0]],
EdgeForm[White], Lighting -> None,
PolyhedronData["Echidnahedron", "Faces"]}]
```

Raspberry special

For the most part, the Pi edition of Mathematica is a diet version of the full product. However, it does some features which are exclusive, namely the ability to talk to devices connected via the GPIO pins and the CSR connected PiCam module. All this invocation happens through the **DeviceRead** and **DeviceWrite** commands, eg to set pin 14 to high:

```
DeviceWrite["GPIO", 14 -> 1]
```

replacing 1 with 0 to set it to low. To read the status of GPIO pin 14 (GPIO14 in the BCM numbering, take care here), do:

```
status = DeviceRead["GPIO", 14]
```

and the variable **status** will take on the value 0 or 1 as appropriate. You can take import an image from the camera module into Mathematica as follows:

```
img = DeviceRead["RaspiCam"]
```

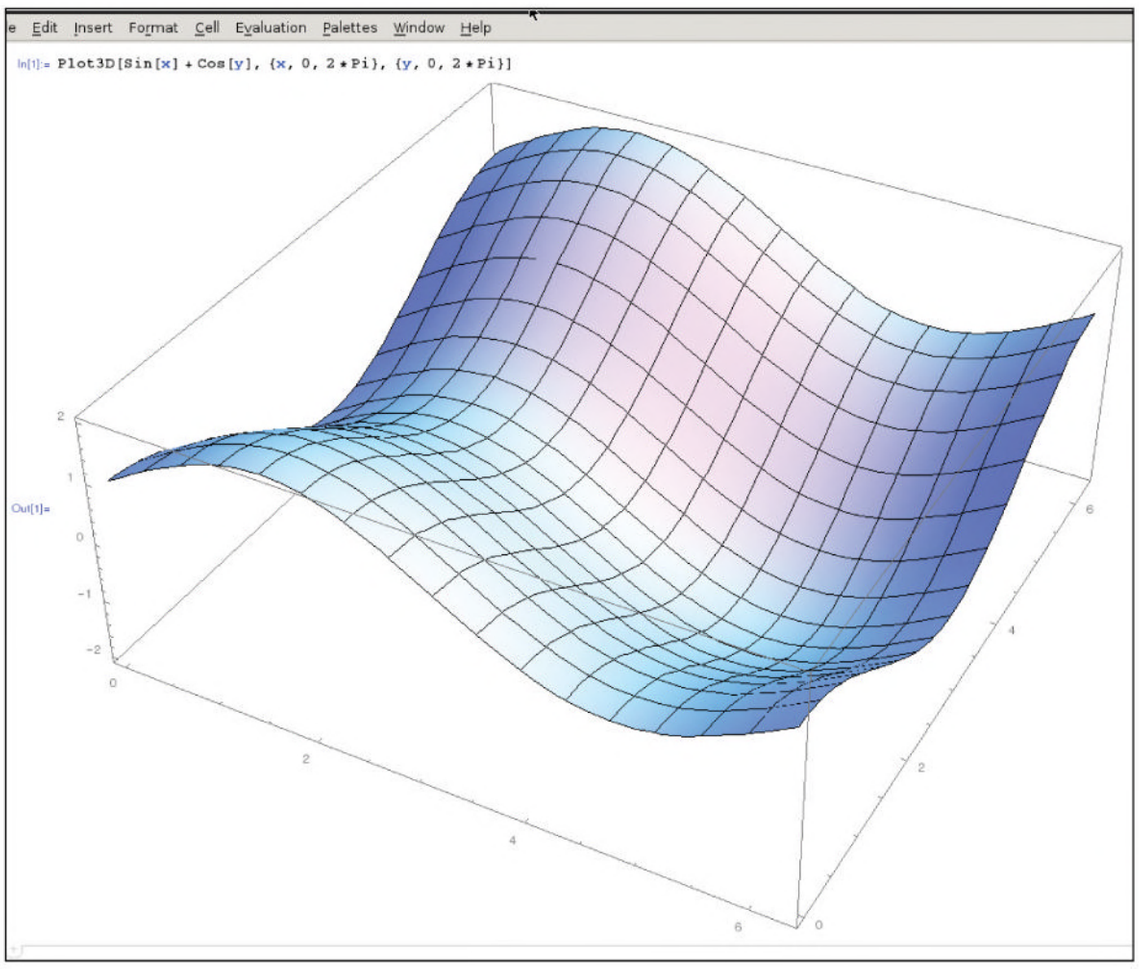
And then you can export it as a JPEG, or whatever format you like using the following:

```
Export["/home/pi/img.jpg", img]
```

And here we end our quick journey through the Pi edition of Mathematica.

Quick tip

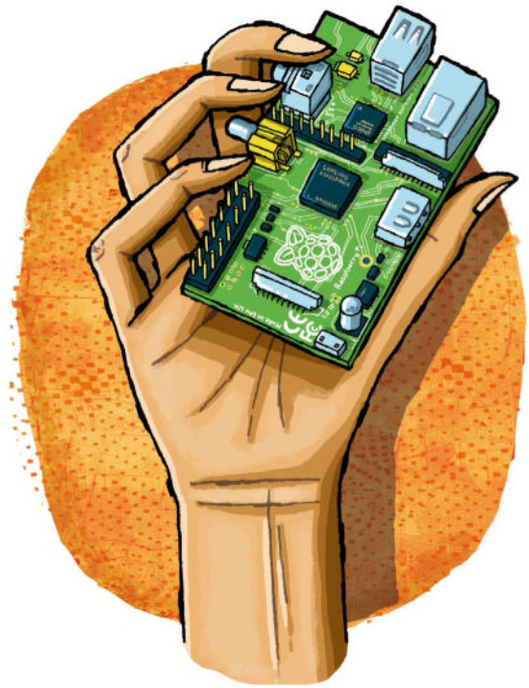
Check out the regularly updated **http://blog.wolfram.com** to see some weird and wonderful Mathematica excursions, including how to win at Rock, Paper, Scissors.



Adding sines and cosines is what Fourier analysis is all about.

Python: Build a multi-Pi cluster

Let's connect a bunch of Pis together in a cluster and harness their power to crack passwords.



Once, it was common to network (not necessarily powerful) computers together to combine their power towards a single task, or group of related tasks. Examples include render and simulation farms, in which the work can be easily distributed between nodes. Such clusters are, in a sense, becoming less fashionable – specialist or enterprise-grade hardware offers a much more favourable answers-to-watts ratio, and multicore processors share data far faster than network cables. But let's not allow trends to get in the way of good old-fashioned tinkering.

A thorny issue

The collective term for the aforementioned networks is a Beowulf cluster. This name derives from the eponymous hero of the classic Anglo-Saxon poem, who had "30 men's heft of grasp in the gripe of his hand". Where the compute hardware consists solely of Raspberry Pis, a new moniker has been invented by the RPi community: a Bramble. While most desktop computers are (depending on your chosen benchmark) around an order of magnitude faster than the Pi, it is still an enjoyable and inexpensive exercise to multiply by Pi and therefore learn about distributed computing in Python. For this demonstration, we used two Pis powered by a Pimoroni hub (which gives independent power to each port) and connected via a 100Mb five-port switch (which cost £5 from eBay). The hub can power a maximum of five Raspberry Pis, so one day perhaps this Bramble will grow.

For this tutorial, we will have our compute units brute-forcing two well-known hashing algorithms: MD5 and SHA-1. More specifically, we shall try to find a pre-image for a given hash value which we shall compute from a (rather spineless) passphrase. While both of these algorithms are vulnerable to collision attacks (where the goal is to find any pair of inputs with the same hash), pre-image attacks are much harder to come by. Such an attack does exist for MD5, but it still would require an unreasonable length of time. At the time of writing, there is no pre-image attack known for SHA-1. The Flame malware in 2012 exploited collision weaknesses in MD5 to forge Microsoft security certificates.

Crunch some code

We often hear of database compromises in which leaked credentials include password hashes. While the use of salt helps defend against some attacks, passwords remain vulnerable to a wordlist-based attack, so such a compromise should be considered serious. We won't cover wordlist attacks in our Python code, concentrating rather on elementary character combinations. You are encouraged to have a go at implementing wordlists in Python – you can download prefabricated lists or generate your own using a tool such as *Crunch*. You could then write some simple code to split up this list and share the work among the workers.

The cryptographic hash functions are all available in a standard Python installation via the *hashlib* library. For example, you can find the MD5 hash of the string password1 from the Python prompt thus:

```
>>> import hashlib
>>> h = hashlib.md5('password1')
>>> h.hexdigest()
'7c6a180b36896a0a8c02787eeafb0e4c'
```

By replacing **md5** with **sha1** above, we also obtain the corresponding SHA-1 hash:

```
'db6a4668837a519ccef4616751e41674807d6a8a'.
```

For our tutorial, we will attempt to find a pre-image for the following MD5 hash:

```
target_hash_md5 = '392b2bd9f1571ff02e7dc21adfb2f0b0'
```

Suppose we are only interested in passwords consisting of uppercase and lowercase letters, together with the numbers 0 to 9. This can be done painlessly by combining

the appropriate ranges of ASCII codes:

```
ascii = range(48,59)+ range(65,91) + range(97,123)
chars = [chr(j) for j in ascii]
```

We can use the *itertools* module to generate all possible combinations of a given length here. This saves on some potentially irksome code. The *itertools.product* class returns a generator object, so rather than squashing us (or rather our Pis and Brambles) with a huge list, it spits out one combination per iteration. The repeat parameter is the length of the outputted strings. So we repeatedly check to see if its output hashes to our target hash.

```
for j in itertools.product(chars,repeat=length):
    guess = ''.join(j)
    m = hashlib.md5(guess)
    hash = m.hexdigest()
    if hash == target_hash_md5:
        print "Great victory!"
        break
return(guess)
```

The functions md5cracker() and shacracker() can be called with the length argument described above and it is straightforward enough to modify it to brute-force a range of password lengths. On a single-core powered 2.1GHz Athlon laptop, attempting all three-character passwords using our set of characters took about 0.7 seconds; four characters took 40 seconds; and five figures took about 40 minutes. As we are dealing with 62 characters, the latter figure suggests we are churning through about 380 kilohashes per second, so we could expect to crack a six-character password in about two days and an eight-character one in about 18 years. For SHA-1, the same machine managed about 314kH/s, which is sinisterly close to 100,000 times pi. Less decrepit hardware will fare much better, but a lone Pi does much worse, managing a pitiful 6kH/s for MD5 and a not much better 8kH/s for SHA-1. All this with no X server or other daemons running. Clearly something does not work so well on the ARM architecture – further investigation is beyond the scope of this article, since anything we can do John the Ripper can do better (more on that later). Incidentally, the Python file also contains the following codeblock at the end:

```
if __name__ == '__main__':
    import timeit
    print(timeit.timeit(stmt = "shacracker(4)", setup="from __
main__ import shacracker", number=1))
```

This means that if the file is executed from the command prompt, then it will be automagically benchmarked by the *timeit* module. One could attempt to speed up this code using Cython to generate C code, but gains here may be limited. The *hashlib* and *itertools* functions are all implemented in C, so it is unlikely that you would be able to speed up these at all. Besides, even if you could, string manipulations have their own drawbacks in C, so unless you are pretty deft with malloc() and free() it would be hard to speed these up. You are welcome to try, though.

Adding more brains

Brute-forcing hashes is an example of an "embarrasingly parallel" computation: it is so simple to divide up the work that it would be embarrassing not to. As such, you can pretty much expect to multiply your hash rate by the number of Pis you are using, since the work can be easily distributed by some judicious tampering with the generator function. More precisely, by assigning each unit a fixed range to consider for the first character. For example, with two Pis we would let one of them deal with passwords beginning with the first 31 characters of our range (0-u), and the second with the range (v-Z). In this way there is no further data that requires to be shared among the Pis – they are left to get on with their assigned duties without wasting time chatting.

We will use the *dispy* module to handle all the parallelism, which requires that all nodes run the *dispynode* program. They can then be very easily called to action as required. We divide the whole keyspace into 62 chunks so that workers can request chunks as they are needed. This means that faster (or slower) machines can be added to the cluster with no fuss, since otherwise the workload would have to be distributed manually to avoid concurrency issues.

We make an own outer loop for the first character, with the *itertools* loop inside it for the remainder.

```
def md5cracker(chunk_start,chunk_end,chars,length,thash):
    for j in range(chunk_start,chunk_end):
        for k in itertools.product(chars,repeat = length - 1):
            guess = chars[j] + ''.join(k)
            ghash = hashlib.md5.new(guess).hexdigest()
            if ghash == target_hash_md5:
                return guess
    return False
```

While you could manually run from one Pi to another to call this function with appropriate chunk parameters, the work distribution can be made less tedious by using the *dispy* module, which you can find on the disc. Install it with:

```
$ tar -xvzf dispy-3.15.tar.gz
$ sudo pip install -e dispy-3.15
```

We've given our two Pis the addresses **10.0.1.1** and **10.0.1.2**, which can be easily achieved by fiddling with

»

Distributed Dzargon

Computing architecture can be roughly split into four classes: Single Instruction Single Data (SISD), Single Instruction Multiple Data (SIMD), Multiple Instruction Single Data (MISD) and Multiple Instruction Multiple Data (MIMD). This categorisation originated in the 1960s and is known as Flynn's Taxonomy, after its inventor Michael J Flynn. Multiprocessor systems as we know them did not exist at the time, but machines were nonetheless capable of performing parallel operations, in much the same way as Intel's SSE (Streaming SIMD Extensions) that first appeared in Pentium III processors. Certain applications today straddle the boundaries of the Flynn classes, but they still provide a reasonably accurate overview of the present day situation. The pre-image discovery that is the subject of this tutorial falls nicely into the SIMD category; we perform the same instruction on multiple data sets – discrete sets of passwords of a given length. The most common category in use nowadays, however, is MIMD.

At one end of the distributed computing spectrum we have a distributed memory model in which the processing unit has its own individual memory store. If data needs to be shared between processors, then a messaging interface needs to be invoked. At the other end we have parallel computing, in which there is a shared memory that can be accessed by all processors. This may be bus-based, if all the processors are connected to the same board, or software-based, where consistency is maintained by creating a virtual memory store. Either way, bus contention and access time penalties mean that care must be taken with memory-based operations when working in the parallel computing model.

network config files or with a simple:

```
$ ifconfig eth0 10.0.1.1/24
```

This might involve a hefty bout of display and keyboard cable swapping if you have many Pis. The actual addresses aren't so important here, as long as everyone is on the same subnet then *dispy* will be able to discover all of the slaves.

A simple job cluster for our md5cracker() function is created using *dispy* like this:

```
cluster = dispy.JobCluster(md5cracker,callback=callback)
```

The **callback** parameter tells our cluster to call a function called **callback()** whenever one of the nodes finds the password or runs out of work. A callback function can also be used to collate intermediate or approximate results for more complex scenarios, but we shan't worry about this. Our callback function need only check the result and if it's not False then it can shut down all the other jobs:

```
def callback(job):
    if job.result:
        print "Great Victory!", job.result
        for j in jobs:
            if j.status in [dispy.DispyJob.Created, dispy.DispyJob.
Running]:
                cluster.cancel(j)
```

Since the md5cracker() function needs the *itertools* and *hashlib* modules, these have to be imported from inside the function. For reasons of synchronisation, *dispy* won't let you share any in-scope variables with the workers, so the md5cracker() function accepts the following additional parameters: chars (list of characters), length (password length) and thash (the target hash).

Finally we will use the following bit of boiler-plate to kick the system into action when *multipi.py* is executed and display some stats when 'tis done:

```
if __name__ == '__main__':
    cluster = dispy.JobCluster(md5cracker,callback=callback)
    jobs = []
    start = 0
    for j in range(nchunks):
        end = start + chunk_size
        job = cluster.submit(start,end,chars,length,target_hash_
md5)
        job.id = j
        start = end
        jobs.append(job)
    cluster.wait()
    cluster.stats()
```

Don't fear the Ripper

There are many alternatives to the *dispy* module (**https://wiki.python.org/moin/ParallelProcessing**), but it was chosen because it is well suited to simple parallelism such as our distributed hashing. If you want to investigate parallel processing using just a multicore machine, you should check out Python's native multiprocessing library. This allows things such as shared variables to be set up much more easily and consistently than when your processes are non-local. If you want to go even deeper into the parallelism rabbit hole, you should get to grips with the Message Parsing Interface (MPI).

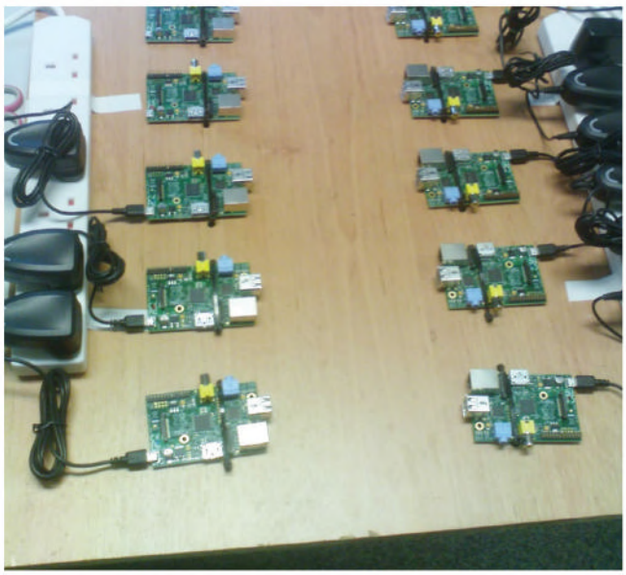
John the Ripper is a popular open source password cracker. There is also a community-enhanced edition, which features support for GPU-based cracking, as well as some additional hash and cipher types. The source for this version (codenamed Jumbo) is available at the project's website, **www.openwall.com/john**. We can compile this from source on the Pi. Raspbian includes *gcc* by default, but we

Other Bramble projects

Office document cracking (10 Raspberry Pis)

www.softwareontheside.info/2013/01/my-raspberry-pi-cluster.html

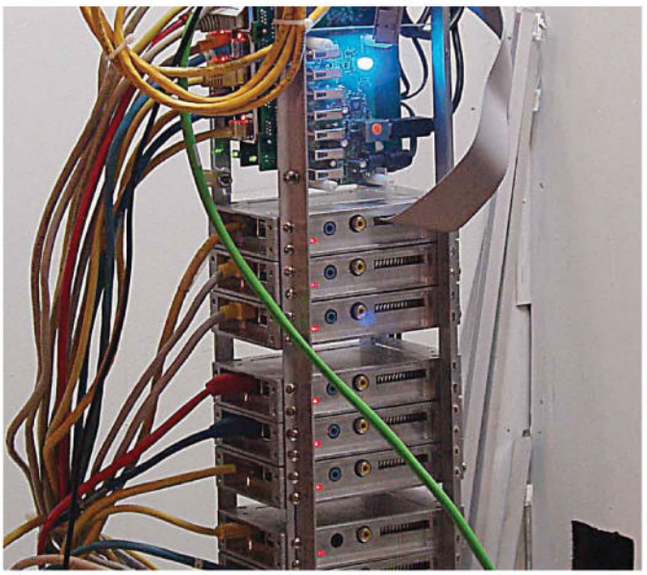
Shane's project uses a patched version of JtR to crack password protected *MS Office*, *LibreOffice* and PDF documents. *MS Office 2007* introduced AES encryption with a 128-bit key, ouch!



Tower of Pi (10 Raspberry Pis)

http://terminus.attitude.net/raspberry-pi

This used 9 RPi slaves and 1 Cubie board as a master controller for MPI processing. It had a neat 16x2 RGB display for status information. Now it has been repurposed as the 32-node Cubical Monolith Project.



additionally require the *libssl* and *libcrypto* headers. The following commands will install these on Raspbian, assuming that you have a working internet connection:

```
$ apt-get update
$ apt-get install libssl-dev
```

Download the source from the website, and then unzip it and begin the lengthy compilation process like so:

```
$ tar -xvzf john-$VER-jumbo-$REL.tar.gz
$ cd john/src
$ make generic
```

This takes about half an hour on a standard-clocked Pi. Once it has finished building, we can use the following command to test out our install:

```
$ cd ../run
$ ./john --test
```

Our freshly-squeezed binary will benchmark all of its available algorithms. *RawMD5* scored about 330kH/s on our device, outperforming by a factor of about 50 our Python implementation. For SHA-1 (**--mode=raw-sha1**), John managed a respectable 190kH/s. Now make a text file **target.md5** consisting of a single line with our target MD5 hash from before, beginning 392b2. We can implement our letters and numbers-only character set by using Incremental mode with the Alnum character set.

```
$ ./john --incremental=alnum --length=6 --format=raw-md5
target.md5
```

This would probably take about two days to complete on your average Raspberry Pi, and it still would not find the required pre-image. This remains as a challenge for you, dear reader. However, your challenge is eased by JtR's extremely helpful **--node** option, which enables you to achieve some primitive parallelism. For example, if we were to saturate our switch with five Pis, we could add the option **--node n/5** to the previous code, replacing **n** with the number of the Raspberry Pi on which it is being run. This will cause John to work on only the relevant fifth of the possible inputs, so that we could realistically expect to get through all six-character alphanumeric passwords in about 10 hours.

You can generate MD5 hashes with the **md5sum** command. For example:

```
$ echo -n "test"|md5sum
098f6bcd4621d373cade4e832627b4f6  -
```

Paste this result, without the space and dash at the end (**md5sum** is most often used to hash files and the filename would usually go after these characters – add **|cut -d- f1** if it bothers you) into a file **test.md5**. Now run the following to convince yourself that John is actually capable of processing these hashes correctly.

```
$ ./john --incremental=alnum –length=4 --format=raw-md5
test.md5
```

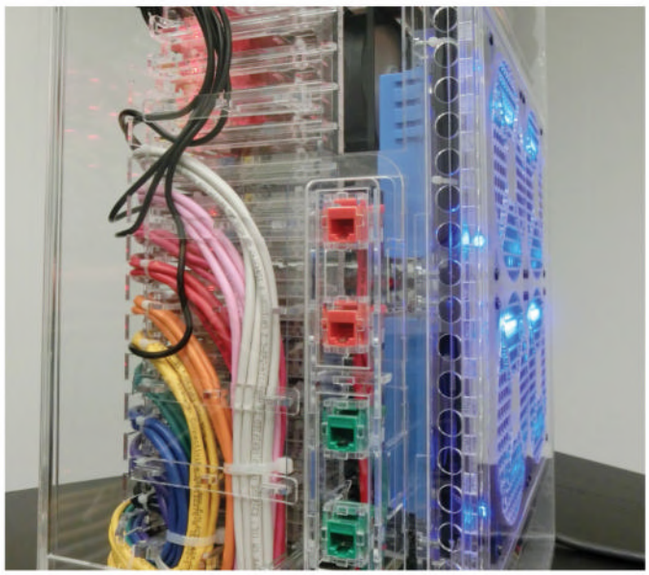
Words work

It would be remiss not to mention the more intelligent wordlist mode of JtR. Numerous wordlists are available from the website, and one can also define mangling rules for combining them. It is even possible to use a hybrid mode with some incremental characters, to maximise your chances of success prior to the universe ending. At the time of writing, the most impressive MD5 benchmark is close to 2GH/s (this is with the aid of 16 Radeon 7550s). With such power, an eight-character password (using only the 62 characters we've been working with) could be recovered in about a week. Faster technology is always just around the corner, and so the old adage that eight-character passwords are secure no longer holds up. In sum: get thyself a password manager and be safe, kids.

David Guill's 40-Pi cluster

http://likemagicappears.com/projects/raspberry-pi-cluster/

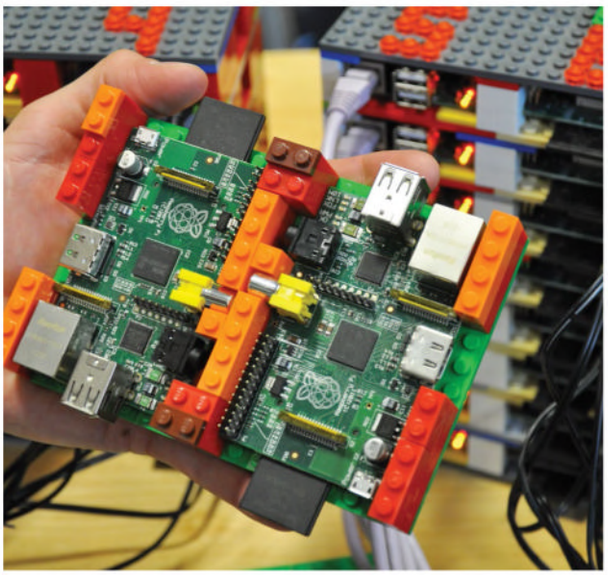
David's goal was to make a model supercomputer which behaves exactly like the real thing. This beastie has 12 1TB drives and 6 external Ethernet ports. A CNC laser cutter was used to make the stylish case.



Iridis-Pi Lego Supercomputer (64 Pis)

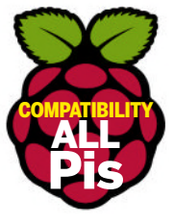
http://www.southampton.ac.uk/~sjc/raspberrypi/

The whole thing cost a mere £2,500 to build and achieves 1.14Gflops/s running the HPL matrix inversion benchmark across all 12 nodes. It uses the older 256MB model, so memory is scarce.



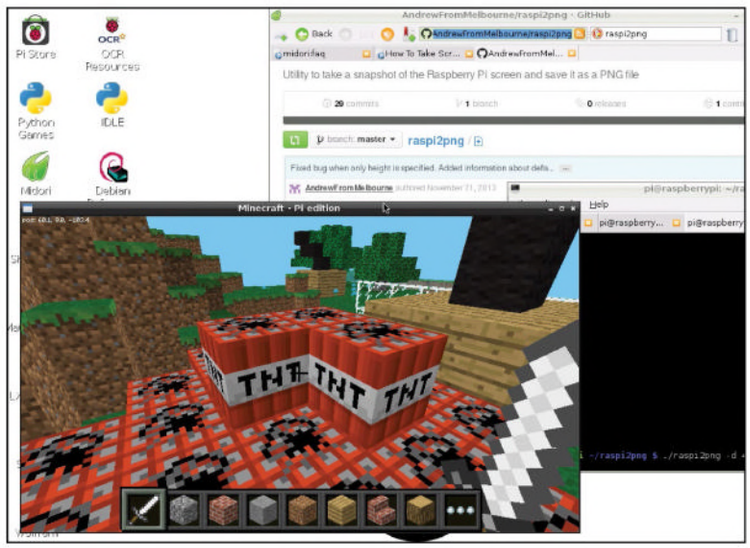
Python: Hack Minecraft Pi

Use Python on your Pi to merrily meddle with Minecraft.



Arguably more fun than the generously provided *Wolfram Mathematica: Pi Edition* is Mojang's generously provided *Minecraft: Pi Edition*. The latter is a cut-down version of the popular *Pocket Edition*, and as such lacks life-threatening gameplay, but includes more blocks than you can shake a stick at, and three types of saplings from which said sticks can be harvested.

This means that there's plenty of stuff with which to unleash your creativity, then, but all that clicking is hard work, and by dint of the edition including an elegant Python API, you can bring to fruition blocky versions of your wildest dreams with just a few lines of code.



Don't try this at home, kids... Actually, do try this at home.

Assuming you've got your Pi up and running, the first step is downloading the latest version from **http://pi.minecraft.net** to your home directory. The authors stipulate the use of Raspbian, so that's what we'd recommend – your mileage may vary with other distributions. *Minecraft* requires the X server to be running, so if you're a boot-to-console type, you'll have to **startx**. Start *LXTerminal* and extract and run the contents of the archive like so:

```
$ tar -xvzf minecraft-pi-0.1.1.tar.gz
$ cd mcpi
$ ./minecraft-pi
```

See how smoothly it runs? Towards the top-left corner you can see your x, y and z co-ordinates, which will change as you navigate the block-tastic environment. The x and z axes run parallel to the floor, whereas the y dimension denotes altitude. Each block (or voxel, to use the correct parlance) that makes up the landscape is described by integer co-ordinates and a BlockType. The 'floor' doesn't really have any depth, so is, instead, said to be made of tiles. Empty space has the BlockType AIR, and there are about 90 other more tangible substances, including such delights as GLOWING_OBSIDIAN and TNT. Your player's co-ordinates, in contrast to those of the blocks, have a decimal part since you're able to move continuously within AIR blocks.

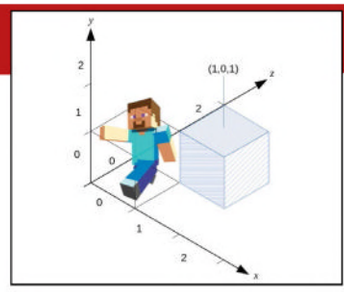
The API enables you to connect to a running *Minecraft* instance and manipulate the player and terrain as befits your megalomaniacal tendencies. In order to service these, our first task is to copy the provided library so that we don't mess with the vanilla installation of *Minecraft*. We'll make a special folder for all our mess called **~/picraft**, and put all the API stuff in **~/picraft/minecraft**. Open *LXTerminal* and issue the following directives:

```
$ mkdir ~/picraft
$ cp -r ~/mcpi/api/python/mcpi ~/picraft/minecraft
```

Dude, where's my Steve?

Here we can see our intrepid character (Steve) inside the block at (0,0,0). He can move around within that block, and a few steps in the x and z directions will take Steve to the shaded blue block. On this rather short journey he will be in more than one block at times, but the *Minecraft* API's **getTilePos()** function will choose the block which contains most of him.

Subtleties arise when trying to translate standard concepts, such as lines and polygons from Euclidean space into discrete blocks. A 2D version of this problem occurs whenever you render any kind of vector graphics. Say, for instance, you want to draw a line between two points on the screen, then unless the line is horizontal or vertical, a decision has to be made as to which pixels need to be coloured in. The earliest solution to this was provided by Jack Elton Bresenham in 1965, and we will generalise this classic algorithm to three dimensions in our next tutorial (over the page).



Isometric projection makes *Minecraft*-world fit on this page.

Now without further ado, let's make our first *Minecraft* modifications. We'll start by running an interactive Python session alongside *Minecraft*, so open up another tab in *LXTerminal*, start *Minecraft* and enter a world, then [Alt]+[Tab] back to the terminal and open up Python in the other tab. Do the following in the Python tab:

```
import minecraft.minecraft as minecraft
import minecraft.block as block
mc = minecraft.Minecraft.create()
posVec = mc.player.getTilePos()
x = posVec.x
y = posVec.y
z = posVec.z
mc.postToChat(str(x)+' '+ str(y) +' '+ str(z))
```

Behold, our location is emblazoned on the screen for a few moments (if not, you've made a mistake). These co-ordinates refer to the current block that your character occupies, and so have no decimal point. Comparing these with the co-ordinates at the top-left, you will see that these are just the result of rounding down those decimals to integers (for example, -1.1 is rounded down to -2). Your character's co-ordinates are available via **mc.player.getPos()**, so in some ways **getTilePos()** is superfluous, but it saves three float to int coercions so we may as well use it. The API has a nice class called Vec3 for dealing with three-dimensional vectors, such as our player's position. It includes all the standard vector operations such as addition and scalar multiplication, as well as some other more exotic stuff that will help us later on.

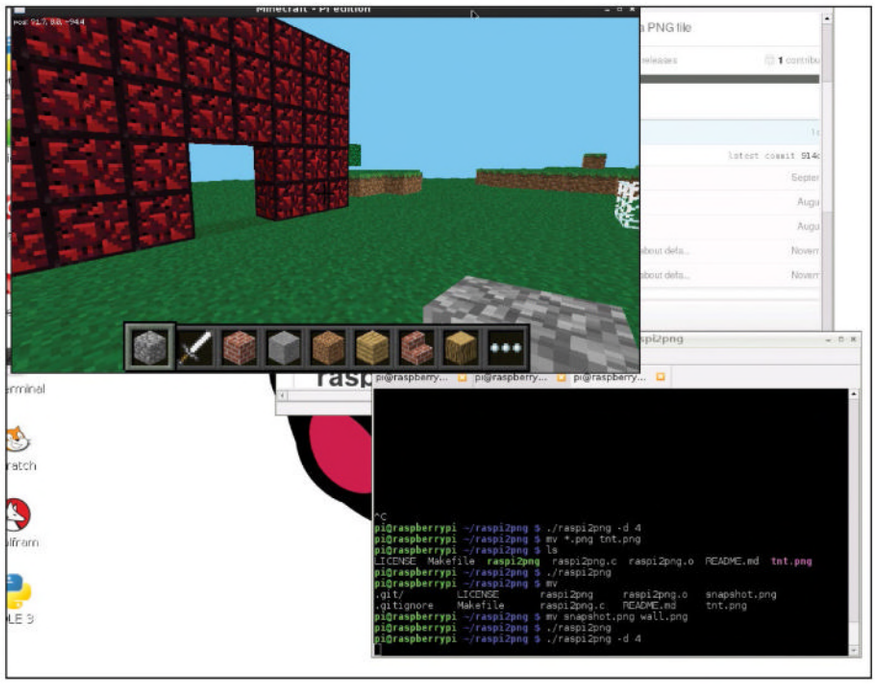
We can also get data on what our character is standing on. Go back to your Python session and type:

```
curBlock = mc.getBlock(x, y - 1, z)
mc.postToChat(curBlock)
```

Here, **getBlock()** returns an integer specifying the block type: 0 refers to air, 1 to stone, 2 to grass, and you can find all the other block types in the file **block.py** in the **~/picraft/minecraft** folder we created earlier. We subtract 1 from the y value because we are interested in what's going on underfoot – calling **getBlock()** on our current location should always return 0, because otherwise we would be embedded inside something solid or drowning.

As usual, running things in the Python interpreter is great for playing around, but the grown-up way to do things is to put all your code into a file. Create the file **~/picraft/gps.py** with the following code.

```
import minecraft.minecraft as minecraft
import minecraft.block as block
mc = minecraft.Minecraft.create()
oldPos = minecraft.Vec3()
while True:
    playerTilePos = mc.player.getTilePos()
    if playerTilePos != oldPos:
        oldPos = playerTilePos
        x = playerTilePos.x
        y = playerTilePos.y
        z = playerTilePos.z
        t = mc.getBlock(x, y – 1, z)
        mc.postToChat(str(x) + ' ' + str(y) + ' ' + str(z) + ' ' + str(t))
```



All manner of improbable structures can be yours.

Now fire up *Minecraft*, enter a world, then open up a terminal and run your program:

```
$ python gps.py
```

The result should be that your co-ordinates and the BlockType of what you're standing on are displayed as you move about. Once you've memorised all the BlockTypes (joke), [Ctrl]+[C] the Python program to quit.

We have covered some of the 'passive' options of the API, but these are only any fun when used in conjunction with the more constructive (or destructive) options. Before we sign off, we'll cover a couple of these. As before, start *Minecraft* and a Python session, import the *Minecraft* and block modules, and set up the **mc** object:

```
posVec = mc.player.getTilePos()
x = posVec.x
y = posVec.y
z = posVec.z
for j in range(5):
    for k in range(x - 5, x + 5)
        mc.setBlock(k, j, z + 1, 246)
```

Behold! A 10 x 5 wall of glowing obsidian has been erected adjacent to your current location. We can also destroy blocks by turning them into air, so we can make a tiny tunnel in our obsidian wall like so:

```
mc.setBlock(x, y, z + 1, 0)
```

Assuming, of course, that you didn't move since inputting the previous code.

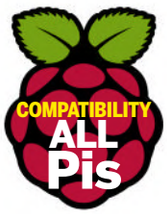
Over the page, we'll see how to build and destroy structures, dabble with physics, rewrite some of the laws thereof, and generally go crazy within the confines of our 256 x 256 x 256 world. Till then, why not try playing with the **mc.player.setPos()** function? Teleporting is fun, after all.

Quick tip

Check out Martin O'Hanlon's website **www.stuffaboutcode.com**, which includes some great examples of just what the API is capable of.

Minecraft: Make a Pi trebuchet

Build your labour of love and then blow it sky-high with some pyrotechnic code, a stash of TNT and an age-old siege machine.



Now that we're au fait with the basics of the API, it's time to get crazy creative. Building a house is hard, right? Wrong. With just a few lines of sweet Python your dream home can be yours. Provided your dream home is a fairly standard box construction, that is. If your dreams are wilder, all it takes is more code. You will never have to worry about planning permission, utility connection, chancery repair contributions or accidentally digging up a neolithic burial ground (unless you built it first).

It never actually rains in *Minecraft Pi*, so a flat-roof construction will happily suit our purposes just fine. We kick off proceedings by defining two corners for our house: v1 is the block next to us in the x direction and one block higher than our current altitude, whereas v2 is an aesthetically pleasing distance away:

```
pos = mc.player.getTilePos()
v1 = minecraft.Vec3(1,1,0) + pos
v2 = v1 + minecraft.Vec3(10,4,6)
```

Now we create a solid stone cuboid between these vertices and then hollow it out by making a smaller interior cuboid full of fresh air:

```
mc.setBlocks(v1.x,v1.y,v1.z,v2.x,v2.y,v2.z,4)
mc.setBlocks(v1.x+1,v1.y,v1.z+1,v2.x-1,v2.y,v2.z-1,0)
```

Great, except our only means of egress and ingress is via the rather generous skylight, and a proper floor wood (geddit?) be nice. If you're standing in a fairly flat area, you'll notice that the walls of your house are hovering one block above ground level. This space is where our floor will go. If your local topography is not so flat, your house may be embedded in a hill, or partly airborne, but don't worry – the required terraforming or adjustments to local gravity will all be taken care of. Let's make our rustic hardwood floor:

```
mc.setBlocks(v1.x,v1.y-1,v1.z,v2.x,v1.y -1,v2.z,5)
```

The windows are just another variation on this theme:

```
mc.setBlocks(v1.x,v1.y+1,v1.z+1,v1.x,v1.y+2,v1.z+3,102)
mc.setBlocks(v1.x+6,v1.y+1,v1.z,v1.x+8,v1.y+2,v1.z,102)
mc.setBlocks(v2.x,v1.y+1,v1.z+1,v2.x,v1.y+2,v1.z+3,102)
mc.setBlocks(v1.x+2,v1.y+1,v2.z,v1.x+4,v1.y+2,v2.z,102)
```

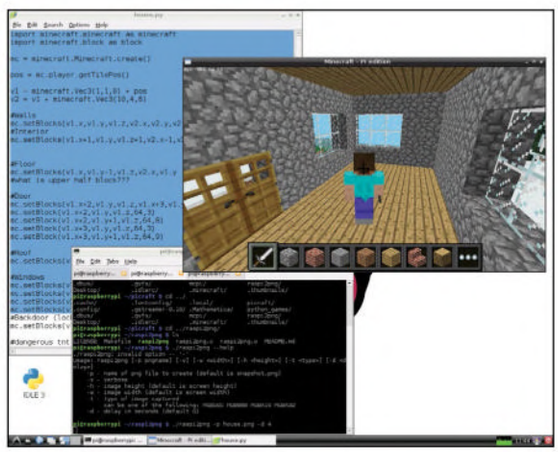
The roof uses the special half block 44, which has a few different types. Setting the blockType makes it wooden, matching our floor:

```
mc.setBlocks(v1.x,v2.y,v1.z,v2.x,v2.y,v2.z,44,2)
```

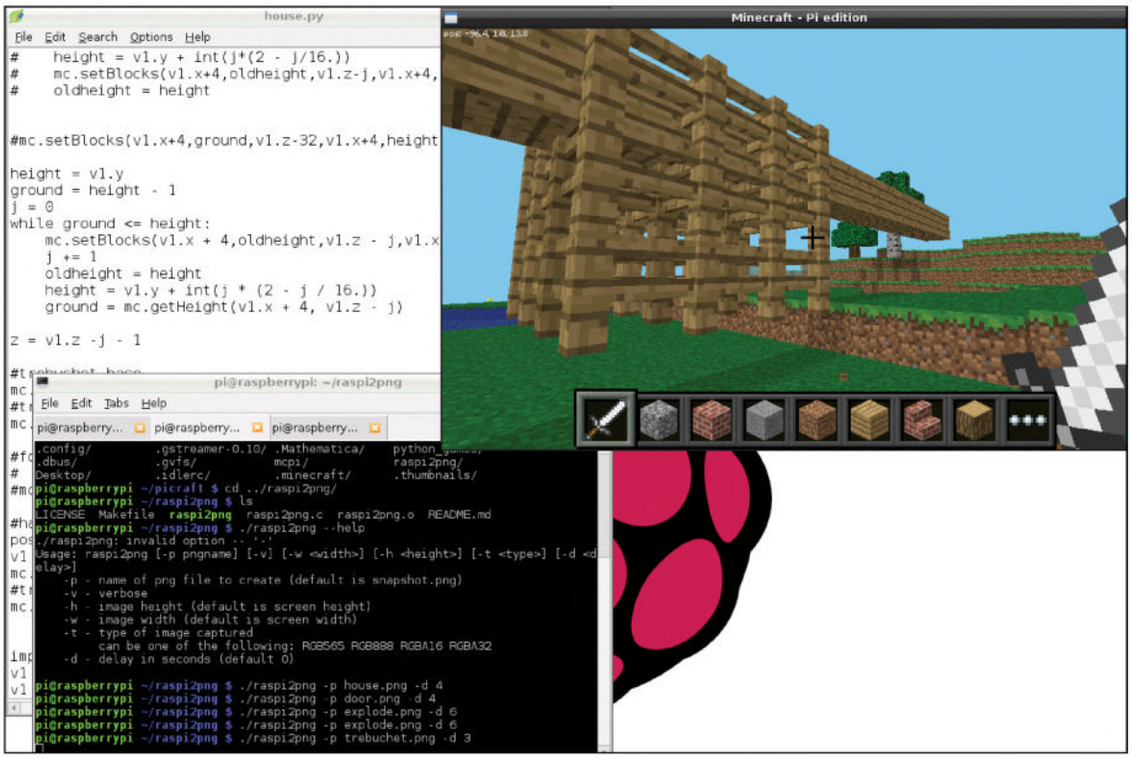
The door is a bit more complicated, the gory details are in the box on page 62, but the following three lines do the job:

```
mc.setBlocks(v1.x+2,v1.y,v1.z,v1.x+3,v1.y,v1.z,64,3)
mc.setBlock(v1.x+2,v1.y+1,v1.z,64,8)
mc.setBlock(v1.x+3,v1.y+1,v1.z,64,9)
```

Having lovingly and laboriously constructed our new property, the next step is to come up with new and inventive ways of destroying it. We have already mentioned that TNT



A house. Now let's blow it up.



It doesn't look like much, but just you wait...

can be made live, so that a gentle swipe with a sword (or anything really) will cause it to detonate. It would be trivial to use setBlocks to fill your house with primed TNT, but we can do much better. Allow us to introduce our beta trebuchet.

Rather than simulating a projectile moving through space we will instead trace its parabolic trajectory with hovering TNT. Detonating the origin of this trajectory will initiate a most satisfying chain reaction, culminating in a big chunk of your house being destroyed. First we will cover some basic two-dimensional mechanics. In the absence of friction, a projectile will trace out a parabola determined by the initial launch velocity, the angle of launch and the local gravitational acceleration, which on earth is about $9.81ms^{-2}$.

As a gentle introduction, we will fiddle these constants so that the horizontal distance covered by this arc is exactly 32 blocks and at its peak it will be 16 blocks higher than its original altitude. If blocks were metres, then this fudge would correspond to a muzzle velocity just shy of $18ms^{-1}$, and an elevation of 60 degrees. We will only worry about two dimensions, so the arc will be traced along the z axis with the x co-ordinate fixed just next to our door. This is all summed up by the simple formula y = z(2- z/16), which we implement this way:

```
for j in range(33):
  height = v1.y + int(j*(2 – j/16.))
  mc.setBlock(v1.x+4,height,v1.z-j,46,1)
```

The final argument sets the TNT to be live, so have at it with your sword and enjoy the fireworks. Or maybe not: the explosions will, besides really taxing the Pi's brain, cause some TNT to fall, interrupting the chain reaction and preserving our lovely house. We don't want that, so we instead use the following code:

```
height = v1.y
ground = height - 1
j = 0
while ground <= height:
  mc.setBlocks(v1.x + 4,oldheight,v1.z - j,v1.x + 4,height,v1.z - j,46,1)
  j += 1
  oldheight = height
  height = v1.y + int(j * (2 - j / 16.))
  ground = mc.getHeight(v1.x + 4, v1.z - j)
```

This ensures that our parabola is gap-free and also mitigates against the TNT arc-en-ciel terminating in mid-air. We have dealt with this latter quandary using the **getHeight()** function to determine ground level at each point in the arc, and stop building when we reach it. Note that we have to make the **getHeight()** call before we place the final TNT block, because the height of the world is determined by the uppermost non-air object, even if the said object is hovering.

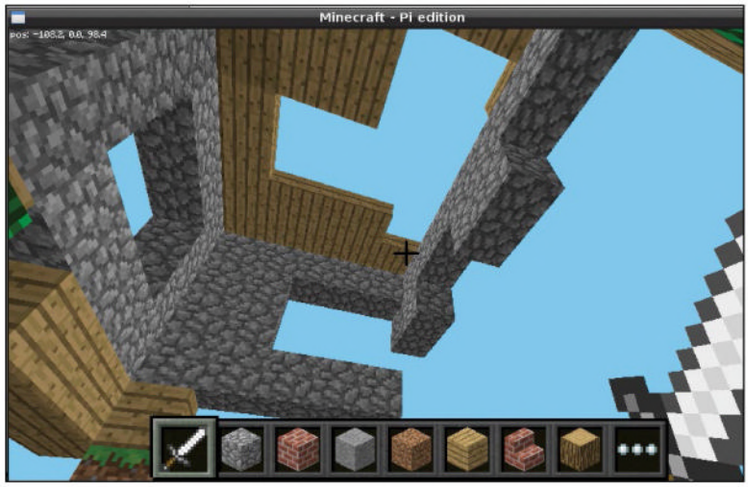
If our construction exceeds the confines of the *Minecraft* world, you could just build another house in a better situation, or you could change **v1.z - j** to **max(-116,v1.z-j)** in the above loop, which would make a vertical totem of danger right at the edge of the world. Now that we have our trajectory, we can add the mighty siege engine:

```
z = v1.z -j - 1
mc.setBlocks(v1.x + 3, oldheight, z + 10, v1.x + 6, oldheight + 2, z + 7,85)
mc.setBlocks(v1.x + 4, oldheight + 2, z + 12, v1.x + 4, oldheight + 2, z + 1, 5)
```

Up until this point, we have aligned everything along a particular axis. Our house (before you blew it up) faces the negative z direction, which might be akin to facing south, and this is also the direction along which our explosive parabola is traced. Naturally, we could rotate everything 90 degrees and the code would look much the same (modulo some judicious permuting of x,y and z and +/-) though your house would »

Quick tip

The trebuchet code was inspired by the amazing Martin O'Hanlon and his Raspberry Pi-based projects on **www.stuffabout code.com**.



Swiss cheese, baby! (Your house may need some repairs after this tutorial.)

look a bit funny built on its side. Things get complicated if we want to shed the yoke of these grids and right angles, to work instead with angles of our choosing. The problem is how to approximate a straight line when our units are blocks of fixed orientation rather than points.

A general three-dimensional **drawline()** function will prove invaluable in your subsequent creations, enabling you to create diverse configurations from parallelepipeds to pentagrams. What is required is a 3D version of the classical Bresenham algorithm. Pi guru Martin O'Hanlon's github contains several wonderful *Minecraft Pi Edition* projects, including a mighty cannon from which this beginner project takes its inspiration. Martin has a whole Python drawing class, which includes the aforementioned 3D line algorithm, but once you understand the 2D version, the generalisation is reasonably straightforward.

Let us imagine we are in a Flatland-style *Minecraft* world in which we wish to approximate the line in the (x,y) plane connecting the points (-2,-2) and (4,1). This line has the equation y = 0.5x - 1. The algorithm requires that the gradient of the line is between 0 and 1, so in this case we are fine. If we wanted a line with a different slope, we can flip the axes in such a way as to make it conform. The crux of the algorithm is the fact that our pixel line will fill only one pixel (block) per column, but multiple pixels per row. Thus as we rasterise pixel by pixel in the x direction, our y co-ordinate will either stay the same or increment by 1. Some naïve Python would then be:

```
dx = x1 – x0
dy = y1 – y0
y = y0
error = 0
grad = dy/dx
for x in (x0,x1):
  plot(x,y)
  error = error + grad
  if error >= 0.5:
    y += 1
    error -= 1
```

where **plot()** is some imaginary plotting function and **grad** is between 0 and 1. Thus we increment y whenever our error term accumulates sufficiently, and the result is the image that meets your gaze.

Bresenham's trick was to reduce all the calculations to integer operations, which were far more amenable to 1960s hardware. Nowadays we can do floating point calculations at speed, but it is still nice to appreciate these novel hacks. The floating point variables grad and error arise due to the division by **dx**, so if we multiply everything by this quantity and work around this scaling, we are good to go.

To get this working in three dimensions is not so much of an abstractive jump. We first find which is the dominant axis (the one with the largest change in co-ordinates) and flip things around accordingly, moving along the dominant axis one block at a time and incrementing the co-ordinates of minor axes as required. We have to pay careful attention to the sign of each co-ordinate change, which we store in the variable **ds**. The **ZSGN()** function returns 1, -1 or 0 if its argument is positive, negative or zero respectively; we have left coding this as an exercise for you. We make extensive use of a helper function **minorList(a,j)** which returns a copy of the list a with the jth entry removed. We can code this using a one-liner thanks to lambda functions and list slicing:

```
minorList = lambda a,j: a[:j] + a[j + 1:]
```

Our function **getLine()** will take two vertices, which we will represent using three-element lists, and return a list of all the vertices in the resulting 3D line. All of this is based on Martin's code, for which we should all be grateful. The first part of it initialises our vertex list and deals with the easy case where both input vertices are the same...

Here our line is just a single block:

```
def getLine(v1, v2):
  if v1 == v2:
```

Quick tip

You can do all the coding here in the interpreter, but copying errors are frustrating. Thus it might be easier to put it all in a file called **house.py** which you can execute with **python house.py** while Minecraft is running.

Double door details

Putting doors into our house is our first encounter with the additional **blockData** parameter. This is an integer from 0 to 15 and controls additional properties of blocks, such as the colour of wool and whether or not TNT is live. Our door occupies four blocks and is aligned in the x direction. It's recessed slightly back from the surrounding walls, closed, and has the handles helpfully placed towards the middle. These properties are controlled by various bits of the **blockType**. We number the four bits from the right-most bit 0 to the left-most bit 3 and in little-endian notation so that 8 is represented in binary as 1000. Bit 3 is set if the block is part of the top section of a door. If this is the case, bit 0 is the only other bit of concern – it determines the placement of the handles and hinges. Top sections of doors thus have blockType 8 or 9.

For the bottom sections we have the following bit assignments:

bit 3.....off
bit 2.....door is open
bit 1.....door recessed
bit 0.....alignment (off=x, on=z)

The top sections must be placed after the bottom ones, because they inherit their properties from their inferiors.

Doors are always a good idea for those wishing to avoid claustrophobia/death.

```
    vertices.append([v1])
```

After this it gets a bit ugly. We set up the previously mentioned list of signs **ds**, and a list of absolute differences (multiplied by two) **a**. The **idx =** line is technically bad form – we want to find our dominant axis, thus the index of the maximum entry in a. Using the **index()** method together with **max** means we are looping over our list twice, but since this is such a short list we shan't worry – it looks much nicer this way. We refer to the dominant co-ordinates by **X** and **X2**. Our list **s** is a re-arrangement of **ds**, with the dominant co-ordinate at the beginning. And there are some other lists to keep track of the errors. The variable **aX** refers to the sign of the co-ordinate change along our dominant axis.

```
  else:
    ds = [ZSGN(v2[j] - v1[j]) for j in range(3)]
    a = [abs(v2[j]-v1[j]) << 1 for j in range(3)]
    idx = a.index(max(a))
    X = v1[idx]
    X2 = v2[idx]
    delta = a[idx] >> 1
    s = [ds[idx]] + minorList(ds,idx)
    minor = minorList(v1,idx)
    aminor = minorList(a,idx)
    dminor = [j - delta for j in aminor]
    aX = a[idx]
```

With all that set up we can delve into our main loop, in which vertices are added, differences along minor axes examined, errors recalculated, and major co-ordinates incremented. Then we return a lovely list of vertices.

```
    loop = True
    while(loop):
      vertices.append(minor[:idx] + [X] + minor[idx:])
      if X == X2:
        loop = False
      for j in range(2):
        if dminor[j] >= 0:
          minor[j] += s[j + 1]
          dminor[j] -= aX
        dminor[j] += aminor[j]
      X += s[0]
  return vertices
```

To conclude in style, we will test this function by making a mysterious and precarious beam of wood next to where we are standing as a fitting testament to your wondrous labours this day, padawan.

```
v1 = mc.player.getTilePos() + minecraft.Vec3(1,1,0)
v1 = minecraft.Vec3(1,1,0) + pos
v2 = v1 + minecraft.Vec3(5,5,5)
bline = getLine([v1.x,v1.y,v1.z],[v2.x,v2.y,v2.z])
for j in bline:
  mc.setBlock(j[0],j[1],j[2],5)
```

And that's it! Using *Minecraft Pi Edition* is a great way to practise how to code in Python. You'll be having so much fun you'll probably not even realise you're learning!

Now we can escape the gridlock and build at whatever angles our heart desires.

Don't try this at home, kids.

Python: 2048 in Minecraft

Discover how you can implement an awesome *2048* clone in *Minecraft*.

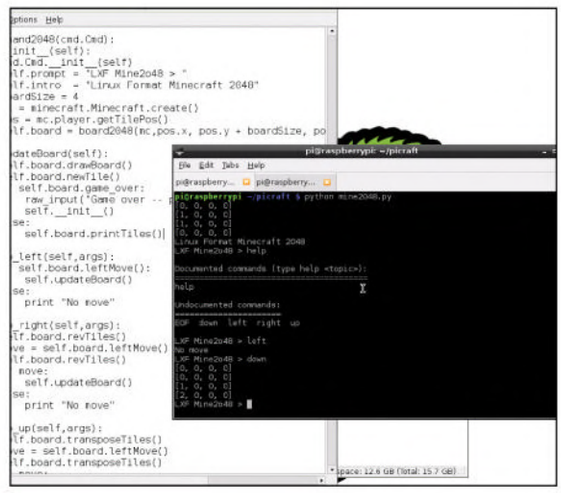
If you haven't yet witnessed the opium-like moreishness of the Android/iOS game *2048*, then consider yourself lucky. Once they get a taste of the thrilling addition of successive powers of two, then addicts, like so many hopeless lovers, are powerless to abandon the pursuit of the cherished 2048 tile, always just beyond reach. Granted, much of the pleasure comes from the touch interface: there is an innate pleasure in orchestrating an elegant and board-clearing sequence of cascades with just a few well-executed swipes. Be that as it may, the game has simple rules and is based on blocks. As such, it's perfect material for the next instalment in our *Minecraft:Pi Edition* series.

We'll worry about how to draw and move the blocks later. The first challenge is to understand the algorithm underlying the game. So dutifully read the rules in the box – they might seem clumsily worded or overly complex, but rest assured they have been carefully crafted for easy translation to Python. Sometimes it's easier to visualise the situation using pseudocode – this way we can see the shape of the program without getting bogged down in details.

```
for each row:
    move tiles left, clearing empty space

for each row:
    for column 0 to 2:
        if tile[row,column + 1] = tile[row,column]:
            double tile[row, column]
            for x in column + 1 to 2:
                tile[row][x] = tile[row, x + 1]
            empty tile[row,3]

insert random 2 or 4 tile
if no move can be done:
    game over
```

Our task now is to flesh out the English bits of the above code, but to do this we need to state how we're going to represent the board and tiles. This will be done using a 4x4 array called **tiles**. The first co-ordinate will be the row number, and the second the column number, numbering the rows from top-to-bottom and the columns from left-to-right. Python uses 0-based lists, so **tiles[0][0]** will be the the top-left hand tile. Non-empty tiles will all be powers of 2, so we can use the exponent rather than the actual value here – for example, **32** will be represented by **5**. We can represent empty tiles using 0, so our array **tiles** is comprised entirely of (small) integers. Lovely.

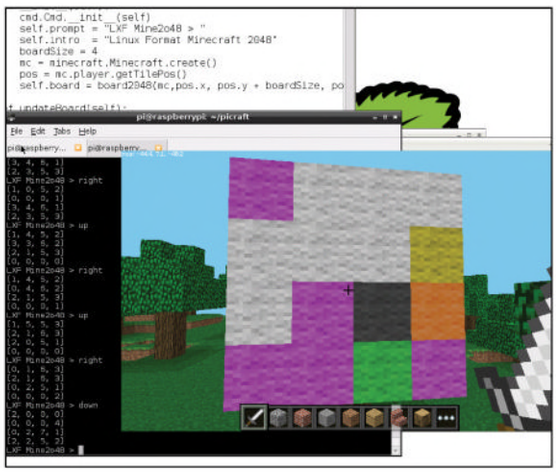
Now we know what we're dealing with, we can set about doing some actual Python. We've gone for an object-oriented approach here, which means that there will be a lot of **self** flying around – but don't worry, it makes it easier for the *Minecraft* functions to talk to those to do with the board and



You even get a handy help command, but it kind of looks like a cheat for *Sonic the Hedgehog*...

Quick tip

You could make this project a bit less of a *Zork*-esque text adventure by writing a control system based on player moves. You'd want to reset the player's position after each move, but in principle it's quite a simple modification.



The standard wool colour numbers don't really do high value blocks justice: e.g. 128 is represented by black.

the command interpreter. Honest.

Let's look at line 2 of our pseudocode, in which we move tiles in the row left, occupying any available empty tiles. Python does have a **remove()** method for lists, but it only works on one element at a time, so we'll clear the zeroes from our row the old-fashioned way. We don't waste time fiddling with empty rows, and we also need to keep track of whether we actually moved anything (through the boolean **isMove**). Also don't worry about the three other possible moves for now; we can cater for them later with a very cunning trick.

```
def leftMove(self):
    isMove = False
    for row in range(self.boardSize):
        j = 0
        while j < self.boardSize - 1:
            # check rest of row non-empty
            row_empty = True
            for k in range(j,self.boardSize):
                if self.tiles[row][k] != 0:
                    row_empty = False
            if self.tiles[row][j] == 0 and not row_empty:
                for k in range(j,self.boardSize - 1):
                    self.tiles[row][k] = self.tiles[row][k + 1]
                self.tiles[row][self.boardSize - 1] = 0
                isMove = True
            else:
                j += 1
```

Now we can deal with the situation in the second block in the pseudocode, finding two horizontally-adjacent tiles are the same.

```
    for row in range(self.boardSize):
        for column in range(self.boardSize - 1):
            if self.tiles[row][column] == self.tiles[row][column + 1]
\
            and self.tiles[row][column] != 0:
                self.tiles[row][column] += 1
                for k in range(column + 1, self.boardSize - 1):
                    self.tiles[row][k]= self.tiles[row][k + 1]
                self.tiles[row][self.boardSize - 1] = 0
                isMove = True
    return isMove
```

Things turn out to be simpler if we split off the third pseudocode block into its own function, **newTile()**. We first need a list of co-ordinates of all the empty tiles; if there's more than one then it's certainly not Game Over, but even if there's only a single space (which will get filled by a new tile) there still may be possible moves. We use a couple of functions from the **random** module to help decide the where and what is of the new tile.

```
def newTile(self):
    empties = []
    self.game_over = True
    for j in range(self.boardSize):
        for k in range(self.boardSize):
            if self.tiles[j][k] == 0:
                empties.append([j,k])

    if len(empties) > 1:
        self.game_over = False

    rnd_pos = random.choice(empties)
    rnd_n = random.randint(1,2)
    self.tiles[rnd_pos[0]][rnd_pos[1]] = rnd_n
    self.drawBoard()
```

The **drawBoard()** function is what will actually place the blocks in *Minecraft* world. Checking whether moves are possible after the tile is added is a little awkward:

```
    # check row neighbours
    for j in range(self.boardSize):
        for k in range(self.boardSize - 1):
            if self.tiles[j][k] == self.tiles[j][k + 1]:
                self.game_over = False

    # check col neighbours
    for j in range(self.boardSize):
        for k in range(self.boardSize - 1):
            if self.tiles[k][j] == self.tiles[k + 1][j]:
                self.game_over = False
```

»

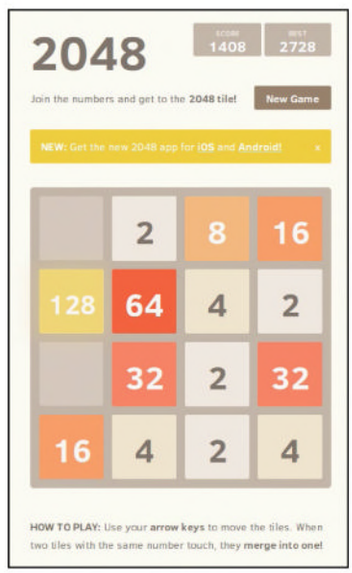
Quick tip

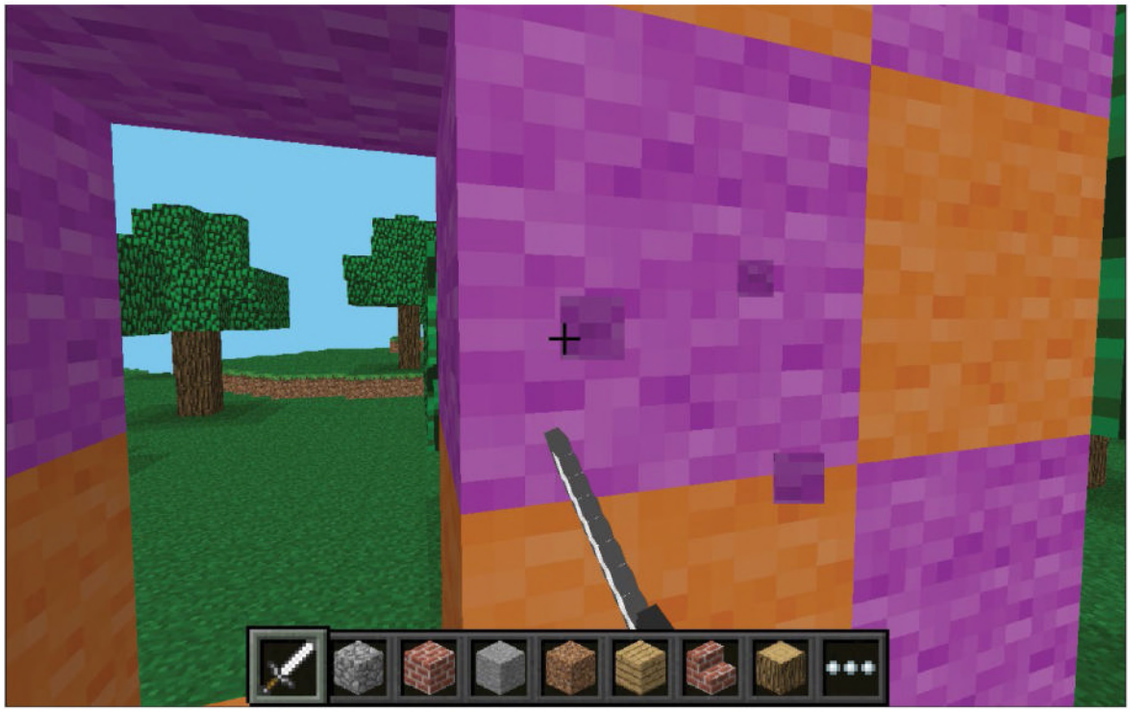
The board is updated twice for a successful move, before and after any tile merging. If you were feeling adventurous, you could use this to implement a two step animation by spacing out the tiles somehow. We were not feeling adventurous.

2048: An Open Source Odyssey

The *2048* game was written by young Italian programmer Gabriele Cirulli, who wanted to see if he could write a game in a weekend. The fruits of his labour proved hugely popular, accruing some 4 million downloads within a week of its release in March 2014. Numbers soared after mobile versions were released the following May, with up to 50,000 simultaneous players at its peak.

The game is described as a clone of Veewo Studio's *1024* and conceptually similar to the indie title *Threes!*. Rather than profit from his success, Cirulli's blog says he "didn't feel good about keeping [the code] private, since it was heavily based off someone else's work." Thus it is available to all at **https://github.com/gabrielecirulli/2048**. In the spirit of open source, this spurred all kinds of additions and modifications and further depleted global productivity.





It can be frustrating, and sometimes Steve's temper gets the better of him

Amazingly, that's the guts of the code all dealt with, but we still need a means by which to input our moves, and of course we need to render the results in *Minecraft* world.

Master and commander

We'll use the **cmd** module to provide a simple command line interface to our game, all the code is available from **www.pastebin.com/5zEZUfBS** or **www.linuxformat.com/files/mine2048.py.zip**. We shall have a command for initialising the board, four directional commands and a command to quit, Ctrl-D, aka the end of file (EOF) character or the quick way to end a Python session. We even get a help command for free (see picture, page 80). When you enter a command, for example **left**, the cmd module will run the function with that name prefixed by **do_** hence we have a function **do_left()** which calls the **leftMove()** function above. This function is really part of our **board** class.

The **do_left()** function will check if the left move is valid, and if so update the board, via the imaginatively-titled **updateBoard()** function, which in turn will add a new tile and decide whether or not the game is up. If it is all over, then we cheekily use **raw_input()** as a means to wait for the user to press Return.

The cmd module works by subclassing its Cmd class. Any class you write that instantiates this will inherit all of its functionality. We use the standard boilerplate to start things up when the program is executed:

```
if __name__ == "__main__":
    command2048().cmdloop()
```

This will instantiate our command interpreter when the program is run with:

```
$ python mine2048.py
```

Minecraft must be running when you do this, or you'll run into errors. Also you'll need to make sure you've copied the Python API from your *Minecraft* install (e.g. **~/mcpi/api/python/mcpi**) to a subdirectory called **minecraft** in the same directory as the **mine2048.py** file. But if you've been following our *Minecraft* series you'll be au fait with all this.

The **command2048** class's **__init__()** method sets up the basics, including setting up the **mc** object and using it to get the player's position. This provides a reference point for drawing our board, which we will arbitrarily decide to be 3 blocks away in the z direction. Make sure you're not looking at the back of the board, though, as everything will be backwards. For convenience, we also have a **printTiles()** function which draws the board at the command terminal.

Rules of the game

A key tenet of programming anything is knowing what the rules are – knowing exactly how your program should behave in a given set of circumstances. A vague idea of what should happen often won't cut it: we need specifics and exactitude, free of any semblance of ambiguity. With that in mind, let's consider how the game of *2048* works. We start with a 4x4 grid which has some 2s and some 4s on it at random locations. Initially two tiles are populated. The player can make one of four moves, shifting the tiles up, down, left or right. Suppose the player chooses left, then the following algorithm applies: For every row start by moving all the numbered tiles left so that any empty tiles are on the right of the row. Now we will look at each tile's situation and decide how the tiles will change. Starting with the tile on the left, if the tile to its right has the same value then the left-most of the pair will double in value, the rightmost of the pair will vanish, and all tiles to the right will move one position left. Repeat this process for the second and third tiles from the left. Repeat the whole process for the second, third and fourth rows. If there is still space then another 2 or 4 tile is added randomly in one of the available empty spaces. If no move is possible after this then it's Game Over.

For other directions, the same algorithm applies with strategic substitution of row with column and right with left, up or down.

We display only the exponents since (a) we're lazy and (b) it looks nicer if most things occupy just a single character. Change it if you like.

The **drawBoard** function uses different colours of wool (blockType 35) to represent the content of our array **tiles**:

```
def drawBoard(self):
    for j in range(self.boardSize):
        for k in range(self.boardSize):
            self.mc.setBlock(self.x + k, self.y - j, self.z, 35, self.tiles[j][k])
```

This works reasonably, but feel free to add to it, for example making fancy glowing obsidian blocks for the higher valued ones. Note that our **row[0]** is the top one, so we count down, subtracting **j** from our y co-ordinate.

One Direction (or alternatively "The only way is Left")

But wait, an elephant in the room. We still haven't dealt with 75 percent of the moves allowed in *2048*. We could do this by copying the **leftMove()** function and painstakingly changing all the ranges and indices, and plus and minus signs, going through all the incorrect combinations until we find one that works. But that would be silly, and we don't tolerate such inefficiency here at Future Towers. Let us take a more grown-up approach. First, observe that moving the tiles to the right is exactly the same as reversing the rows, moving the tiles to the left, and then reversing the rows again. Reversing the rows is easy, once you figure out how to make a structural copy of our two-dimensional list:

```
def revTiles(self):
    oldTiles = [j[:] for j in self.tiles]
    for j in range(self.boardSize):
        for k in range(self.boardSize):
            self.tiles[j][k] = oldTiles[j][self.boardSize - k - 1]
```

So then we can make implement the (ahem) right move, just so:

```
def do_right(self,args):
    self.board.revTiles()
    move = self.board.leftMove()
    self.board.revTiles()
    if move:
        self.updateBoard()
    else:
        print "No move"
```

But the symmetry does not stop there. We can construct the Up move by transposing our tiles array (replacing each row by its respective column), then performing our **leftMove()** and transposing back again. Transposition is almost exactly as easy as row-reversal:

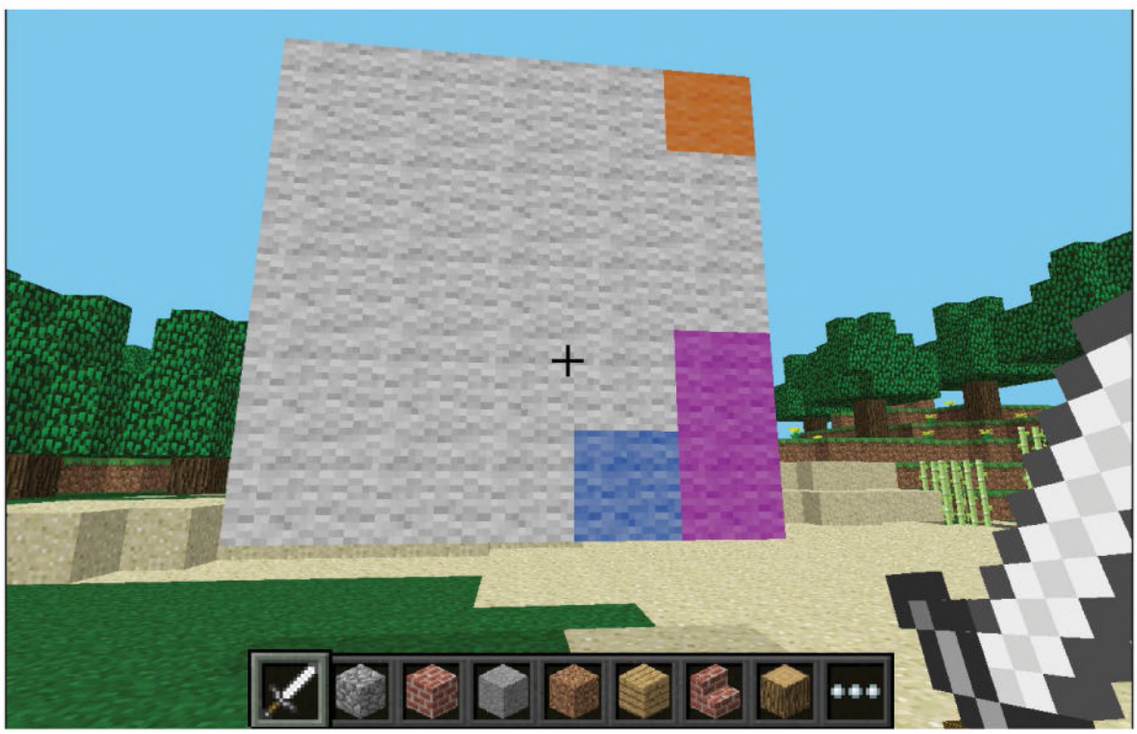
```
def transposeTiles(self):
    oldTiles = [j[:] for j in self.tiles]
    for j in range(self.boardSize):
        for k in range(self.boardSize):
            self.tiles[j][k] = oldTiles[k][j]
```

Similarly we can complete our directional command set, making the Down move by a combination of transposition, row reversal, left move, row reversal and transposition:

```
def do_down(self,args):
    self.board.transposeTiles()
    self.board.revTiles()
    move = self.board.leftMove()
    self.board.revTiles()
    self.board.transposeTiles()
```

Note that this is the same as transposition, right move and transposition.

Spotting wee tricks like this is great because they don't only make your program smaller (our provided code is about 160 lines long, not bad for a whole mini-game that can sap hours from your life). They also make it much easier to debug. The **leftMove()** function is by far the most complicated, so not having another three very similar functions is a considerable benefit.



You can change the boardSize variable, but it's a bit of a different game in a larger arena.

RASPBERRY Pi PROJECTS 2015

Get into Linux

Discover Linux 84
Welcome to the Terminal 92
Installing software 94
Core terminal commands 96
Using Raspbian packages 98
Getting help with Linux 100
Make your own Pi distro 103

What is Linux?

Let's delve deep into the world's best operating system and find out what makes it tick.

The word 'Linux' is one of the most used in this book, but what does it mean? It means different things to different people, from the purist who considers it to be the kernel, to the GNU advocate who sees it as a part of GNU/Linux and the new user who thinks it is another name for Ubuntu. In truth, Linux is all of these, depending on your point of view. Strictly speaking, the term Linux used alone refers to the kernel of the operating system, while GNU/Linux is the whole operating system, comprising the Linux kernel and GNU tools – either would be useless without the other (or one of its alternatives).

If you then add a collection of application software, along with some tools to manage the whole thing, you have a distro, such as Ubuntu or Raspbian.

There are lots of individual components that make up the operating system we know as Linux, but they are not that individual – they all have to fit together, so here we will try to explain how the whole is the sum of its parts, and what those parts do.

What is an OS? What is a distro?

An operating system can be defined as the software needed to enable the applications to run on the hardware – as such, it consists of several interleaved layers. At the heart we have the kernel, which interacts directly with the hardware through its drivers and allows other software to use that hardware. On top of that, we have various layers that handle things such as input devices, networking, sound and video. Normally, you don't need to know anything about this. It can be helpful to know some of it when things go wrong, but even then it is not essential, especially if you can find someone to fix the computer for you. But if you are reading this book, there is a good chance that you are interested in what is happening 'down below', so we will try to give you an idea of what goes where, and what it does when it gets there. A Linux distribution is just that, a way of distributing a Linux-based operating system and accompanying software. At the start, it was just the files the OS needs and a way of installing them on your computer. Along the way, distros acquired package managers, update tools, configuration GUIs and a host of other niceties. However, underneath the user-friendly (or not if you are a Gentoo user) gloss, all distros are still Linux.

The kernel

The nerve centre at the heart of your Linux operating system.

The kernel is the beating heart of the system, but what is it? The kernel is the software interface to the computer's hardware. It communicates with the CPU, memory and other devices on behalf of any software running on the computer. As such, it is the lowest-level component in the software stack, and the most important. If the kernel has a problem, every piece of software running on the computer shares in that problem.

The Linux kernel is a monolithic kernel – all the main OS services run in the kernel. The alternative is a microkernel, where most of the work is done by external processes, with the kernel doing little more than co-ordinating.

While a pure monolithic kernel worked well in the early days, when users compiled a kernel for their hardware, there are so many combinations of hardware nowadays that building them all into the kernel would result in a huge file. So the Linux kernel is now modular, the core functions are in the kernel file (you can see this in **/boot** as **vmlinuz-version**) while the optional drivers are built as separate modules in **/lib/modules** (the **.ko** files in this directory).

For example, Ubuntu 14.04's 64-bit kernel is 5MB in size, while there are a further 3,700 modules occupying over 100MB. Only a fraction of these are needed on any particular machine, so it would be insane to try to load them all with the main kernel. Instead, the kernel detects the hardware in use and loads the relevant modules, which become part of the kernel in memory, so it is still monolithic when loaded even when spread across thousands of files. This enables a system to react to changes in hardware. Plug in a USB memory stick and the **usb-storage** module is loaded, along with the filesystem module needed to mount it. In a similar way, connect a 3G dongle, and the serial modem drivers are loaded. This is why it is rarely necessary to install new drivers when adding hardware; they're all there just waiting for you to buy some new toys to plug in. Computers that are run on specific and unchanging hardware, such as servers, usually have a kernel with all the required drivers compiled in and module loading disabled, which adds a small amount of security.

If you are compiling your own kernel, a good rule of thumb is to build in drivers for hardware that is always in use, such as your network interface and hard disk filesystems, and build modules for everything else.

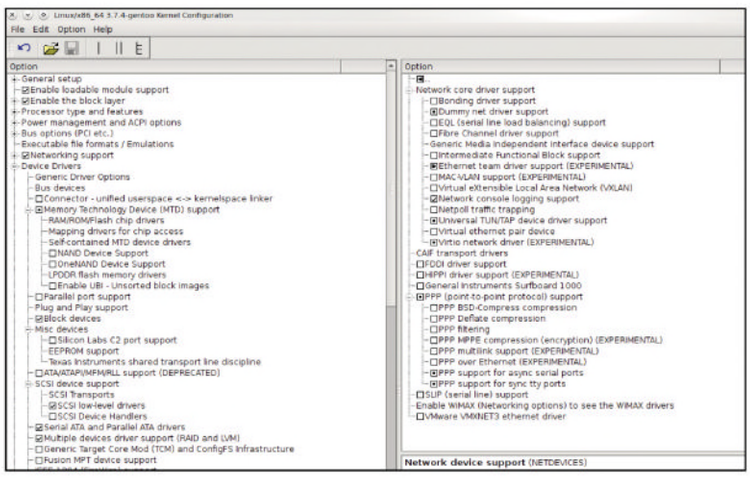
> **"So much hardware is supported, with no need to download drivers"**

Even more modules

The huge number of modules, most of which are hardware drivers, is one of the strengths of Linux in recent years – so much hardware is supported by default, with no need to download and install drivers from anywhere else. There is still some hardware not covered by in-kernel modules, usually because the code is too new or its licence prevents it being included with the kernel (yes ZFS, we're looking at you). The drivers for Nvidia cards are the best known examples. Usually known as third-party modules, although Ubuntu also refers to 'restricted drivers', these are installed from your package manager if your distro supports them. Otherwise, they have to be compiled from source, which has to be done again each time you update your kernel, because they are tied to the kernel for which they were built.

There have been some efforts to provide a level of automation to this, notably DKMS (Dynamic Kernel Module Support), which automatically recompiles all third-party modules when a new kernel is installed, making the process of upgrading a kernel almost as seamless as upgrading user applications.

Phrases that you will see bandied about when referring to kernels are "kernel space" and "user space". Kernel space is memory that can only be accessed by the kernel; no user programs (which means anything but the kernel and its modules) can write here, so a wayward program cannot corrupt the kernel's operations. User space, on the other hand, can be accessed by any program with the appropriate privileges. This contributes towards the stability and security of Linux, because no program, even running as root, can directly undermine the kernel. »



The number of options when building a kernel is truly staggering, and the majority of them relate to hardware device support. Aren't you glad we have distro maintainers to work it all out for us?

Boot sequence

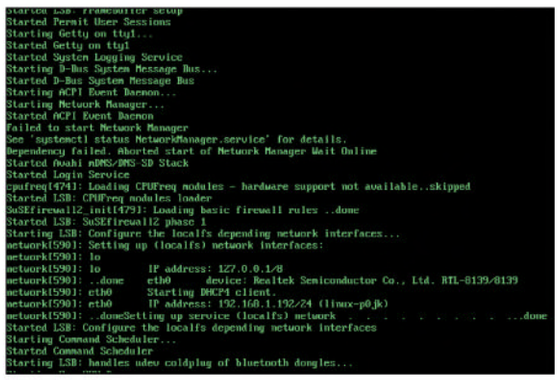
The mysterious sequence of flashes and beeps at startup.

Most distros go straight to a splash screen when booting, so we don't see what is happening. In fact, there's a lot going on both before the splash screen appears and then hidden by the screen. The BIOS starts up first, in the motherboard's hardware. It looks for a boot device and loads code from there.

In the case of a hard drive using the traditional DOS partition system, this is contained in the Master Boot Record (MBR) of the drive – just 512 bytes of storage. 64 bytes are used to hold the partition table for the drive (which is why only four primary partitions are available), leaving all of 446 bytes for the bootloader code, usually *Grub*. 446 bytes doesn't give room for much in the way of features, so all this code does is load the rest of the boot code from elsewhere on the disk, from a location set when the MBR code was installed by the bootloader.

The bootloader reads its configuration file, usually **/boot/grub2/grub.cfg**, for a list of boot options, and either displays a menu or goes straight to the default boot. Linux is not involved at this point, because there is only the bootloader code running. The configuration file tells the bootloader the location of the Linux kernel, and any initramfs file it may need, plus other settings such as the root partition, and whether to hide all this behind a splash screen. If you want to see what happens from here on, you can disable the splash screen on most distros by pressing the [E] key to edit the *Grub* menu entry, removing any quiet and splash options and pressing [F10] to continue booting.

> "The config file tells the bootloader the location of the Linux kernel"



Removing the boot splash screen shows the boot process in its entirety, including the status of the services being started.

Why use a ramdisk?

Most distros use an initramfs file. The main reason for this is that certain drivers have to be loaded with the kernel, particularly those needed to load anything for the disk drive (like SATA controllers and filesystem code). For a generic distro, building all possible options into the kernel would make it unworkably large, so everything is built as modules and those needed for booting are included in the initramfs. This is a type of ramdisk that is loaded with the kernel by the bootloader (using the BIOS routines to read it from the disk) containing all the files needed to mount the root partition. This is done using the kernel's device detection to decide which to load, and then control is passed to the hard disk proper. The initramfs is also used to load any splash screens, so they appear right at the start of the boot process.

Once the root partition is mounted, directly or via the initramfs, the init sequence starts in earnest. Traditionally, this involves running **/sbin/init**, which then runs everything else, as controlled by **/etc/inittab**, and is responsible for the list of service start-up messages you see scrolling up the console if you have no splash screen. This also allows you to see where the boot process is hanging or taking an unreasonable time if you experience such problems.

Newer options

Time moves on, and all of these systems are subject to change. On the latest hardware, the BIOS has been replaced by UEFI, although once the bootloader is installed you won't notice any difference. There are also moves to replace the

The GNU of GNU/Linux

The GNU (Gnu's not Unix) project predates Linux by several years. It had created most of the basic tools needed by a computer of the early 1980s – compilers, text editors, file and directory manipulation commands and much more – but did not have a usable kernel (some would say their kernel, GNU Hurd, is still not that usable).

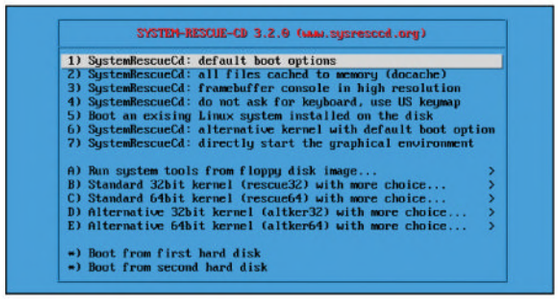
When Linus Torvalds started tinkering with his small project in 1991, he had a kernel without the tools to run on it. The two were put together and GNU/Linux was born – an operating system using the Linux kernel and the GNU toolset.

It is not only the programs in **/bin** and **/usr/bin** that come from GNU; **glibc** is the core C library used in Linux and it also comes from GNU. So just about any time you do anything on your computer, every time you type a command or click an icon, GNU software is being run at some level. No wonder the GNU die-hards get upset when we refer to our operating system as Linux and not GNU/Linux. It is worth mentioning that no one really denies the importance of the GNU aspect; calling the OS Linux rather than GNU/Linux has far more to do with convenience and laziness than politics – the full title is just too cumbersome.

Sorry Richard, we can't be bothered to say 'GNU slash' every time.

traditional SysVinit system, which has been around for many years. Ubuntu introduced *Upstart*, while Red Hat and Fedora are championing *Systemd*.

While they all do the same basic job – start the tasks that need to be started to get the OS running – the methods differ. The main difference is that SysVinit is sequential – each service must be started before the next is tackled; one slow starting service affects everything else. *Upstart* and *systemd* start services in parallel, avoiding such bottlenecks. Of course, there are those who argue that Linux is so stable that boot times are irrelevant – if you hibernate instead of shutting the machine down, reboots become rare.



› *Grub* is the most popular bootloader. Live CDs and USB sticks are more likely to use *isolinux* or *syslinux*.

Libraries

The logic behind sharing functions between programs.

Linux uses libraries to share code between applications. If a program, foo, uses functions that could be useful elsewhere, it places them in libfoo. Then when another program, such as the imaginatively named bar, wants to use the same function, it has only to link to libfoo rather than reinventing the wheel.

This means that there is only one copy of the code on your computer; if either project discovers a bug in the code, it will be fixed for both. It also introduces the concept of dependencies; both **foo** and **bar** depend on **libfoo** and are useless without it. This led to the phenomenon of 'dependency hell', where trying to install a program errored out with a list of unsatisfied dependencies, and trying to install those gave more dependencies. This is largely an unpleasant memory nowadays, as distro repositories became more comprehensive and package managers better at sorting things out.

If you stick with your distro's package manager and repositories, all dependencies should be taken care of without you even having to think about them. Try installing **somerandom.deb** or **somerandom.rpm** you downloaded from **www.somerandomsite.com** and you'll soon discover why you should let the package manager take care of things. One solution proposed to this is that all programs should be compiled statically. This means that instead of dynamically linking to the code in **libfoo** and loading when needed at run time, **foo** and **bar** each include the code in their executable programs. This means each program file is a standalone object with no dependencies; it can also make it a lot larger than it would be with dynamic linking and means that if a bug or security flaw is found in the **libfoo** code, both **foo** and **bar** will need to be recompiled and repackaged for your distro to fix the situation. Generally, dynamic linking is preferred on non-embedded devices, but there is one place where statically linked programs are useful: in an initramfs loaded at boot time, because it avoids the need to include libraries in the ramdisk image. If you are curious, you can see which libraries any program is linked to with the **ldd** command.

> **"You can see which libraries a program is linked to with ldd"**

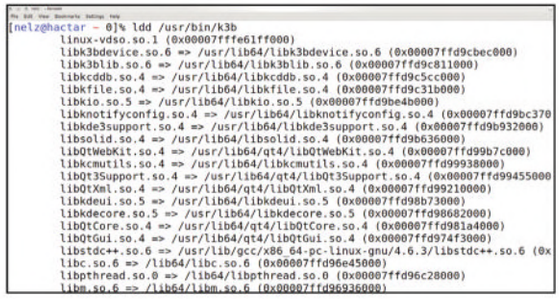
```
ldd /usr/bin/someprogram
```

shows all the libraries that program needs, and the libraries they need and so on, until you almost always end up at **libc** – the granddaddy of Linux libraries. »

Package managers

The great flexibility of Linux distributions means that most elements can be changed. Default applications, desktops, even kernels can be swapped around, so it's best to think of a Linux distribution such as Fedora or Ubuntu as merely a starting point for any customisation that you want to do.

The one thing that can't be changed so easily is the package manager, so the only way to try a different package manager is to try a different distro. Try comparing SUSE's *Yast* with Debian's *Synaptic*, for example, and you'll be amazed at the difference that such a fundamental tool can make to your experience of using Linux.



› Shared libraries enable a more efficient system, by sharing code between applications. Here are just some of the libraries the *K3b* disc burner links to.

Graphics

How your Linux box stays looking so tickety-boo.

The *X Window System* is the standard basis for providing a graphical interface. While the likes of KDE and Gnome provide the user interface and eye candy, it is through *X* that they communicate with the hardware. For many years, this was a mass of arcane configuration options and required a lengthy configuration file containing things such as modelines that specified the likes of pixel clock rates and sync frequencies.

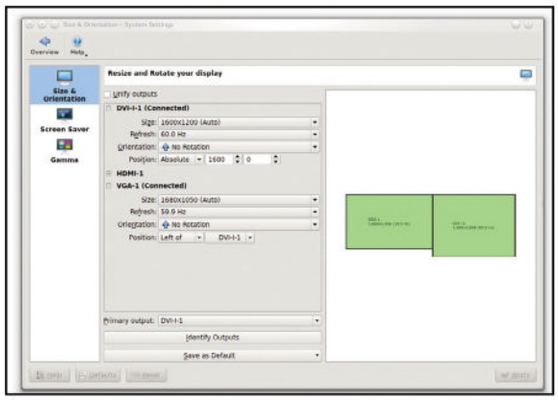
These days, most systems will run without any configuration file at all. Massive improvements in hardware detection mean that a system with a single video card and single display will 'just work'. You may need to install extra drivers to get 3D acceleration if you are using, for example, an Nvidia card, but otherwise you just boot the computer and start clicking your mouse. *X* has a client/server architecture. *X* itself runs as the server, maintaining the display; client programs then communicate with the server, telling it what to draw and where.

> **"These days, most systems will run without any configuration file"**

Legacy features

This may seem excessively complex, but it uses local sockets to communicate between the clients and server, so there is no significant performance hit. One clear advantage of this method is that the client and server do not have to be running on the same computer. You can connect to another computer by SSH and, providing the configuration gives permission for this, run a program on the remote computer and have its GUI displayed on your local screen. This is different from the likes of VNC because only the one application's window is displayed locally, and it only appears locally – not on the remote computer. A VNC connection mirrors the whole desktop on both computers.

Some consider the client/server architecture to be overly complex, so there are moves to develop more simple methods of running a graphical display. The most advanced is *Wayland*. This takes a different approach; not only is the old client/server setup gone, but *Wayland* leaves the rendering of windows and other display elements to the client applications, usually using *OpenGL* and *Cairo*. This simplifies *Wayland*; *X* contains a lot of legacy rendering code that's required by the *X* specification but never used. By giving control to the clients, *Wayland* can be lighter, more efficient and future-proof. It also means your graphical software has more control over how the GUI is displayed.



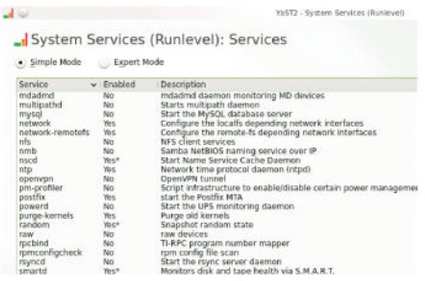
Tools such as KDE's monitor settings help with things like setting up dual monitors, but for a single display you shouldn't need to configure *X* at all.

Daemons

If you ever disable the splash screen most distros use to cover the boot messages, you will see a screen full of information about services being started. What are these services, and are they all necessary? The services are the programs that run in the background, making the computer as useful as it is. Some deal with networking, others handle hardware detection and configuration, while more are the traditional software services, or daemons, that provide functions to other programs when needed.

The answer to the second part of that question is most likely to be "no". While some of these services are used by almost all systems, such as the **syslog** daemon that handles writing information to system log files, others may not be needed. There is no need to start CUPS, the printing system, if you don't have a printer available. Similarly, the *MySQL* database server may not be needed, nor the SSH daemon if you have only one computer on your network. So spending half an hour experimenting could shave a second off your boot time.

You may also save some resources by not starting unnecessary services, but once loaded these daemons consume almost no system resources, and even the memory that they use can be swapped out if they are not called. So only disable those services you know you will never need. Having them patiently listening on a network port or socket makes the operation of your client programs that bit more efficient. Programs don't need to include, or load, code for opening, writing to and closing log files, they just call the **syslog()** function with the log text, and the daemon takes care of the rest. **Syslog** is an important service – when something goes wrong, this is often the first place to look, as most programs send error messages to the system log (usually at **/var/log/messages**).



It is possible to reduce your boot time by only running the services you need.

Why are background services called daemons? There are a few explanations; we prefer the story that daemons were beings in Greek mythology that handled tasks that the gods could not be bothered with.

Networking

How your computer talks to others.

Networking is core to Linux. Even on a standalone machine with no connection to a local network, let alone the internet, networking is still used. Many services run on a client/server model, where the server runs in the background waiting for instructions from other programs. Even something as basic as the system logger runs as a networked service, allowing other programs to write to log files. The *X* graphics system is also networked, with the *X* server running the desktop and programs, telling it what they want displayed. This is why it is so simple to run *X* programs on a remote desktop – as far as the system is concerned, there is no major difference between that and opening a window locally.

Running **ifconfig** will always show at least one interface, called lo with an address of **127.0.0.1** – this is used by the computer for talking to itself, which is regarded as a more sane activity for computers than people. Most other networking is based on TCP/IP, either over wired Ethernet or wireless, but there are other types of network in use. All distros and desktops include good tools for configuring and maintaining TCP/IP networks, from the fairly ubiquitous *NetworkManager* to individual tools such as Gnome's network configuration tool or OpenSUSE's *Yast*. More recent additions to the networking scene include 3G mobile communications and PAN (Personal Area Network) technologies such as Bluetooth. Using a 3G mobile broadband dongle is usually simple, either using *NetworkManager* or your desktop's PPP software. Yes, 3G modems really do work like modems using dialscripts and everything, but without the cacophony of squawks as you connect (younger readers should ignore the last statement). Most problems with 3G are caused by trying to set them up in a poor signal area rather than with either the hardware or software support in Linux.

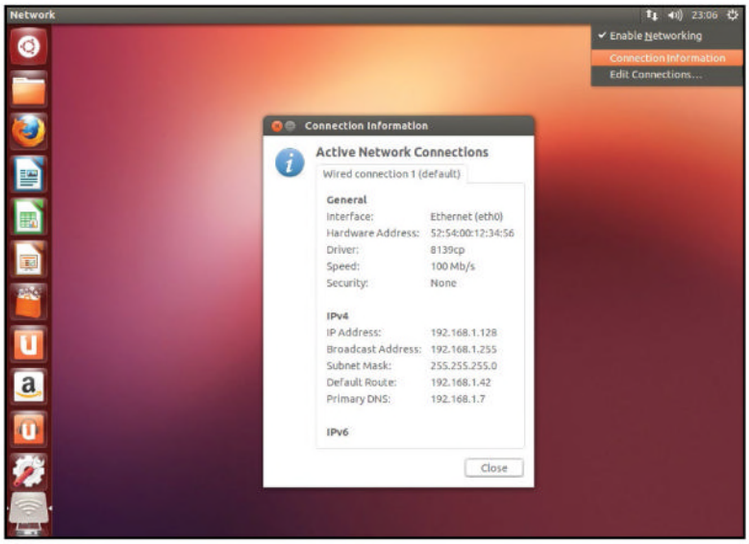
The protocol of kings

Bluetooth is becoming more important as mobile devices proliferate, and the number of input and output devices using it is increasing. It's not only phone and tablet users who benefit – a Bluetooth mouse and speakers can enhance the use of a laptop when at your desk, without having to plug everything in before you can start working. PulseAudio (see the section on sound) makes this easier, because it can switch between devices when they are detected. »

Storage

Storing data on a hard disk can involve several layers in itself. All physical storage (as opposed to network storage) in Linux revolves around block devices, so called because disks store data in blocks. A block device like **/dev/sda1** does indeed refer to blocks, physical areas on the disk, and a collection of them as a disk partition. On top of that we have a filesystem, which is how the data is stored in a sensible structure of directories and files, containing both data and metadata.

What's the difference? Let's say you save a file containing some text. The data in that file is the text, but the file has other attributes: there is the owner of the file, the time they created it, the time they last modified it, the time it was last read and who has permission to read or modify it. This is the information you see when you **ls -l** a file, or inspect its properties in your file manager, and this is stored by the filesystem. The standard filesystem in use nowadays is ext4, but there are alternatives such as ext3, ReiserFS, XFS, JFS and, of course, FAT and NTFS from the world of Windows.



› Networking is core to the operation of a Linux system. The localhost interface is set up automatically; for the rest we have programs such as *NetworkManager*.

Other Linuxes

Everything we have covered relates to Linux running on desktops, laptops and servers – traditional computer hardware if you like, but there are other environments where Linux is used. Many embedded devices, from routers to PVRs and set-top boxes, run Linux, and in many ways it's similar to the Linux we know and love. If your router allows SSH access, you will often feel at home as soon as you log in.

There is another class of device that has seen a huge uptake in recent years and runs a rather different Linux. No prizes for guessing we are referring to the smartphone and its tablet siblings, running Android. Android is Linux, it uses a Linux kernel, but it is not GNU/Linux. The kernel may be based on the same source code, but everything running on top is different. The principles are similar in some ways, but the implementation is very different – although you will find familiar command line tools if you can get to a shell prompt on your phone.

Desktops

Gnome, KDE Cinnamon, Unity – we'll just call it the user interface.

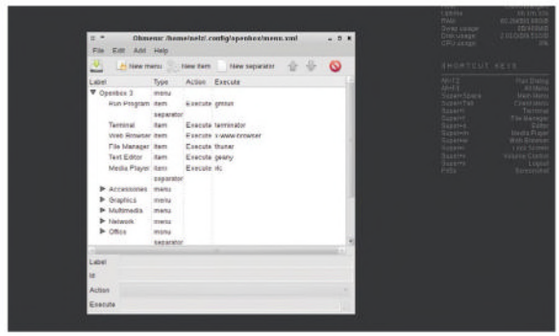
If you consider the kernel to be the lowest level of the system, the highest level is the user interface. Everything else, from the kernel through the drivers and hardware interfaces, is of no use until you can use the computer. This generally means a graphical desktop, and here we come across more layers. X (or maybe *Wayland* in the future) simply provides a blank canvas. You then need something to provide the niceties of a windowed interface, and that something is the window manager.

In the past, window managers were standalone systems, and there are still plenty of these available, such as *OpenBox* or *Enlightenment*, but nowadays they are often part of a larger desktop environment. Strictly speaking, a window manager is responsible for the handling of windows on the desktop, their opening, closing, placement and other manipulations. Over time, they grew to incorporate other features, such as taskbars and program launcher menus, until they developed into desktop environments.

Software collections

A desktop environment is simply a more or less integrated collection of utilities to provide the features needed to run a complete desktop. Running programs, manipulating their windows, keeping track of what is going on and enabling programs to communicate with one another are all features of desktop environments, but they still have a window manager at their heart – *KWin* for KDE and *Metacity* in Gnome to name but two.

What sets a desktop environment apart from a window manager is the level of integration.



There are also plenty of lightweight window managers, like *OpenBox* running on CrunchBang here.

This is particularly evident in KDE, where everything works around a common core, and programs not only communicate with one another, but an instance of one program can even be embedded in the window of another.

While it may not make much sense to use *KWin* on Gnome, you may want to try one of the more specialist window managers that offer greater control over window handling, or use a different method of displaying them. There are tiling window managers, like *awesome* and *xmonad*, that resize windows so they all fit on the desktop (KDE has its own option to behave like this). There are also window managers designed to be controllable with the keyboard, and minimal window managers that are useful for specialist systems that run a single program in a full-screen window and don't want any widgets cluttering up the place.

Gnome, KDE, Unity, Cinnamon, Mate – we aren't exactly short of choice when it comes to desktop environments, but how many of you have tried more than a couple?

Sound

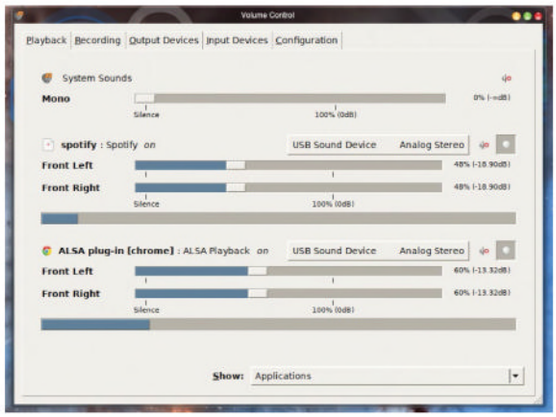
A once thorny subject.

The Linux sound system has been through many changes. We had OSS (Open Sound System) for years, before ALSA (Advanced Linux Sound Architecture) came along. ALSA is still with us, but we also have PulseAudio and Jack now.

ALSA supports multiple sound cards, each of which may have multiple inputs and outputs. It also provides support for hardware mixing, where the hardware supports it, and software mixing where it does not. This removes the need for sound managers, as provided by KDE and Gnome in the past, to enable more than one application to play sound at the same time. ALSA works at a low level, close to the hardware, so it gives low latency. Most hardware is directly supported now, and installing a distro should result in sound working from the first boot. ALSA is a combination of kernel code and user-space applications. It also provides an API so that other programs can control it directly, like the mixer control panels included with desktop environments.

PulseAudio performance

PulseAudio is a newer audio framework, but it is not a replacement for ALSA. Instead, it sits on top of the kernel audio system, providing a greater amount of control. It works as a server, accepting input from sources and forwarding it to sinks (output hardware or capture software). In many cases, the sink is ALSA, and the source can be an ALSA driver, too, for applications that don't directly support PulseAudio. Hence you can end up with an application sending output to an



If anyone tries to tell you that PulseAudio is complicated, it's best not to argue with them. Not that the complexity of this layout matters too much if it just works for you.

ALSA device, which intercepts the stream and routes it through PulseAudio back to ALSA. It is no surprise that many found PulseAudio complex. A good setup should render all of this chicanery transparent to the user, which is where we are now with distro installers and PulseAudio, so most of the time we are back at the 'just works' situation of ALSA, but with better support for multiple devices. ALSA supports multiple output devices, but the default is a global setting. PulseAudio allows you to direct music through speakers while using a Bluetooth headset for VOIP calls. It also allows for less complex but equally useful separation, such as separate volume settings for each application. PulseAudio is network-aware – it can be used to find other PulseAudio servers and play audio through their speakers – great for streaming music around the house.

JACK (Jack Audio Connection Kit) is a sound server designed for professional audio applications. Its forte is providing low-latency real-time connections between applications, for audio and MIDI data. It is not needed for typical desktop use, only for budding musicians.

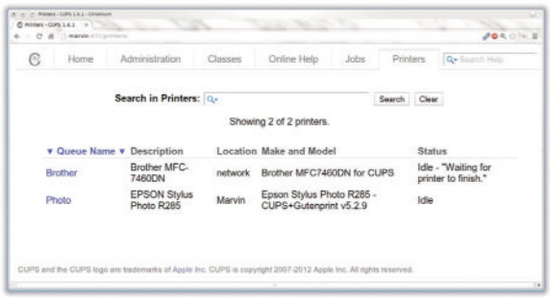
Printing

CUPS and drivers.

While open source encourages choice, and therefore several programs that do the same thing but in slightly different ways, there are some areas where one program is virtually unchallenged. *X.org* is used universally to provide graphical displays, and CUPS maintains a similar position in the printing arena.

If you have a printer attached to your Linux box, you need two things – CUPS and drivers for your printer. In many cases, these come together. CUPS is a server that sits in the background waiting for print requests. Just about any program that prints knows about the Internet Printing Protocol (IPP) that CUPS speaks. Hit 'Print' in your word processor or browser, and a window pops up showing your printer, and giving a choice if you have more than one.

The application only needs to send the data to be printed, usually as PostScript, to CUPS, which takes care of everything else – including waiting for you to remember to switch the printer on. CUPS does not even need to be running on the same computer; it is a networked service and a printer on one computer should be available to everyone on the same



Taking care of printers with CUPS is as easy as following a few links in a browser, thanks to its built-in web interface.

network. CUPS is useless without drivers that tell it how to speak to the printer. It includes a large number of drivers by default, and many more are available by installing the **gutenprint** driver package (so many that your distro may well have installed this by default). HP also provides a driver (and scanner) driver package called **hplip**, which you need if you use its products.

However, some printer companies insist on providing their own drivers instead of having them bundled with CUPS, usually for licensing reasons. In that case, you have the choice of trawling the printer manufacturer's website for a driver package suitable for your system and installing it separately. After that, the drivers should appear in CUPS and your distro's printer configuration tool. The other choice is to check with **linuxprinting.org** before buying a printer, and stick to the more enlightened manufacturers.

The Terminal: Getting started

There's no need to be afraid of the command line – we're here to help you with your first steps into the world of text commands.

It won't be long after starting to use Linux that you ask a question and the answer begins with, "Open a terminal and..." At this point, you may be thrown into an alien environment with typed commands instead of cheery-looking icons. But the terminal is not alien, it's just different. You are used to a GUI now, but you had to learn that, and the same applies to the command line. This raises an obvious question: "I already know how to use a windowed desktop, why must I learn something different?" You don't have to use the command line, almost anything you need can be done in the GUI, but the terminal has some advantages.

It is consistent The commands are generally the same on each distribution while desktops vary.

It is fast When you know what you are doing, the shell is much faster for many tasks.

It is repeatable Running the same task again is almost instant – no need to retrace all your steps.

There is more feedback Error messages from the program are displayed in the terminal.

Help is available Most commands provide a summary of their options, while man pages go into more detail.

You can't argue with the plus points, but what about the cons? Well, apart from not giving us pretty screenshots to brighten up the pages, the main disadvantage of the terminal is that you need to have an idea of the command you want to run, whereas you can browse the menus of a desktop system to find what you're after.

In this tutorial, we will look at the layout of the filesystem on Linux, and the various commands that you can use to manipulate it. On the following pages we will cover several other aspects of administering and using a Linux system from the command line.

What ls tells us about files

```
File  Edit  View  Bookmarks  Settings  Help
-rw-r--r-- 1 nelz users 1.3K Feb 14 12:46 FirefoxOS_emulation.
-rw-r--r-- 1 nelz users 350K Feb 12 10:53 FirefoxOS_emulat.png
-rw-r--r-- 1 nelz users    3 Feb 14 13:12 .issue
-rwxr-xr-x 1 nelz users 2K Jul 19  2013 lxfanswers
-rw-r--r-- 1 nelz users 1.1K Feb 14 12:46 Misty_MINT.txt
-rw-r--r-- 1 nelz users 523K Feb 28 12:44 New_laptop,_no_D.png
-rw-r--r-- 1 nelz users 3.4K Feb 14 12:41 New_laptop,_no_DVD.t
-rw-r--r-- 1 nelz users  72K Feb 14 12:50 Old_but_Smart.png
-rw-r--r-- 1 nelz users 2.9K Feb 14 12:02 Old_but_Smart.txt
-rw-r--r-- 1 nelz users  124 Feb 14 13:13 .order
-rw-r--r-- 1 nelz users 1.9K Feb 13 11:16 QR.txt
-rw-r--r-- 1 nelz users 2.8K Feb 27 20:48 Raspberry_capture.tx
drwxr-xr-x 2 nelz users    2 Mar  5 15:52 temp
-rw-r--r-- 1 nelz users 111K Feb 16 23:18 tut_1.png
-rw-r--r-- 1 nelz users  79K Feb 14 23:45 tut_2.png
-rw-r--r-- 1 nelz users 583K Feb 16 23:14 tut_3.png
-rw-r--r-- 1 nelz users 7.8K Feb 28 12:47 Tutorial183.txt
-rw------- 1 nelz users 8.1K Feb 14 23:31 tutorial.ncd
-rw-r--r-- 1 nelz users 7.3K Mar  4 11:22 Tutorial.txt
-rw-r--r-- 1 nelz users  68K Feb 10 11:49 Very_little_help.png
-rw-r--r-- 1 nelz users 3.2K Feb 28 10:57 Very_little_helps.tx
[nelz@hactar lxf/Answers 0]%
```

1 File permissions – this is a script as it has the execute bits set.

2 The user and group owning the file.

3 A directory usually has x set but also the special character d.

4 The time and date that the file was last modified.

5 Many distros add the --color=auto option, which helps distinguish between different types of file.

What goes where?

Users coming from Windows can be puzzled by the way Linux handles separate drives and partitions. Unlike the drive letter system used by Windows, Linux mounts everything in the same hierarchy. Your root partition, containing the core system files, is mounted at **/**, the root of the filesystem tree. Other partitions or drives can be mounted elsewhere at what are called mount points. For example, many distros use a separate partition for the home directory, where users' files are kept, to make installing a new version easier. This is a completely separate partition, it can even be on a different hard drive, but it appears at **/home** just as though it were part of the root partition. This makes everything easier and transparent for the user.

There is another difference. Linux, in common with every operating system but MS-DOS, uses a forward slash to separate directories. The layout of directories is also different, organising files according to their type and use. The main directories in a Linux filesystem are as follows...

/ The root of the filesystem, which contains the most critical components.

/bin and **/usr/bin** General commands.

/sbin and **/usr/sbin** System administration commands for the root user.

/etc Where system configuration files are kept.

/usr Where most of the operating system lives. This is not for

user files, although it was in the dim and distant past of Unix and the name has stuck.

/lib and **/usr/lib** The home of system libraries.

/var Where system programs store their data. Web servers keep their pages in **/var/www** and log files live in **/var/log**.

/home Where users' data is kept. Each user has a home directory, generally at **/home/username**.

Moving around

Now that we know where everything is, let's take a look at the common commands used to navigate the filesystem. Before going anywhere, it helps to know where we are, which is what **pwd** does. Many Unix commands are short, often two to three characters; in this case, **pwd** is print working directory – it tells you where you are. Many distros set up the terminal prompt to display the current directory, so you may not need this command often. Moving around is done with the **cd** (change directory) command. Run it with no arguments to return to your home directory. Otherwise it takes one argument, the directory to change to.

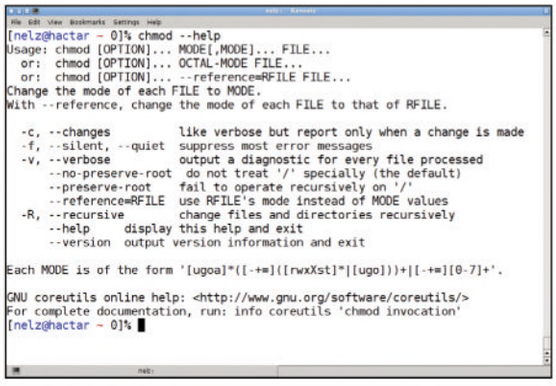
Directory paths can be either relative or absolute. An absolute path starts with **/** so **cd /usr/local** goes to the same place wherever you are starting from. A relative path starts at the current directory, so **cd Documents** goes to the Documents sub-directory of wherever you are, and gives an error if it is not there. That sounds less than useful if you can only descend into sub-directories, but there are a couple of special directory names you can use. To go up a directory use **cd ..** – a single dot is the current directory. There is also a shortcut for your home directory: **~**. Let's say you have directories called Photos and Music in your home directory and you are currently in Photos, either of these commands will move into Music:

```
cd ../Music
cd ~/Music
```

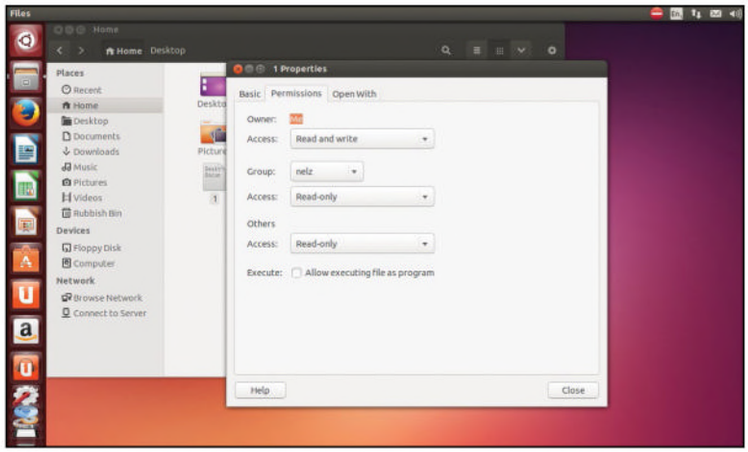
You can tell where you are with **pwd**, but how do you know what is in the current directory? With the **ls** command. Used on its own, it gives a list of files and directories in the current directory. Add a path and it lists the contents of that directory. If you want to know more about the files, use the **-l** (**--long**) option, which tells you the size and date of the file, along with information about ownership and permissions, which we will look at later.

With your permission

Every file object (that is files, directories and device nodes in **/dev**) has a set of permissions associated with it, as shown in the screenshot of the output from **ls -l**. These are normally in



If you need help with a command, ask the command for it. Most commands give a brief summary of their options when run with --help.



Here is the GUI way of changing file permissions. You would need to do this for each file you wanted to change, and click a separate box for each permission.

the form **rwxrwxrwx** and shown by **ls**, or the numeric equivalents. The three letters stand for read, write and execute, and are shown three times for the file's owner, the group it belongs to, and other users. For example, **rw-r--r--** is a common set of permissions for files; it means the owner of the file can read from or write to it, all other users can only read it. Program files usually appear as **rwxr-xr-x**, the same permissions as before but also all users can execute the file. If a program does not have execute permissions, you cannot run it. This is sometimes the case with system programs owned by the root user and only executable by root.

When applied to directories, the meanings are slightly different. Read means the same, but write refers to the ability to write into the directory, such as creating files. It also means that you can delete a file in a directory you have write permissions for, even if you don't have write permissions on the file – it is the directory you are modifying. You can't execute a directory, so that permission flag is re-purposed to allow you to access the contents of the directory, which is slightly different from read, which only allows you to list the contents (that is, read the directory).

File permissions are displayed by using the **-l** option with **ls** and modified with **chmod**, which can be used in a number of different ways, best shown by example:

```
chmod u+w somefile
chmod o-r somefile
chmod a+x somefile
chmod u=rw somefile
chmod u=rwx,go=rx somefile
chmod 755 somefile
```

The string following **chmod** has three parts: the targets, the operation and the permissions. So the first example adds write permission for the user. The next one removes read permission for other users, while the third adds execute permission for all users. **+** and **-** add and remove permissions to whatever was already set, while **=** sets the given permissions and removes the others, so the next example sets read and write for the file's owner and removes execute if it was previously set. The next command shows how we can combine several settings into one, setting read, write and execute for the owner, and read and execute for the group and others. The final command does exactly the same, but using the numerical settings. Each permission has a number: **4** is read, **2** is write, and **1** is execute. Add them together for each of the user types and you have a three-digit number that sets the permissions exactly (there is no equivalent to **+** or **-** with this method).

Terminal: Apt-

New to Linux? Then allow us to guide you through your first steps with **apt-get**, the powerful command line tool.

One of the biggest changes that catches Windows users moving to Linux is the way that software is installed. Instead of downloading an executable file from some website or other, running it and hoping it doesn't clobber your existing library files (DLLs) or install some dubious adware or malware, Linux distributions maintain repositories of software, which are all packaged up for that distro and tested for compatibility with the rest of the distro.

In this tutorial, we will look at how this is done by distros that use the *Advanced Packaging Tool* (apt) software management system, as developed by Debian and used by distros from Ubuntu to Raspbian on the Raspberry Pi.

Repositories

A repository is a collection of software packages for a distro. Each major release of a distro will have its own repositories, and the packages will have been built for and tested with that release, but a repository is more than a collection of files. Each repo (as they are usually called) is indexed, making it easy to find what you want. It can also be quickly checked for updates for your package manager without any need to visit websites to check for updates, or the need for software to 'phone home' to check.

More importantly, each package in a repo is signed with the repository's GPG (encryption) key, which is checked when installing packages. This means you can trust the software installed from there to be what it says it is, and not some infected trojan that's been uploaded maliciously.

A repository also makes dependency handling simple. A dependency is a program that the program you want to install needs to run, such as a library. Instead of bundling everything in the package and ending up with multiple copies of the same library on your computer (which is what Windows does), a package simply lists its dependencies so that your package manager can check whether they are already installed, and grab them from the repo if not.

In addition to the default repositories provided by the distro, there are several third-party ones that can be added to your package manager. These are not guaranteed to be tested to the same standards as the official repos, but many of them are very good, and if you stick to the popularly recommended repos for your distro, you won't go far wrong.

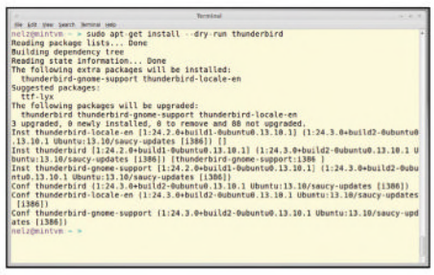
Ubuntu has also introduced the concept of the PPA, or Personal Package Archive, which are small repositories for individual projects. These may each be added individually to your package manager, but be careful about adding any untrusted sources.

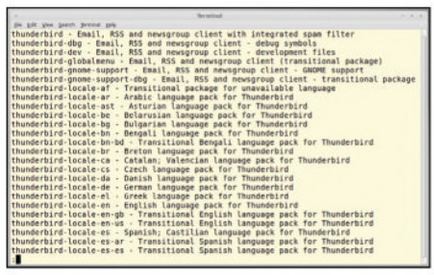
Package management

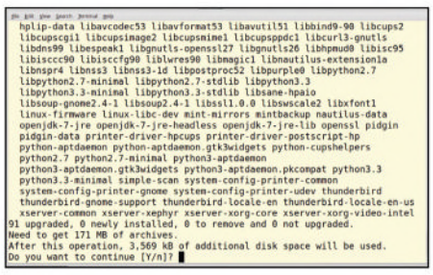
We have used the term 'package manager' a few times now but what is it? Basically, this is a program that enables you to install, update and remove software, including taking care of dependencies. It also enables you to search for programs of interest, as well as performing other functions. All distros will have command line package management tools. You can access them either by using your system's search and looking for **terminal** or using [Ctrl]+[Alt]+[T] in desktops such as Unity, Gnome or Xfce, even if they also provide a fancy graphical front end. The main commands are:

- **apt-get** Installs, upgrades and uninstalls packages.
- **apt-cache** This works with the repository index files, such as searching for packages.

Package management







1 Install

Using **apt-get install** will check the dependencies of the packages you want and install any that are needed. Adding **--dry-run** to **apt-get install** enables you to see what would be done, without actually writing anything to your hard drive. If you are happy, run the command again without **--dry-run**.

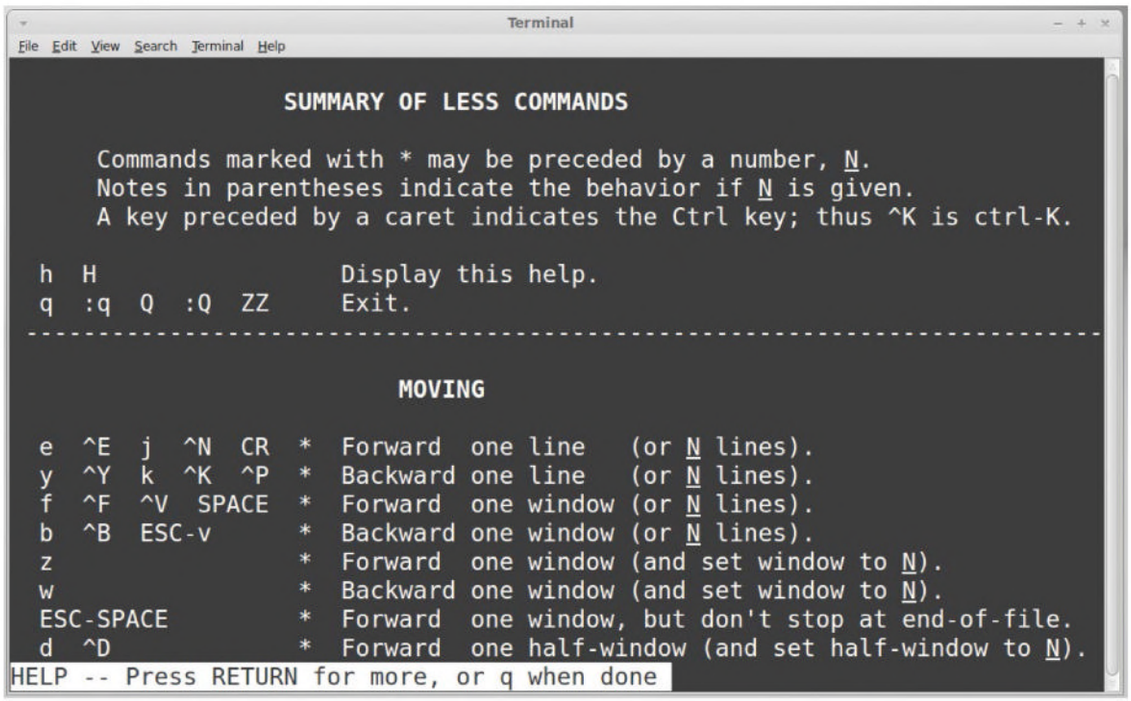
2 Search

Use **apt-cache search** to find what's available. The **--names-only** option can give a more manageable set of results if you know the program's name. Otherwise let **apt-cache search** go through the descriptions, too, and view the results in **less**. You don't need to use **sudo** as **search** doesn't write to your drive.

3 Update

Run **apt-get update** to update all your package lists, followed by **apt-get upgrade** to update all your installed software to the latest versions. In our case, it's well overdue. Then **apt** will show you what needs to be updated, and how much needs to be downloaded, before asking whether you want to proceed.

get in action



Less displays text from any source – from a file, the output of another program or its built-in help if you manage to get stuck.

» **add-apt-repository** Adds extra repositories to the system.
» **dpkg** A lower level package manipulation command.

These commands generally require root (superuser) access, so should be run at the root user or with **sudo** – we will stick with the **sudo** approach here. We've already mentioned that repos are indexed, so the first thing to do is update your index files to match the current contents of the repositories with:

```
sudo apt-get update
```

Then you probably want to make sure that your system is up to date:

```
sudo apt-get upgrade
```

This will list the packages it wants to install, tell you how much space it needs for the download, and then get on with it when you tell it to. When you want to install some new software, unless you have been told the exact name to install, you may want to search for it first, like this:

```
apt-cache search gimp
```

This will spit out a long list of packages, because it searches both name and description, and lists anything mentioning gimp, and there are a lot of them. To search only the names, use the **-n** or **--names-only** option:

```
apt-cache search -n gimp
```

This often gives a more manageable output, but still a lot in this case, perhaps too much to fit in your terminal window. The solution to this is to pipe the output from this command to the program **less**:

```
apt-cache search -n gimp | less
```

The **less** command is a pager – it lets you read text page by page and scroll through it. It can be used with any program that generates lots of terminal output to make it easier to read (see the 'Package management' walkthrough opposite for more details). Once you have found the package you want, installation is as simple as:

```
sudo apt-get install gimp
```

You can install multiple programs by giving them all to **apt-get** at once:

```
sudo apt-get install program1 program2...
```

Not every program you try will be what you want, so you can tidy up your hard drive by uninstalling it with:

```
sudo apt-get remove program1
```

Or you can use:

```
sudo apt-get purge program1
```

Both commands remove the program, but **remove** leaves its configuration files in place while **purge** deletes those, too.

There are a number of extra options you can use with **apt-get**, the **man** page lists them all (type **man apt-get** in the terminal), but one of the most useful is **--dry-run**. This has **apt-get** show you what it would do without actually doing it, a useful chance to check that you are giving the right command. Remember, computers do what you tell them to, not what you want them to do! Finally, you don't normally need to use **dpkg**, but it is useful for listing everything you have installed with **dpkg -L**.

Terminal: Core programs

Out of the hundreds of different terminal commands available, here's a handy summary of some of the more useful ones.

We've looked at various shell commands in the last few tutorials, but they have each been in the context of performing a particular task. It's time to take an overview of some of the general-purpose commands. There are thousands of terminal commands, from the commonplace to the arcane, but you need only a handful of key commands to get started. Here we will look at some of the core workhorse commands, giving a brief description of what each one is for. As always, the **man** pages give far more detail on how to use them. Many of these produce more output than can fit in your terminal display, so consider piping them through **less**.

Central to any terminal activity is working with files, creating, removing, listing and otherwise examining them. Here are the main commands for this.

» **ls** Lists the contents of the current or given directory.

» **ls -l** As for **ls**, but gives more information about each item. Add human-readable file sizes with **-h**:

```
ls -lh MyPhotos
```

» **rm** Deletes a file. Use the **-i** option for confirmation before each removal or **-f** to blitz everything. With **-r** it deletes directories and their contents, too.

» **rmdir** Deletes a directory, which must be empty.

» **df** Shows free disk space on all filesystems or just those given on the command line.

» **du** Shows the amount of space used by individual files or directories.

```
df -h /home
du -sh /home/user/*
```

» **file** Identifies the type of a file. Unlike Windows, which uses the file name extension, this command looks inside the file to see what it really contains.

» **find** Searches the current or given directory for files matching certain criteria. For example, you could find all *LibreOffice* spreadsheets with:

```
find Documents -name '*.ods'
```

» **locate** This also looks for files but using a much faster system. The **locate** database is automatically rebuilt each day by *updatedb*, and **locate** then searches this. It's fast, but doesn't know about very recent changes.

Text handling

Text files are all around us, from emails to configuration files, and there are plenty of commands that deal with them. If you want to edit a text file, for example, there are a number of choices, with the two big ones being **Emacs** and **Vi**. Both are overkill if you just want to tweak a configuration file; in this instance, try **nano** instead:

```
nano -w somefile.txt
```

The **-w** option turns off word wrapping, which you certainly don't want when editing configuration files. The status bar at the bottom shows the main commands – for example, press [Ctrl]+[X] to save your file and exit.

This assumes you know which file you want, but what if you know what you're looking for but not the name of the file? In that case, use **grep**. This searches text files for a string or regular expression.

```
grep sometext *.txt
```

will search all .txt files in the current directory and show any lines containing the matching text from each file, along with the name of the file. You can even search an entire directory hierarchy with **-r** (or **--recursive**):

```
grep -r -I sometext somedir
```

Be careful when you are searching large directory trees,

Getting help

The command line may appear a little unfriendly, but there's plenty of help if you know where to look. Most commands have a **--help** option that tells you what the options are. The **man** and **info** pages are the main sources of information about anything. To learn all the options for a program and what they do, run:

```
man progname
```

The **man** pages are divided into numbered sections. The ones that are most applicable to using the system are:

» **1** User commands

» **5** File formats and conventions

» **8** System administration tools

If you don't specify the number, **man** will pick the first available, which usually works. But **man** pages are not limited to programs; they also cover configuration files. As an example, passwords are managed by the **passwd** command, and information is stored in the **/etc/passwd** file, so you could use:

```
man passwd
man 1 passwd
man 5 passwd
```

The first two would tell you about the **passwd** command, while the third would explain the content and format of the **/etc/passwd** file.

Man pages have all the information on a single page but **info** pages are a collection of hypertext-linked pages contained in a single file. They often provide more detail but aren't very intuitive to read – try **info info** to see how to use them. It's often easier to use a search engine to find the online version of **info** pages, which contain the same information in the more familiar HTML format.

Commands, options and arguments

You'll often see references to command arguments and options, but what are they? Options and arguments are the things that tell a program what to do. Put simply, arguments tell a command what to do, while options tell it how to do it – although the lines can get a little blurred. Take the **ls** command as an example – this lists the contents of a directory. With no options or arguments, it lists the current directory using the standard format:

```
ls
Desktop  Downloads  Music  Public  Videos
Documents  examples.desktop  Pictures  Templates
```

If you want to list a different directory, give that as an argument:

```
ls Pictures
```

or

```
ls Desktop Downloads
```

Arguments are given as just the names you want listed, but options are marked as such by starting with a dash. The standard convention among GNU programs, and used by most others, it to have long and short options. A short option is a single dash and one letter, such as **ls -l**, which tells **ls** to list in its long format, giving more detail about each file. The long options are two dashes followed by a word, such as **ls --reverse**, which lists entries in reverse order, as is pretty apparent from the name. **ls -r** does the same thing but it is not so obvious what it does. Many options, like this one, have long and short versions, but there are only 26 letters in the alphabet, so less popular options are often available only in the long version. The short options are easier to type but the long ones are more understandable. Compare

```
ls -l --reverse --time
```

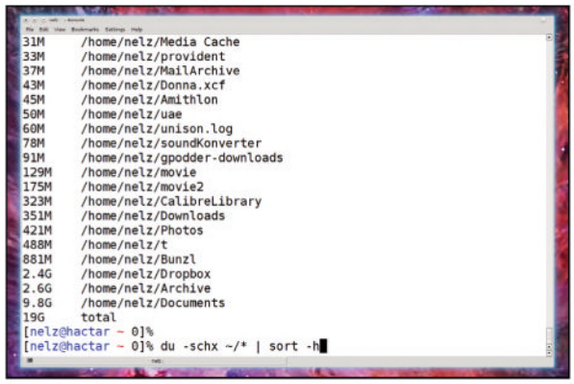
with

```
ls -l -r -t
```

or even

```
ls -lrt
```

Each gives a long listing in reverse time/date order. Notice how multiple short options can be combined with a single dash. While we're talking about **ls**, this is a good time to mention so-called hidden files. In Linux, any files or directories beginning with a dot are considered hidden and do not show up in the output from **ls** or in most file managers by default. These are usually configuration files that would only clutter up your display – but if you want to see them, simply add the **-A** option to **ls**.



By piping the output from du through sort, and adding extra options to both commands, we can see which directories use the most space.

because it can be slow and return strange results from any non-text files it searches. The **-I** option tells **grep** to skip such binary files.

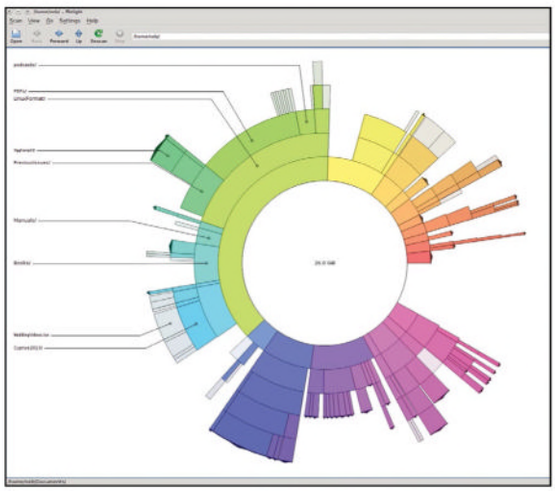
Text is also the preferred way of passing data between many programs, using the pipes we looked at previously. Sometimes you want to pass data straight from one program to the next, but other times you want to modify it first. You could send the text to a file, edit it and then send the new file to the next program, or you could pass it though a pipe and modify it on-the-fly. **Nano** edits files interactively, **grep** searches them automatically, so we just need a program to edit automatically; it's called **sed** (Stream EDitor). **Sed** takes a stream of text, from a file or pipe, and makes the changes you tell it to. The most common uses are deletion and substitution. Normally, **sed** sends its output to **stdout**, but the **-i** option modifies files in place:

```
sed -i 's/oldtext/newtext/g somefile.txt
sed -i '/oldtext/d' somefile.txt
```

The second example deletes all lines containing **oldtext**. Another useful program is **awk**, which can be used to print specific items from a text file or stream.

```
awk '{print $1}' somefile.txt
cat *.txt | awk '/^Hello/ {print $2}'
```

The first example prints the first word from each line of the file. The second takes the contents of all files ending in .txt, filters the lines starting with **Hello** (the string between the slashes is a pattern to match) and then prints the second word from each matching line.



KDE's Filelight, shown here, and Gnome's GDU provide a graphical alternative to du, representing disk usage visually.

Networking

We normally think of big graphical programs like *Chromium* and *Thunderbird* when we think of networked software, but there are many command line programs for setting up, testing and using your network or internet connection.

Ping Sends a small packet to a remote server and times the response, which is useful if you want to check whether a site is available, or if your network is working.

```
ping -c 5 www.google.com
```

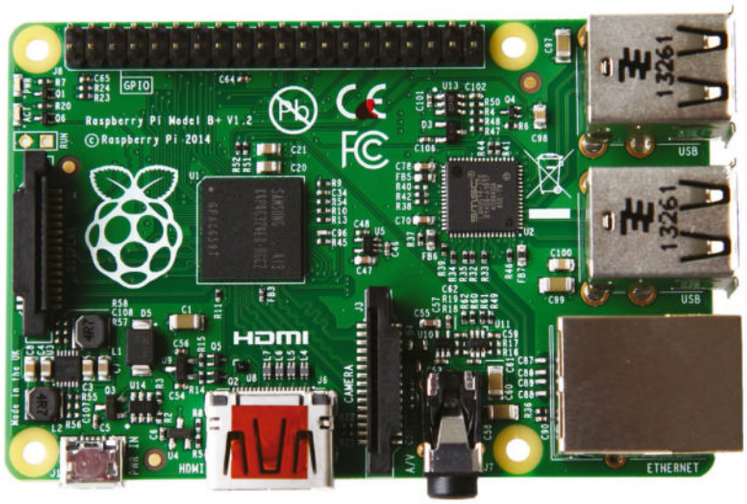
wget Downloads files. The only argument it needs is a URL, although it has a huge range of options that you will not normally need.

hostname Shows your computer's host name, or its IP address with **-i**.

lynx A text mode web browser. While not as intuitive as *Chromium* or *Firefox*, it is worth knowing about in case you ever suffer graphics problems.

Packages: How

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



If you're used to Windows, you may be used to each bit of software having its own installer, which gets the appropriate files and puts them in the appropriate places. Linux doesn't work in the same way (at least, it doesn't usually). Instead, there is a part of the operating system called the package manager. This is responsible for getting and managing any software you need. It links to a repository of software so it can download all the files for you. Since Linux is built on open source software, almost everything you will need is in the repositories and free. You don't need to worry about finding the install files, or anything like that – the package manager does it all for you.

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

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

apt-get is a command line program, so to start with you'll need to open a terminal (see the command line interface section on page 34 for more information on this). Since package management affects the whole system, all the commands need to be run with **sudo**. The first thing you need to do is make sure you have the latest list of software available to you. Run:

```
sudo apt-get update
```

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

```
sudo apt-get upgrade
```

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

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

```
apt-cache search iceweasel
```

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

To install *Iceweasel*, run:

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

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

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

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

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

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

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

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

```
sudo apt-get synaptic
```

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

```
--where is synaptic
```

The boxout on the next page shows you what to expect.

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

do they work?

Further information

Synaptic

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

Raspberry Pi store

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

Compiling software

Sometimes, you'll find you need software that isn't in the repository, and so you can't get it using **apt-get**. In this case, you'll need to compile it. Different projects package their source code in different ways, but usually, the following will work. Get the source code from the project's website, and unzip it. Usually, the filename will end in .tar.gz or .tgz. If this is the case, you can unzip it with:

```
tar zxvf <filename>
```

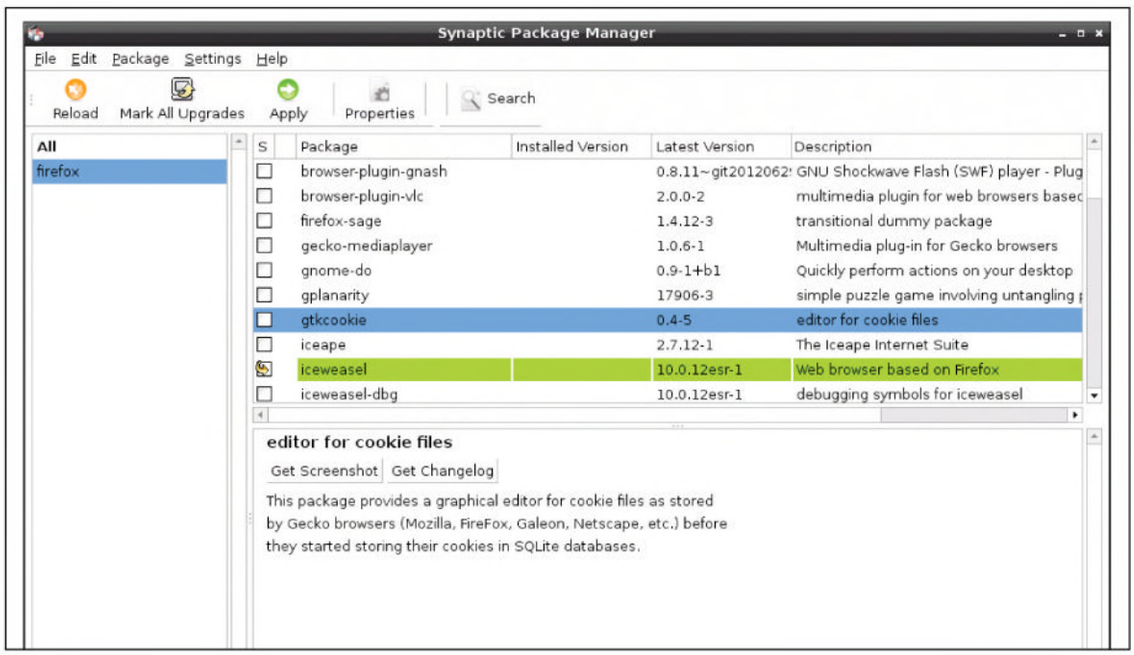
If the filename ends in .tar.bzip2, you need to replace **zxvf** with **xjf**. This should now create a new directory which you need to **cd** into. Hopefully, there'll be a file called **INSTALL**, which you can read with **less INSTALL** This should tell you how to continue with the installation. Usually (but not always), it will say:

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

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

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

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



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

Man pages: Accessing help

Need more advice? Then you need to browse the ultimate collection of useful self-help books that resides inside the Linux operating system.

RTFM has long been considered the battle cry of the supposed Linux experts, and is claimed to scare new users away. If you haven't come across it before, it stands for something like 'Read the fine manual'. It is perhaps easy to appreciate the frustration some individuals feel when asked a question for the umpteenth time when the information is clearly covered in the manual. However, it's only possible for a user to read a program's documentation if they know where to find it. Happily, there are some sources of help within Linux, so let's look at each of them.

Before you even look for the manual, remember that many programs have built-in help. Run the command from a terminal with the **--help** option to see a synopsis of its options. This applies to GUI programs, as well as shell commands. For example:

```
firefox --help
```

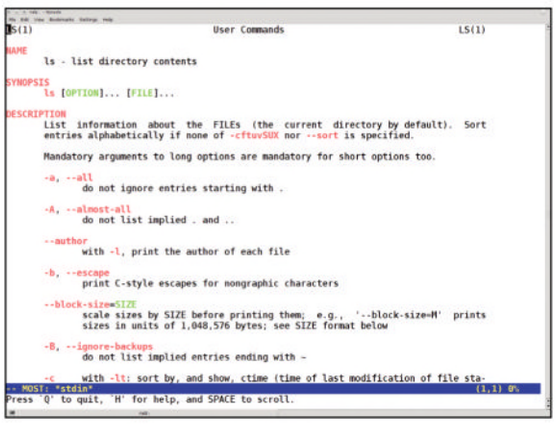
If you need more detail, then it may be time to RTFM, which on Linux systems usually means the man page. Man pages document just about everything on your system and are viewed by keying in the **man** command. If you want to know how **man** itself works, the classic recursive example is to open a terminal and run:

```
man man
```

A man page is a single page containing the reference documentation for its topic. The **man** command renders this document as readable text and displays it in your default pager, usually *less*. This means you use the same keyboard controls as *less* for navigating the page: cursor keys to scroll up and down, [Spacebar] to page down, and so on. Some man pages can be very long, so try **man bash** to enable you to search. In *less*, press [/] to start searching (or [?] if you want to search backwards), followed by your search term. Then use [n] to jump to the next match or [N] for the previous one. The man pages are divided into sections:

1. User commands
2. System calls
3. C library functions
4. Devices and special files
5. File formats and conventions
6. Games *et al*
7. Miscellany
8. System administration tools and daemons

As a normal user, you would normally only use sections 1, 5 and 8 (and possibly 6). If you use the section number with

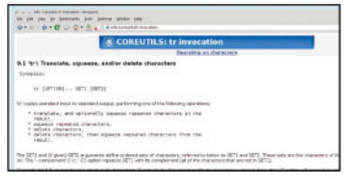


This is the man page for ls, and it helpfully lists the various options that you can use in alphabetical order.

Desktop viewing

Man and **info** are intended to be readable in a terminal, because you may need to use them on a system without a desktop. There are GUI viewers for them, though, the most convenient being in KDE, where you can press [Alt]+[F2] and type **man:/command** or **info:/command** and get an HTML formatted version of the document displayed in *Konqueror*. There are also the *tkInfo* and *tkMan* programs to display the respective formats in a GUI window.

There are several websites containing comprehensive man page collections: such as **http://linux.die.net**, **www.linuxmanpages.com** and **http://manpages.ubuntu.com**. These are particularly useful if you want to read about something that you do not have installed locally.



KDE users can read info and man pages in a browser, with clickable links, thanks to KDE's KIO slaves.

the **man** command, it will only look in that section, otherwise it will show the first match it finds. This is necessary because you can have more than one page with the same name. The **passwd** command is used to set user passwords, which are stored in the file **/etc/passwd**. Try:

```
man passwd
man 1 passwd
man 5 passwd
```

The first two document the **passwd** command from section 1, while the third shows the man page for the **passwd** file. This is one of the strengths of the man page system – it documents everything: commands, configuration files, library functions and more. It's not limited to specific commands or files, and section 7 contains man pages for all sorts of things. Ever wondered how symbolic links work, or what happens when you turn on your computer? Try:

```
man 7 symlink
man 7 boot
```

Quick help

There is more to **man** than a bunch of formatted text pages and a program to display them in a pager. **Man** maintains a searchable database of pages, which is automatically updated by *Cron*, and has some other programs for working with it. Each man page has a NAME section, which includes a short description of the page's topic. The **whatis** command gives the description – it tells you what the program (or file) is, without going into details of options. Here is the classic geeky, recursive example:

```
whatis whatis
whatis           (1)  - search the whatis database for
complete words
```

This is a faster way to see what commands do, especially if you want to check more than one:

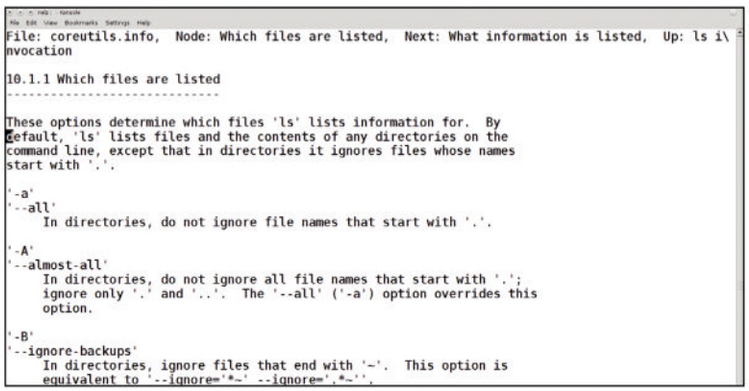
```
whatis grep sed symlink
```

The **whatis** command searches only the name, and only matches on whole words, so it assumes you know the name of the command and just want to know what it does. For a more wide-ranging search, use **apropos**, which does a similar job but searches the descriptions as well and returns all matches – compare these two commands:

```
whatis png
apropos png
```

There is another form of documentation that's favoured by the GNU project: info pages. While a man page is basically one very long text file with a bit of formatting markup, an info document contains a tree of linked pages in a single file. It's more like HTML than plain text, but is designed for reading in a text console and all the 'pages' are contained in a single file. Unsurprisingly, the command used to read info pages is this:

```
info info
```



The ls info page goes into more detail and groups options according to their function. Info pages generally provide more detail than man pages.

This time, the self-referencing is not gratuitous. Man pages are usually quite intuitive to navigate – it's just like reading any other text in a pager. **Info** uses a different set of key commands, so reading its own info page is a good place to start. You move around individual pages as you would normally, but if you press [Enter] when the cursor is on a link (noted by an asterisk), you descend into that node, getting further information on the topic at hand. To go back up to the parent node, press [u]. You can also navigate within a level of the documentation tree with [n] and [p], which take you to the next and previous nodes. If you have ever looked at any GNU documentation online (**http://bit.ly/grubmanual**) you will recognise this layout. It's simpler than HTML, with the navigation commands generally moving around the same level of the tree or up and down one level.

You may be asking yourself what the point of this structure is. Well, if you've ever tried to find information in a long man page, such as **man bash** or **man mplayer**, you'll know how frustrating and time-consuming the 'everything on one page' approach can be. Info documents are divided into sections and chapters, enabling clearer and more succinct presentation. The majority of GNU programs have fairly brief man pages, but have more detail in their info pages. Splitting the document up into pages alters the way that searching is carried out. Press [s] followed by a search string and then [Enter]. Info will jump to the next occurrence of the string, even if it's in a different node. Continue to press [s] then [Enter], with no search string, to jump to subsequent occurrences of the same string. Those keys, plus [q] to quit, should enable you to navigate info pages with ease. You might be wondering why we're not using HTML. The main reasons are that **info** pre-dates HTML, and that info documents are contained within a single file. Conceptually, they are similar – so much so that **info2html** (**http://info2html.sourceforge.net**) can be used to convert an info document to a series of HTML pages.

Printing manuals

There may be times you want a hard copy. As **man** pages are stored in a markup format and converted for display by the **man** program, you can use **-t** to convert them to a printable format, Postscript by default, like this:

```
man -t somecommand | lpr
```

The Postscript is output to **stdout**; piping it to **lpr** like this sends it straight to the printer. You could also convert the Postscript to PDF, and therefore create versions of your man pages suitable for carrying around on a tablet or e-reader:

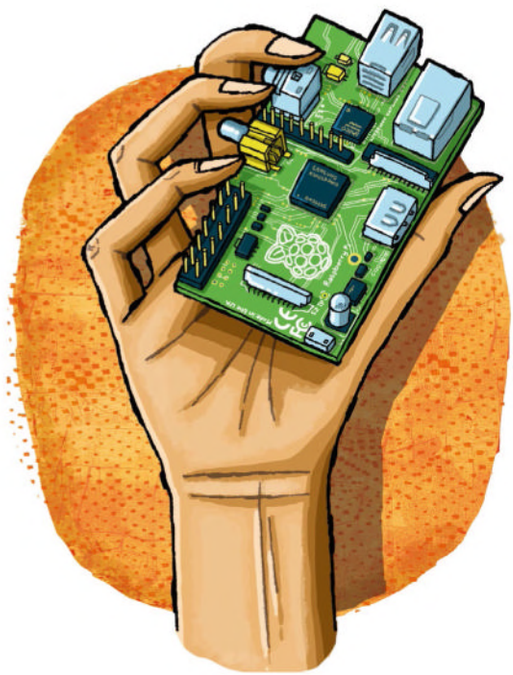
```
man -t somecommand | ps2pdf - somecommand.pdf
```

Print info documents by passing them through the **col** command:

```
info somecommand | col -b | lpr
```

Hardcore: Build a distro

You don't have to wait for your Raspberry Pi to boot before tinkering with it – you can create a customised distribution containing just what you want.



In this tutorial, we will explore how to create a customised system image for your Raspberry Pi that contains what you want. You may wish for different packages than those included in the official images, you may wish to customise the configuration, perhaps even customise the kernel. You'll be able to write the image to an SD card to boot your Pi, or you could use an emulator to boot the image on your PC. You can customise an image right on the Pi, but it will be very slow, so we'll also explore ways of using your PC to cross compile or even execute ARM code.

You could, by all means, hack a running system by adding and removing packages but this is risky, because one mistake can stop things working. Also, doing it that way isn't very easy to reproduce, and doesn't provide a suitable candidate for automation.

The first thing we need to do is to fill our tool box. For this tutorial, we'll be using Arch Linux, because this is a great distro that is available for both the PC and the Pi. We'll assume you're running the most recent official Arch Linux image on your Raspberry Pi and that it's up to date. Boot Arch Linux on your Pi and ensure the necessary build tools and other packages are installed:

```
# pacman -Syu
# pacman -S base-devel python2 git parted dosfstools
```

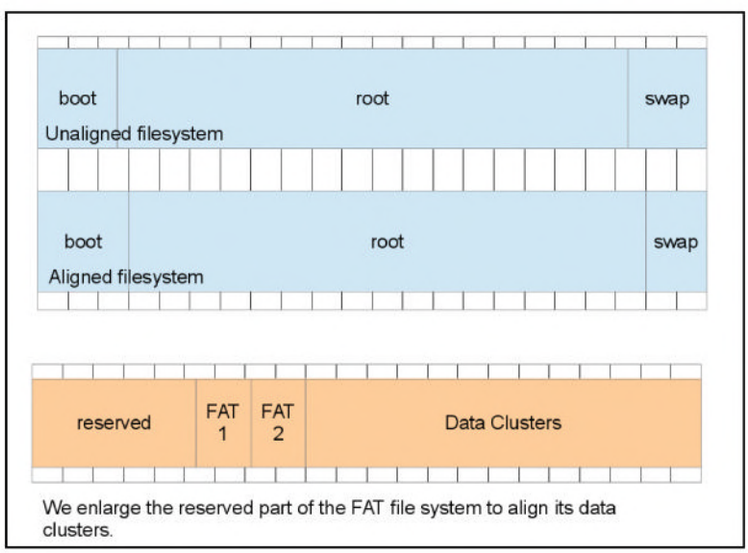
Choose your own adventure

The first step towards having a customised image is to decide what it should contain. A reasonable selection to start with is Arch Linux's base package group, adding and removing packages as needed. Unwanted kernel packages should be removed and the Raspberry Pi ones added.

It's necessary to build a custom kernel if you need to add features. It can also be useful to disable unwanted kernel features to reduce its size (given the limited memory resources of a computer like the Pi). Given that compiling a kernel natively takes in excess of 10 hours, an alternative method is essential. Using a cross compiler is one way to do that; you can also use **distcc**, but you still need a cross compiler with distcc because the **distcc** 'server' has to generate ARM code. We decided to cross-compile the kernel, so both techniques are covered by the tutorial.

The commands below use *Pacman*, the Arch Linux package manager to get the list of packages in the base group. The unwanted packages are then removed and the Raspberry Pi kernel and firmware ones are added. You can also use your favourite text editor to add or remove anything else according to your own requirements (you might want to add openssh).

```
$ pacman -Sg base | awk '{print $2}' | grep -v "^\
(linux\|kernel\)" | tr '\n' ' ' > base_packages
```



Aligning with the erase blocks makes for a happy SD card.

```
$ sed -i -e 's/$/linux-raspberrypi linux-headers-raspberrypi raspberrypi-firmware/' base_packages
```

With the package list decided upon, it's time to install the packages for the new image. The **mkarchroot** utility makes this easy. It installs packages into a subdirectory that you can then convert into a disc image.

```
# mkarchroot archroot $(cat base_packages)
```

This creates a new subdirectory called **archroot** that contains everything needed for a complete system. You could **chroot** into this and it would work; in fact, if you wanted to make any changes, for example to set a root password or add user accounts, this is what you would do.

We will create a system image in a file. You could also create it directly on an SD card, but having an image is useful because it can be rewritten to a card quickly or booted in an emulator without ever writing it to a card. The kernel has a loop facility that allows a file to be treated like a physical disk that can be partitioned into filesystems. It creates a regular device node in **/dev** that is used to access the file in this way.

The image needs to be big enough for the filesystems but small enough to fit on an SD card. We'll create a 2GiB image, the same size as the official image. Ensure the loop kernel module is loaded, create the file and a loop device for it:

```
# modprobe loop
$ truncate -s 2G myimage
$ device=$(losetup -f)
# losetup $device myimage
```

We use **losetup** twice – to allocate a device name, usually **/dev/loop0**, for the loop device, and to create it (we store the device name in a variable so we can refer to it later). The image needs to have a specific partition layout; its first partition must be FAT16 and it must contain the boot files and kernel image. A second partition is required for the root filesystem, and this can be ext4. If you want a swap partition, a third one can be created for that. The Raspberry Pi's boot firmware looks for an MS-DOS-style MBR partition table (unlike the PC BIOS, it doesn't execute a bootloader from the MBR). Use **parted** to create a partition table on the image file:

```
# parted -s $device mktable msdos
```

When creating partitions, it is wise to align them with the erase blocks of the target SD card. Doing this is optional but recommended. Most cards have a preferred erase size of 4MiB, but you can check the value (in bytes) for a card:

```
$ cat /sys/class/block/mmcblk0/device/preferred_erase_size
```

For a 4MiB preferred erase size, alignment occurs every 8,192 sectors (a sector contains 512 bytes, so 4MiB/512). Partitions must therefore start on a sector that is a multiple of 8,192; the first erase block contains the first 8,192 sectors (0..8191) and the second erase block contains the next 8,192 sectors (8192..16383), and so on. Sector 0 is reserved for the partition table, so the first partition needs to start at the next aligned sector, which is sector 8192. A good size for the boot partition is 40MiB – 81,920 sectors. This is big enough to contain the boot files and ends on an erase block boundary. Create the boot partition starting at sector 8192 and ending on sector 90111 (8192+81920-1):

```
# parted -s $device unit s mkpart primary fat32 8192 90111
```

Reserve some space for a swap partition if you want one (writing swap on an SD card is questionable due to the slow speed and limited write ability of the medium). As an example, we'll create space for a 256MiB swap partition (this is 524,288 sectors, or 64 write blocks).

Our 2GiB image contains 4,194,304 sectors (2GiB/512 = 4,194,304). Recalling that sector numbering starts at 0, that makes the last sector 4194303. The root partition can use all the space between the boot and swap partitions, starting at aligned sector 90112 and ending at an alignment boundary, leaving 256MiB for the swap partition. Create them:

```
# parted $device unit s mkpart primary ext2 90112 3670015
# parted $device unit s mkpart primary linux-swap 3670016 4194303
```

Should you wish, you can print out the partition table with **parted -s $device unit s print**. Next, create the filesystems. The loop device needs to be recreated so that device nodes are created for the partitions (the **-P** option does this):

```
# losetup -d $device
# device=$(losetup -f)
# losetup -P $device myimage
```

Consider aligning the filesystems' internal structure. Aligning a FAT filesystem requires knowledge of its FAT size. You have to make a filesystem to find this out:

```
mkfs.vfat -I -F 16 -n boot -s 16 -v ${device}p1 | grep "FAT size"
```

The structure of a FAT filesystem is shown in the diagram. To achieve alignment, adjust the reserved space so that the beginning of the data area is aligned. This size of the reserved space needs to equal the preferred erase size, less the size of the two file allocation tables. Continuing our example with a FAT size of 32 sectors, the reserved space would be 8,192-(2*32)=8,128 sectors. Make the filesystem again using this information:

```
mkfs.vfat -I -F 16 -n boot -s 16 -R 8128 -v ${device}p1
```

For the root partition, use an ext4 filesystem. This aligns with the erase size due to its 4KiB block size. Create it without journalling, as this reduces writes to the SD card:

Almost like the real thing – *QEMU* emulates the Pi using our custom image.

Quick tip

If you're running a 32-bit system, there is a pre-built cross compiler called arm-bcm2708-linux-gnueabi included in the Raspberry Pi Tools. See **http://github.com/raspberrypi/tools**

Quick tip

Make the most of your multi-core CPU: **make**'s **-j** parameter tells it to start multiple concurrent jobs. For a Core-i7 with its four cores (eight threads), use a value of 8.

Work in your image with chroot

Sometimes it's useful to work inside a filesystem image, such as the **archroot** that we created in this tutorial. You can use **chroot** to do this, but it helps to bind-mount some parts of the filesystem first. Here is a little script you can use to enter a **chroot**:

```
#!/bin/bash
mkdir -p $1/{dev/pts,proc}
mount proc -t proc $1/proc
mount devpts -t devpts $1/dev/pts
chroot $1 /usr/bin/env -i
TERM="$TERM" /bin/bash --login
umount $1/{dev/pts,proc}
```

```
$ mkfs.ext4 -O ^has_journal -L root  ${device}p2
```

Mount the filesystems and copy the files from the **archroot** subdirectory into place. The boot partition is mounted at **/boot** on the root partition so that the boot files get placed on the correct partition:

```
# mount ${device}p2 /mnt
# mkdir /mnt/boot
# mount ${device}p1 /mnt/boot
# (cd archroot ; cp -a * /mnt)
# umount /mnt/boot /mnt
```

Now, write the image to the SD card (if your Pi has its root filesystem on the SD card, you'll need to copy the image to another machine with a card reader):

```
# dd if=myimage of=/dev/mmcblk0 bs=4M
```

Shut down your Pi, pop in the new card and boot up. Alternatively, copy your image on to your PC and use an emulator to run it there…

The **linux-raspberry** package that we included in our image contains the official kernel image, but it is quite straightforward to customise it. You need the kernel source code and a compiler to convert that into the executable kernel image. A compiler normally produces executable code for the same CPU but, by using a cross compiler, it's possible to produce executables for different CPUs. With a cross compiler, we can take advantage of our more powerful x86 hardware to compile for the Raspberry Pi's ARM hardware much quicker than we could do it directly on the Pi (compiling the 3.2.27 kernel on the Pi takes hours; with a cross compiler, an Intel i7 can do it in two and a half minutes). So leave your Pi to one side for a moment and get ready to compile on your PC. The native compiler is called **gcc** and the cross compiler version for the ARM architecture is called **arm-linux-gnueabi-gcc**. It isn't in the repositories, but building it is simple if you use **yaourt** (yet another user repository tool):

```
yaourt -S arm-linux-gnueabi-gcc
```

You'll need stay close to your terminal so you can answer **yaourt**'s many prompts (choose 'no' when asked if you want to edit the PKGBUILD and 'yes' when asked if you want to build or install a package. It builds in stages, so also choose 'yes' to replace conflicting packages).

Hello world!

Before going further, it's probably worth checking your cross compiler. In true programmer fashion, we'll do a quick 'hello, world'. Create a new file called **hello.c**:

```
#include <stdio.h>
int main ()
{
  printf("Hello, World!\n");
  return 0;
}
```

Now compile it for ARM and check its type is correct:

```
$ arm-linux-gnueabi-gcc -o hello hello.c
$ file hello
hello: ELF 32-bit LSB executable, ARM, version 1 (SYSV), dynamically linked (uses shared libs), for GNU/Linux 2.6.27, not stripped
```

If you copy the 'hello' file over to your Pi, you should be able to execute it:

```
$ ./hello
Hello, World!
```

With that success, we have a cross compiler, and we can get on with compiling our custom kernel.

```
$ git clone --depth 1 git://github.com/raspberrypi/linux.git
$ cd linux
$ ssh root@alarmpi zcat /proc/config.gz > .config
$ make -j 8 ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- menuconfig -k
```

We download the kernel source from github. Using **--depth 1** just gets that latest version and not the entire history; it's much quicker this way. We then copy the Raspberry Pi's current kernel configuration into a file called **.config** and open that file in the kernel menuconfig editor. This is where you can customise the kernel configuration to meet any specific requirements that you may have. Exit menuconfig when done, opting to save the updated **.config**. You're now ready to build the kernel and its modules (which we install to a subdirectory):

```
$ make -j 8 ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- -k
$ mkdir -p modules
$ make -j 8 -k ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- modules_install INSTALL_MOD_PATH=modules
```

When the build finishes, you could upload the new kernel image and modules to the Pi and then reboot it (you may want to back up the old kernel and modules directories first):

```
$ scp arch/arm/boot/Image root@alarmpi:/boot/kernel.img
$ cd modules/lib/modules
$ tar cJf - * | ssh root@alarmpi '(cd /usr/lib/modules; tar xjf -)'
$ ssh root@alarmpi reboot
```

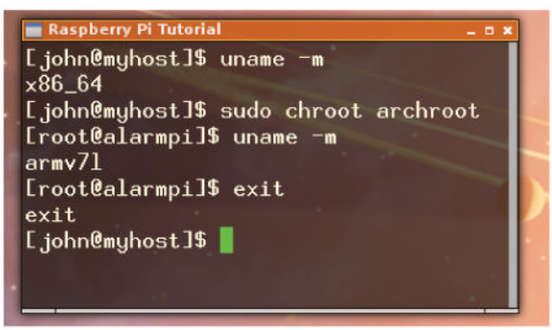
Alternatively, you can install it into your **archroot** tree prior to creating an image. Copy **arch/arm/boot/Image** to »

Quick tip

If you don't have **Yaourt**, its source code is in the Arch User Repository (AUR). You can avoid the hassle of building it yourself if you use the package in the unofficial [archlinuxfr] repository. See **http://archlinux.fr/yaourt-en.**

Quick tip

More Raspberry Pi goodies can be found at **https://github.com/johnlane/rpi-utils**



Split Personality: x86 turns into ARM!

Raspberry Pi boot sequence

The Raspberry Pi does not boot the same as a PC. It is the GPU on the Pi that starts the process (not the ARM chip, as you may assume). After power-on, the GPU runs its first-stage bootloader (ROM firmware). This loads further stages from the first (FAT16 formatted) partition on the SD card. The second-stage bootloader, called **bootcode.bin**, is loaded and executed in the GPU's L2 cache.

This, in turn, loads a third-stage bootloader, called **loader.bin** in RAM, which reads the GPU firmware, called **start.elf** Additional files **config.txt** and **cmdline.txt** allow configuration of this boot process. Finally, **start.elf** loads the Linux kernel image, named **kernel.img**, into the ARM processor's RAM space and the ARM processor is enabled. This starts executing that image and the standard Linux boot process follows thereafter.

The traditional master boot record (MBR, sector 0) of the SD card only contains the partition table information. Unlike on a standard PC, no MBR code is executed.

» **archroot/boot/kernel.img** and copy **modules/lib/modules** to **archroot/usr/lib**.

Emulator, too

You can also use your image with an ARM emulator to run a virtual Raspberry Pi on your x86 PC. *QEMU* is a processor emulator that can run ARM code on your x86-based PC, and **pacman -S qemu** will install it. Create a directory and copy your image there. You'll also need a custom kernel image designed for the emulator. A suitable kernel (kernel-qemu) is provided at **www.linuxformat.com/archives?issue=166** – copy that to the same place as your image. Now you can run an emulator with:

```
qemu-system-arm -machine versatilepb -cpu arm1176 -m 256 \
        -no-reboot -serial stdio -kernel kernel-qemu \
        -append "root=/dev/sda2 panic=1" -hda myimage
```

There is another trick we can perform with an emulator: we can execute ARM code from our x86 command line. This is called transparent emulation, and it uses a kernel feature called **binfmt-misc** to invoke any ARM executable inside *QEMU*. But to achieve this, there are hoops to jump through.

First, you need to have access to **/proc/sys/fs/binfmt_misc**. If you don't have it, then you can mount it:

```
# mount binfmt_misc -t binfmt_misc /proc/sys/fs/binfmt_misc
```

You then need to register the ARM binary format with the kernel. This tells it how to recognise them and what it should to do with them. Recognition is based on pattern matching at the beginning of the file. To register ARM executables:

```
echo ':arm:M::\x7fELF\x01\x01\x01\x00\x00\x00\x00\x00\
x00\x00\x00\x00\x02\x00\x28\x00:\xff\xff\xff\xff\xff\xff\xff\
x00\xff\xff\xff\xff\xff\xff\xff\xfe\xff\xff\xff:/usr/bin/qemu-
arm-static:' > /proc/sys/fs/binfmt_misc/register
```

With that in place, any attempt to execute an ARM binary will result in **/usr/bin/qemu-arm-static** being executed instead. It is passed the ARM binary's path as a parameter. The result is that *QEMU* executes the ARM binary transparently. Putting **/usr/bin/qemu-arm-static** inside your image tree will allow you to **chroot** into it but, for this to work, we need a version of *QEMU* that does not rely on shared libraries, because there are only ARM libraries inside the **chroot**.

This is what **qemu-arm-static** is for. Unfortunately, few distributions provide a statically-linked *QEMU*, so you will have to build it yourself. A pre-built one for x86-64, along with details of how to build it, are in the tutorial archive. Install it as **/usr/bin/qemu-arm-static**.

Now, if you place **qemu-arm-static** on the x86 host at **/usr/bin** and also inside your **archroot** subdirectory at **archroot/usr/bin**, you should be able to **chroot** into that ARM filesystem. Once there you could, for example, use *pacman* to install further packages.

Although it's straightforward to build and execute ARM code on the x86 using these techniques, things can get sticky if you need to build packages, because this introduces dependencies into the mix. In a cross-CPU environment, that can be a problem because the **makepkg** tool expects its dependencies to be installed, and you can't install ARM dependencies on to x86. The easiest way use your x86 horsepower to build for the ARM CPU is to use a tool called **distcc** that allows you to build on your Pi, but have it delegate its compile tasks to one or more distributed clients that have more power. With **distcc** in place, using **makepkg** to build packages on your Pi will allow it to transparently make use of your x86 box's power for its compiling tasks.

To use **distcc** you have to install it on both your Raspberry Pi and your desktop. The command is the same in both places: **pacman -S distcc**. The Pi is the client and controls the builds; you run **makepkg** on the Pi. The only configuration required on the client is to set up **makepkg** to use **distcc**. Edit **/etc/makepkg.conf** with the details of your **distcc** server:

```
BUILDENV=(fakeroot distcc color !ccache)
DISTCC_HOSTS="10.0.200.12"
MAKEFLAGS="-j8"
```

Your PC is the server and makes itself available to perform compiler tasks. Configuration requires specifying the hosts that are allowed to connect; this is done in **/etc/conf.d/distccd**. The server should be started as a daemon (this can also be automated at boot time):

```
/etc/rc.d/distccd start
```

With the client and server configured, running **makepkg** on the client (the Pi) will run the compile jobs on the server. There is one more configuration that is needed so that the server's **distccd** can find the correct compiler: the compiler we want is called **arm-linux-gnueabi-gcc** but **distcc** will look for **gcc**, which is the native x86 compiler. To fix this, create a symbolic link and modify the path for **distccd**:

```
# ln -s /usr/bin/arm-linux-gnueabi/gcc /use/bin/arm-linux-
gnueabi-gcc
# sed -i -e '/^PID=/iPATH=/usr/bin/arm-linux-
gnueabi:\$PATH' /etc/rc.d/distcc
```

It's a good idea to set symbolic links for the whole toolchain. You can get the script to do this from **www.linuxformat.com/archives?issue=166**.

Uprooting

While having all this custom image fun, you may decide that the SD card is not the best place for your root filesystem. It's quite straightforward to move your root partition to another device, such as a USB hard drive. All you need to do is copy the existing root filesystem from the SD card on to a partition on the USB hard drive, resizing it if appropriate:

```
# dd if=/dev/mmcblk0p2 of=/dev/sda1 bs=4M
# fsck /dev/sda1
# resize2fs /dev/sda1
```

Edit **/boot/cmdline.txt** to change the location of the root partition in the boot command. Change the **root=/dev/mmcblk0p2** to **root=/dev/sda1**. Now reboot and your root partition will be on the hard disk. This requires your kernel to have the relevant USB drivers compiled in. The standard one does, so this should not be a problem.

Quick tip

If the modules_install step of the kernel build fails with 'No rule to make target modules/lib/firmware/./', just repeat it and it should work.

Make your image on your PC

With emulation, you can use **mkarchroot** to create the **archroot** subdirectory on your PC instead of on your Pi. You will need to configure the package manager so it downloads ARM packages and also patch **mkarchroot** (it is a Bash script) so it installs **qemu-arm-static** into the **archroot** when it creates it. With this final piece of the puzzle, you can build a complete image for your Pi, along with customised kernel and other packages, on your PC.

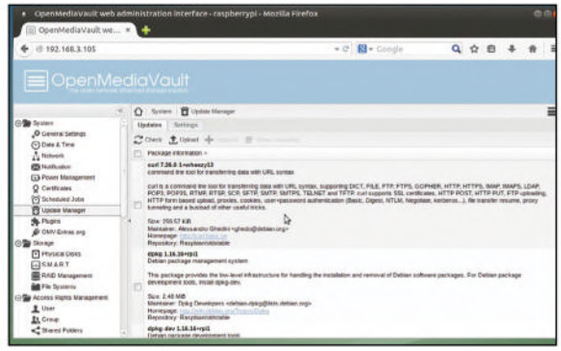
RASPBERRY Pi PROJECTS 2015

Raspberry Pi hardware

Build a network drive 110
Game on an Xbox controller 114
Create a networked printer 118
Using the Pi camera 123
Get started with the GPIO 128
Using the Raspberry Pi 2 130
Build a CCTV with the Pi 134
3D graphics with the GPU 138
The Raspberry Pi 2 142
The Raspberry Pi Model B+ 144
Pi2Go robotics 146
Pi printer 147
The Hover 148
The Kano 149
Bitscope oscilloscope 150

NAS: Open vault

Discover how to manage your data better with a powerful NAS solution.



Head to System > Update Manager and make sure you install all available updates.

Quick tip

If you wish to use the NAS as the target location for storing backups, enable the FTP service. Also enable the SSH service to manage the OMV installation from the CLI.

Do you have a bunch of USB disks that you juggle between your various computers? Did you know that you can plug all of them into a Raspberry Pi, which you can then use as a network attached storage (NAS) box? Using the Pi as an always-on NAS box sounds like a wonderful use of the silent little device. However, setting it up as one used to be an involved process. That's until the Debian-based OpenMediaVault (OMV) distro decided to roll out a version specifically tuned to the Pi.

Once it's up and running, you can configure and manage the distro using its browser-based administration interface. You can then use the USB ports on the Pi to attach USB disks, which are then made available to your entire network for storage. Remember that for best performance, make sure you use self-powered removable disks. You can use the disks attached to the OMV NAS individually, or assemble them in a software RAID array. The distro has ample options to manage other advanced aspects of a NAS distro.

To get started, download the Pi version from the distro's website at **www.openmediavault.org**. The distro has separate releases for the Pi 2 and the original B/B+ models, so ensure you grab the correct one. Then extract the **.img** file from the download and transfer it on to an SD card with

```
sudo dd if=~/omv_1.17_rpi_rpi2.img  of=/dev/sdb
```

replacing **/dev/sdb** with the location of your SD card. If you use Windows, use the *Win32 Disk Imager* app to transfer the image across.

Now boot the Pi with the freshly baked SD card. There's no installation involved and you can start configuring the distro as soon as it boots up. You can access its browser-based interface on the IP address of the Pi – such as **192.168.3.111**. You're asked to authenticate yourself, which you can do using the default credentials for the administrator – **admin:openmediavault**. However, you should change this default as soon as you log in. Head to

Extend your NAS

You can flesh out OMV and add a bunch of features to make it more usable. The distribution supports quite a handful of official and third-party plugins, which you can install and enable as per your needs and requirements. To browse a list of all the officially supported plugins, head to System > Plugins. The page lists over 40 plugins, which are divided into categories such as Administration, Backup, Downloaders, Filesystems, Network and so on. One useful plugin is the *downloader* plugin, which can download files into the NAS, and includes several downloaders such as *Aria2* and *Youtube-DL*. This plugin is well complemented by the *transmission* plugin, which downloads torrents via the *Transmission* app. You should also enable the *clamav* plugin, which gives you the ability to scan your NAS for viruses.

To enable a plugin, simply click on the corresponding checkbox. You can even toggle multiple plugins at one go. After selecting the plugins you wish to enable, click the Install button. OMV then downloads the plugins from the Raspbian repositories via the *apt-get* package management system and enables you to track its progress. Depending on the number of plugins you're installing and their size, this process could take some time to complete.

Once the plugins have been downloaded and installed, they append the OMV administration interface and create an entry for themselves. For example, the *downloader* plugin installs itself under Server > Downloader. Switch to the new section when you want to configure different aspects of the plugin. Each plugin has its own configurable elements.

Stream music

If you've stored music on the NAS, wouldn't it be really cool if you could stream it across the network straight from the NAS itself? Using the *forked-daapd* plugin, you can do just that. To use the plugin, just install it like any other – this adds a new entry under the Services section, labelled iTunes/DAAP.

Before you can stream music, you need to configure the plugin by pointing it to the shared folder on the NAS that contains the music files. Head to the plugin's page and use the Shared folder drop-down menu to select the folder that houses the music. Once you've saved the changes, use a player such as *Rhythmbox, Amarok, Banshee* and so on, which will automatically pick up the DAAP server running on your NAS and enable you to listen to the tracks on the NAS. Use the *DAAP Media Player* app to listen to the music on an Android device. Furthermore, you can also install the *MiniDLNA* plugin to connect to your NAS from DLNA clients. Just as with DAAP, after installing the *MiniDLNA* plugin, you have to head to Services > DLNA > Shares, and click on Add to point to the shared folder that contains the music. You can then use the *BubbleUPnP* app to convert your Android phone into a DLNA compatible device, so that it can browse the library and stream music to and from your now-DLNA-compatible NAS.

System > General Settings in the navigation bar on the left, switch to the Web Administrator Password tab and enter the new password in the appropriate text boxes. You can also use the System menu to configure several aspects of the NAS server, such as the server's date and time, enable plugins *(see 'Extend your NAS')* and keep the system updated.

Add storage

Once it's up and running, plug one or multiple USB disks into the Raspberry Pi. Head to Storage > Physical Disks and click the Scan button to make OMV aware of the disks. Then use the Wipe button to clean the disks individually. If you've inserted multiple disks, OMV can even tie them into a software RAID *(see walkthrough over the page)*. OMV supports multiple RAID levels and each requires a different number of disks. For example, the default RAID level 5 requires a minimum of three disks, while RAID 1, which mirrors data across drives, only needs a minimum of two.

If you don't plan to use the inserted USB disk inside a RAID array, after you've erased a drive, head to Storage > File Systems to create a filesystem on the drive. Here click the Create button and use the pull-down menu to select the device you wish to format. By default, the drives are formatted as EXT4 but you can select a different filesystem using the pull-down menu. Besides EXT4, OMV supports the EXT3, XFS and JFS filesystems. Repeat the process to create a filesystem on all of the attached USB disks. After creating the filesystem, select a drive and then click the Mount button to bring them online.

Before you can store data on the NAS device, you have to create one or more users. To do this, head to Access Rights Management > User. The Add button on this page is a pull-down menu that enables you to either add individual users or import a bunch of users by adding them in the specified format. When adding an individual user, you can also add them to an existing group. By default, all users are added to the Users group.

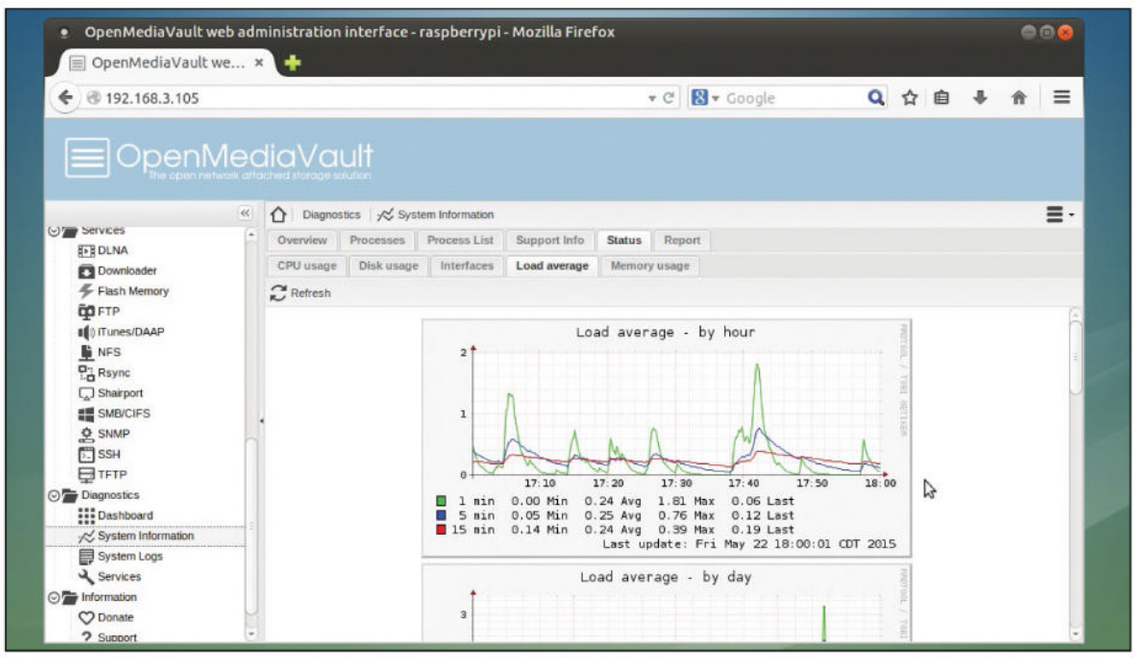
If you wish users to have their own home directories in the OMV server, switch to the Settings tab and mark the checkbox to enable the home directory for the user. You also have to specify the location for the home directory by selecting an existing shared folder on the NAS server or creating a new one.

Shares and permissions

The next step is to define a shared folder. The chief consideration while adding one is whether the NAS will be used by multiple users or a single individual. In case you're going to be sharing the NAS storage space with multiple users, you can define several folders, each one with different »

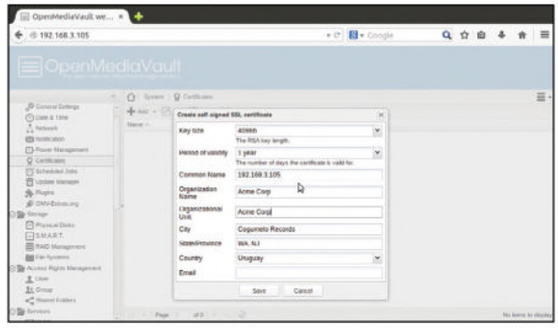
Quick tip

The distro ships with a host of *omv-** utilities, including *omv-release-upgrade*, which upgrades the base to a new release.



OMV keeps tabs on all aspects of the server it's running on. Go to Diagnostics > System Information to see for yourself.

» user permissions. To add a folder, head to Access Rights Management > Shared Folders and click the Add button. In the dialog box that pops up, select the volume that'll house the folder from the pull-down list. Then give the shared folder a name, such as **Backup**, and enter the path of the folder you wish to share, such as **backup/**. OMV creates the folder if it doesn't already exist. You can also optionally add a comment to describe the type of content the folder will hold. Play close attention to the Permissions setting. By default, OMV only allows the administrator and any users you've added to read and write data to this folder, while others can only read its contents. This is a pretty safe default for most installations, but the distro also offers several permutations and combinations of permissions that you can select by using the pull-down menu.

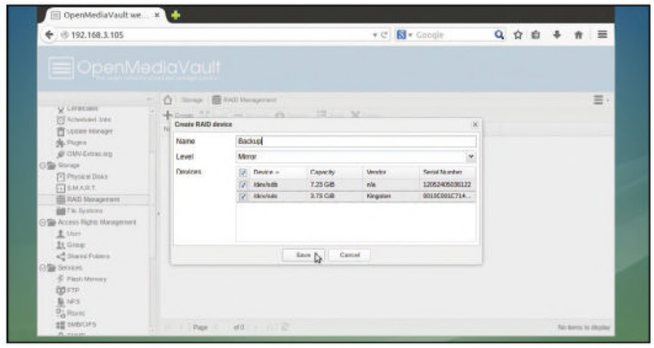


» You can create self-signed security certificates if you don't wish to transfer data to and from your NAS device over unsecured HTTP.

Even if you select the default Permissions setting when creating folders, which lets all users read and write data to the folder, you can fine-tune the access permissions and disable certain users from accessing or modifying the contents of a particular folder. For this, after adding a user, head to the Shared Folders section, select the folder you want to control access to and click the Privileges button. This opens a window with a list of the users you've added, along with checkboxes for controlling their access to that folder.

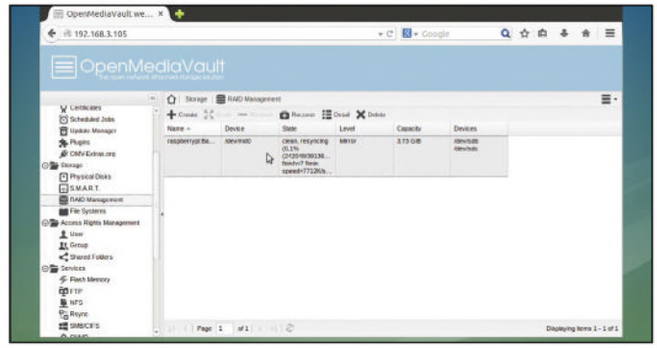
With the users and shared folders set up, you're now ready to share the NAS storage with your network. Follow the walkthrough to enable a network service that people can use

Set up a RAID



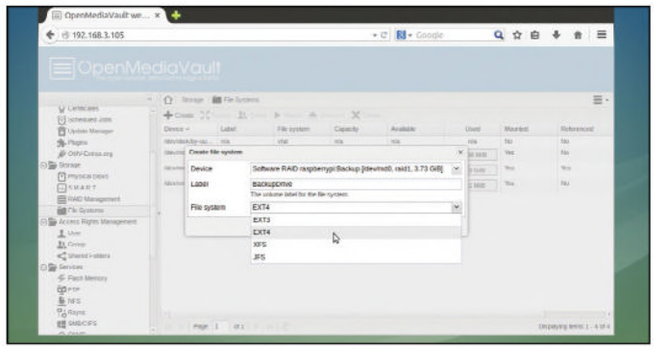
1 Select RAID Level

If you wish to arrange the disks into a RAID device, head to Storage > RAID Management and click the Create button. In the dialog box that pops up, select the devices you want to use in the RAID, as well as the RAID level. Then enter the name you wish to use for the RAID device in the space provided, and click the Save button.



2 Initialise the RAID

After you've created a RAID, OMV asks you to wait until the RAID has been initialised before you proceed to the next step and create a filesystem. You also get a notification to save the changes in order for them to take effect. The RAID Management page now lists the newly created RAID device.



3 Create a filesystem

To use the RAID array, you need to create a filesystem. Head to Storage > Filesystems and click on the Create button. In the dialog box that pops up, select the device you want to format using the pull-down menu, which lists the RAID device you've just created. Then label it and select one of the supported filesystems.

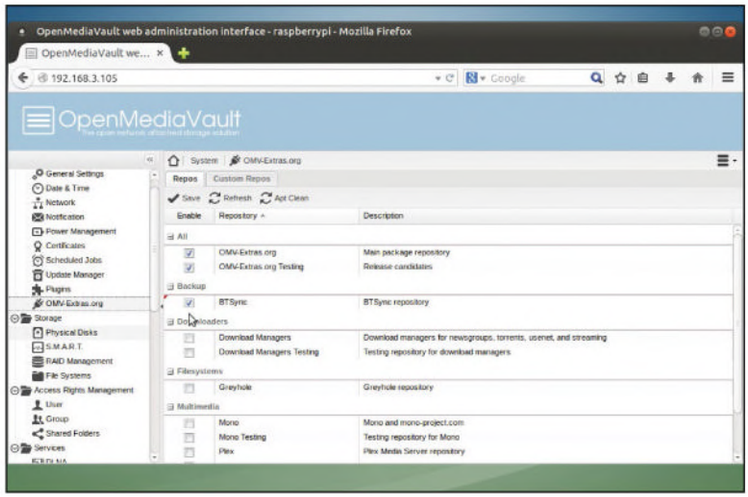
4 Mount the device

After the filesystem has been created, and the disk has been initialised, the RAID device is listed with other devices in the Storage > Filesystems page. To use the drive, select it, then click the Mount button to bring the disk online. You can add new disks to a RAID device by selecting the Storage > RAID Management > Grow option.

to access the shared folders on the NAS. OMV supports various popular protocols and services, including NFS, SMB/CIFS, FTP, TFTP, SSH, rsync and more.

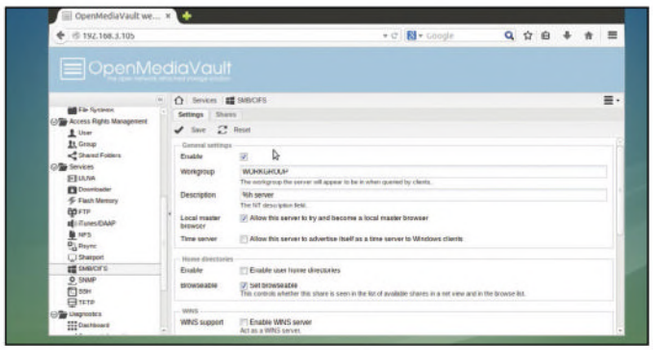
Once you've created a network share, you can access the shared folders from anywhere on the network, irrespective of whether they reside on an individual disk or a RAID array. You can either use your file manager's built-in Network feature to access the network shares, or enter the IP address of the NAS device in the location area, such as **smb://192.168.3.111**. You're prompted for a username and password before you can access the folders – unless, of course, you have marked them as public when adding them via *Samba*. Enter the credentials of the user who has the appropriate permission to access the folder. After they've been verified, OMV mounts the shared folder. You can now upload files into the shared folder or delete them, if you have the permission, just as in a regular folder.

It might take some getting used to but OpenMediaVault is a wonderfully versatile NAS option that helps you exploit the true potential of the Raspberry Pi.



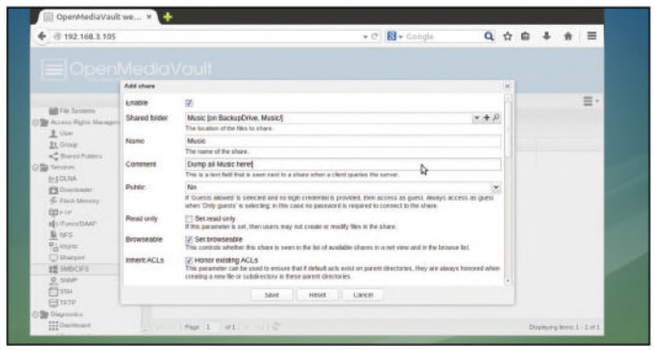
You can fetch additional plugins after enabling more repositories from under the System > OMV-Extras.org > Repos tab.

Enable shares



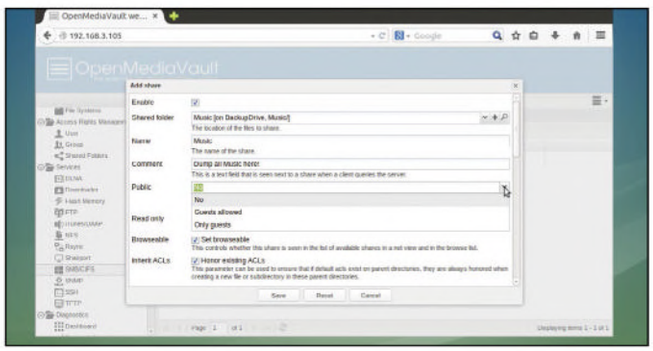
1 Enable Samba

OMV supports several sharing protocols but we'll use the popular SMB protocol commonly known as *Samba,* which works across devices. To activate the service, head to Services > SMB/CIFS and toggle the Enable checkbox. The other settings mentioned on the page are optional. When you're done, click the Save button.



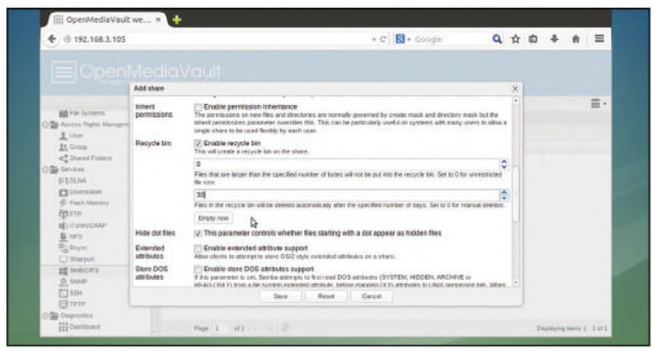
2 Add folders

Next, you have to add the shared folders as *Samba* shares. To do this, switch to the Shares tab and click the Add button. In the window that pops up, select a shared folder from the pull-down list or click on the + button to create a new one. You also have to give the folder a name, which identifies the folder on the network.



3 Define permissions

When adding a *Samba* folder, OMV makes sure it follows the permissions defined when you created the shared folder. Select the Guests Allowed option from the Public pull-down menu to make the folders public. Also, if you select the Set Read Only checkbox, OMV ensures that no user can modify the contents of the folder.

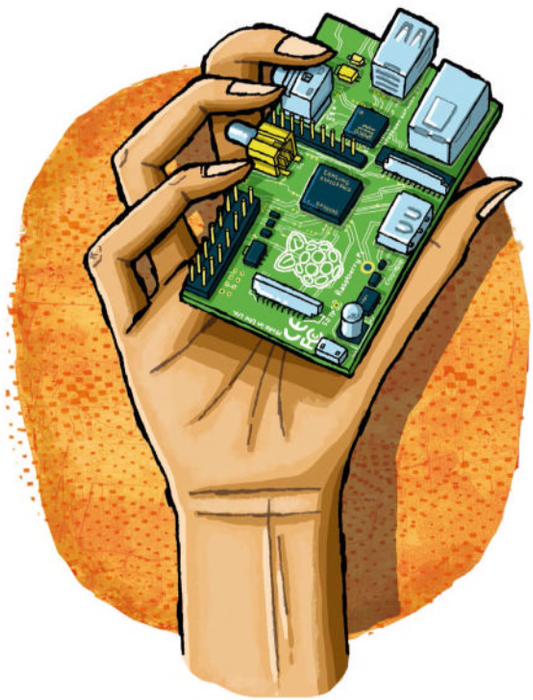


4 Other settings

Take some time to review the other settings on the page. One useful option that isn't enabled by default is the Recycle Bin. When enabled, any file that's deleted from the NAS is moved into a virtual Recycle Bin inside the shared folder. Save the configuration when you've added them all to restart the *Samba* service.

RetroPi: Use an Xbox gamepad

Relive the golden era of gaming by hooking your Pi up to a game controller.



Video games in the '80s were quite different from the latest crop of frag-'em-till-you're-dead point-and-shoot games. They were tastefully crafted 8-bit graphical masterpieces, with an intense storyline, and gameplay that kept you engrossed for hours. If reading this makes you feel nostalgic, you can emulate the golden era of gaming consoles on your modern hardware and escape back to the '80s. The new quad-core Raspberry Pi 2 has enough number-crunching power to recreate the video game consoles of yesteryear virtually. Most of the software that creates the defunct platforms is available as open source software, which you can install on top of a Raspbian distro. However, the easiest way to start playing vintage games on the Raspberry Pi is to install the purpose-made RetroPie distro, which packs a bundle of emulators.

You can manually install RetroPie on top of an existing Raspbian distro but it's more convenient to use the pre-baked image. In addition to Raspberry Pi 2, the distro works with the older models as well, so make sure you grab the correct image. You need to transfer this image to at least a 4GB card, either using the **dd** command in Linux, such as

```
dd if=retropie-rpi2.img of=/dev/sdd
```

or with the *Win32 Disk Imager* app in Windows.

You also need a USB keyboard and mouse for some initial setup that you can't do remotely via SSH. We'll also hook up a compatible Wi-Fi adaptor to the Raspberry Pi, which won't work straight out of the box, but we'll get to that later. Most important of all, make sure you grab some gaming controllers to enjoy the games to the hilt. RetroPie can work with various controllers, from cheap no-names ones to PS3 and Xbox 360 controllers.

Once you have prepared the memory card with the RetroPie image, insert it into the Pi, connect the controller, the Wi-Fi adaptor, the speakers and the USB input devices, hook it up to your HDMI monitor, and power it up. The Pi boots directly into Emulation Station, which is the graphical interface it uses to enable you to switch between emulators. The interface asks you to configure the controller. However, before we do that, we have to tweak a couple of settings. Press the F4 key on the keyboard to exit the Emulation Station, then head to the CLI.

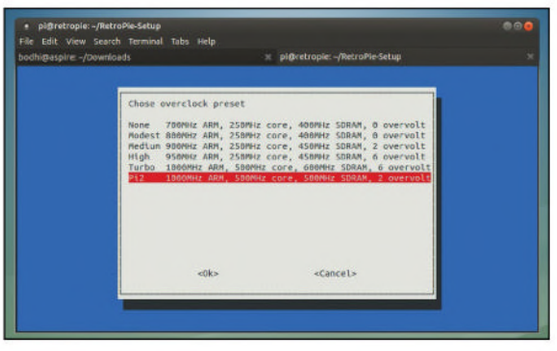
Basic setup

The first order of business is to expand the image to take over the entire card. In order to do this, bring up Raspbian's configuration utility with

```
sudo raspi-config
```

and select the first option to expand the filesystem. Once that's done, head to the second option to change the default password for the **pi** user.

Next up, head to the Advanced Options and select the SSH option to enable remote access. To ensure you use the maximum memory for gaming, head to the Memory Split option. If you're using a Raspberry Pi 2, allocate 512 to the GPU. Users of the older B+ model should earmark 256. Finally, scroll down to the Overclock option, where users of the Raspberry Pi 2 should select the Pi2 option. Once you've made all the changes, head back to the main menu and select



Just to be on the safe side, try playing games with the default clock speeds, before you try to overclock the Pi.

Upgrade RetroPie

The RetroPie script is a wonderful tool that you can use to convert a stock Raspbian distro into the ultimate arcade machine. Or, if you are already running a version of RetroPie, you can use the script to update to a newer version without downloading and reinstalling the whole thing all over again.

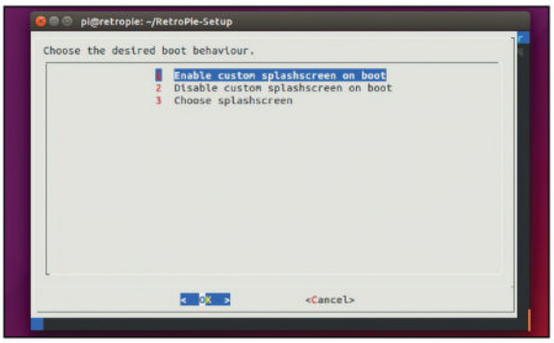
To upgrade your installation, exit *Emulation Station* and enter the following in the CLI:

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

The above commands refresh the distro's repositories and brings it up to date by re-installing any outdated packages. With the base distro updated, it's time to update the various gaming emulators. Again on the CLI, type:

```
$ cd RetroPie-Setup
$ sudo ./retropie_setup.sh
```

As you are in the script, the first task is to update the RetroPie-Setup script itself. The script lists two different upgrade options at the top. The first one fetches pre-built binaries of the emulators, while the second one compiles them from source. The former option is faster, while the latter, although excruciatingly slow on the Raspberry Pi, fetches the bleeding-edge versions of the emulators. You can safely ignore the second option and just go with the first one, which downloads and sets up new versions of all of the emulators.



If you like, you can change the splash screen from the RetroPie-Setup script.

the Finish option to restart the Raspberry Pi and bring the changes into effect.

When you're back up again, press F4 once more to exit out of Emulation Station. We'll now get the Wi-Fi adaptor to work. Open the configuration file with

```
sudo nano /etc/network/interfaces
```

and then change its contents to resemble the following:

```
auto lo
iface lo inet loopback
iface eth0 inet dhcp


allow-hotplug wlan0
auto wlan0
iface wlan0 inet dhcp
  wpa-ssid "Your Wireless Network Name"
  wpa-psk "Your Wireless Network Password"
```

Make sure you replace the text between the **" "** with the SSID and password for your Wi-Fi network. Press Ctrl+X to save the file and exit the text editor. Now reboot the Pi with **sudo reboot**. Once it comes back up, your Wi-Fi adaptor connects you to your router. From this point on, you can do the configuration remotely from another computer.

Exit Emulation Station yet again and make a note of the IP address RetroPie has been assigned by your router. Assuming it is **192.168.3.111**, you can now log in to it from another computer with **sudo ssh pi@192.168.3.111**.

Irrespective of how you're accessing the Pi, the next order of business is to tweak some RetroPie-related settings. Change into the **RetroPie-Setup** directory with

```
cd ~/RetroPie-Setup
```

and execute the configuration script with

```
sudo ./retropie_setup.sh
```

The script fetches any required packages that are missing from your installation. When it's ready, the script displays an Ncurses-based menu. First up, scroll down to the second-to-last option, which updates the RetroPie-Setup script itself. Once it's done, re-launch the script and scroll down to the third option, labelled Setup/Configuration.

In here, scroll down and select option 323, which makes the necessary changes to display the RetroPie configuration menu in Emulation Station. This helps you make changes to the distro without heading back to the CLI interface. Now, depending on your audio gear and how it's connected to the Raspberry Pi, you might need to handhold RetroPie before it can send audio output correctly. Select option number 301 to configure the audio settings. If the default auto option isn't playing any sound, scroll down and select the output to which your speakers are connected. The menu also gives you the option to bring up the mixer to adjust the volume.

Quick tip

To map an exit option to the game controller, edit **retroarch.cfg** and add **input_enable_hotkey_btn = "X"** and **input_exit_emulator_btn = "Y"**. Replace **X** and **Y** with the identifiers for the Start and Menu buttons on your controller.

Configure controllers

Now reboot the distribution one last time and this time continue with Emulation Station. If you've connected your controller, the distro picks it up. Press and hold any key on the controller to help the distro correctly identify the controller. It then asks you to map the keys on the controller. Be aware that this basic mapping is only for navigating the graphical interface and helping you switch between the emulated system and selecting a game. Once you've set up the controller, you're dropped into the main menu of the Emulation Station interface.

Now, to set up the controller for gaming, head to the RetroPie menu in Emulation Station and select the Configure RetroArch Keyboard/Joystick option. Use the keyboard and select the first option, labelled Configure Joystick/Controller. Then follow the on-screen prompts to set up your controller. If

Emulation Station displays the number of games inside a particular emulator.

Quick tip

When using multiple controllers at the same time, it's best to use identical controllers to avoid configuration and gameplay issues.

» your controller doesn't have the buttons you're being asked for, just wait for a few seconds and the setup will move on to the next button.

If you use an Xbox 360 or a PS3 controller, you first have to install their drivers before RetroPie can pick them up. In earlier versions, this involved some hacking on the command line. However, in the latest version of the distro, it's a very simple and straightforward affair. Head to the RetroPie-Setup option in the RetroPie menu inside Emulation Station. This brings you to the Ncurses menu of the RetroPie-Setup script we were in earlier. Use the keyboard to select the third option to configure the distro. Scroll through the list and select the relevant option to install the driver for your controller – number 318 to install the PS3 driver and number 332 to install the driver for the Xbox 360 controller.

The Xbox360 script downloads the xboxdrv driver and edits the **/etc/rc.local** file to start the driver on boot. The script adds entries for wired 360 controllers. If you are using wireless controllers, open the **/etc/rc.local** file in a text editor, hunt for the lines that begin with **xboxdrv** and replace the **--id** option with **--wid**.

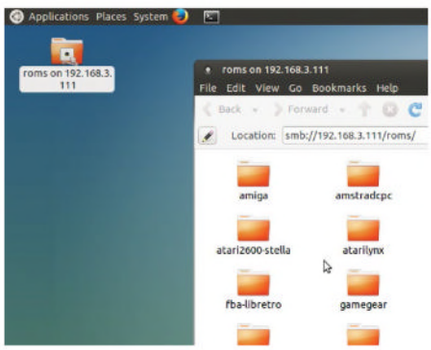
If you are using PS3 controllers, once you've installed the drivers using the script as described earlier, you're prompted to plug in the Bluetooth adaptor for the controllers. Even after you do so, RetroPie will fail to detect your controllers. This is to be expected, according to the developers. Exit the script and out of Emulation Station. Once you're back on the command line, switch to the **/opt/retropie/supplementary/ps3controller/** directory and type

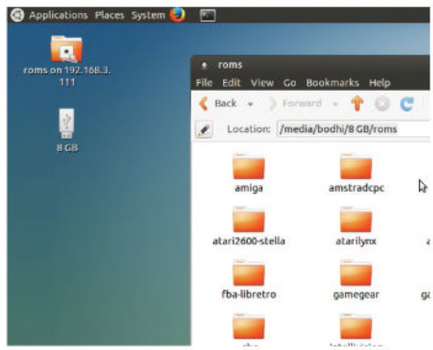
```
sudo ./sixpair
```

This nifty little utility should detect the Bluetooth adaptor and make it known to RetroPie.

Now reboot the Raspberry Pi and, once it's back up, change to the **/dev/input** directory and list its contents with **ls**. If your controller has been detected, it's listed as **js0**. You can test the controller by using:

Transfer ROMs





1 Network transfers

If RetroPie is connected to the router, you can transfer game ROMs to it from any computer on the same network. The distro ships with a pre-configured *Samba* server and shows up as a Windows share. Copy the ROMs inside the directory for their particular emulator.

2 Via USB

The easiest way to transfer ROMs is to use a USB flash drive. When it detects a USB disk, RetroPie creates a directory structure for ROM files that mirrors the emulators installed on the distro. Wait a while as it creates the directories and then remove the drive.

3 Plug and play

Now put that USB stick into your desktop PC and copy the ROMs on to it, making sure to put them in the right folder. When you put this memory stick back in your Pi, RetroPie pulls the ROMs into the matching directory for the associated emulators automatically.

Use a virtual gamepad

Don't sweat if you don't have a gaming controller – you can create and use a virtual one from within your phone or tablet instead.

To create the virtual gamepad, head down to the CLI and enter the following commands to install the required components:

```
$ sudo apt-get update
$ sudo apt-get upgrade
$ wget http://node-arm.herokuapp.com/node_latest_armhf.deb
$ sudo dpkg -i node_latest_armhf.deb
```

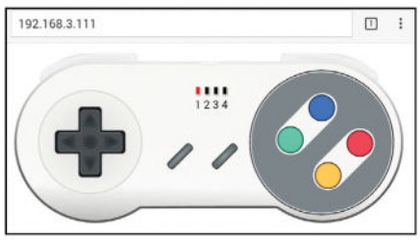
Once you have the components, switch to the root user with the **su** command. You're prompted for the root user's password (**raspberry**). Once authenticated, enter

```
# git clone https://github.com/miroof/node-virtual-gamepads
# cd node-virtual-gamepads
# npm install
# npm install pm2 -g
```

The above steps take a little time to complete. Once they've finished, you can launch the controller and enable it to start up automatically at boot:

```
# pm2 start main.js
# pm2 startup
# sudo pm2 save
```

Now grab your phone or tablet, open the web browser (the developers recommend *Google Chrome* for best results) and enter the IP address of the Pi in the address bar. You should now see a virtual controller on the page. Note that you need to configure your controller with Emulation Station and *RetroArch* just as you would with a physical controller. The game controller web application also provides haptic feedback – if you find it irritating, you can deactivate it by taking your device off vibration.

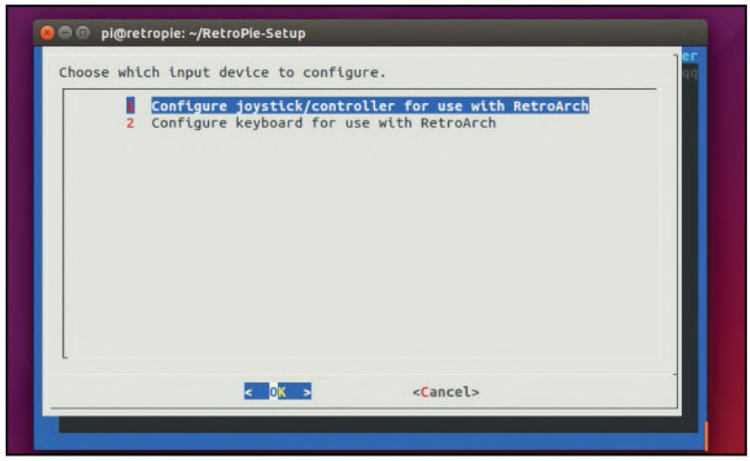


You can hook up as many as four virtual controllers to RetroPie.

```
jstest /dev/input/js0
```

which brings up the *jstest* program designed to test the features of a controller. Now head back to the RetroPie menu in Emulation Station and use the Configure RetroArch Keyboard/Joystick option to set up your controller. And that's it – your controllers are now all set up and ready to go. You can do this with all your controllers and RetroPie saves the configuration and automatically loads it whenever you plug the controller in.

You can now scroll through Emulation Station and play the pre-installed games with your controllers. When you're done with those, follow the walkthrough to transfer your own gaming ROMs into the RetroPie. There are several sites, such as World of Spectrum (**www.worldofspectrum.org**), that host legally downloadable ROMs for free, donated or abandoned by their developers. True retro gaming fans will have created their own ROMs from old cartridges, though, which isn't too tricky thanks to adaptors like the Retrode.



› ***RetroArch*** **is a multi-system emulator that does the heavy-lifting for the distro.**

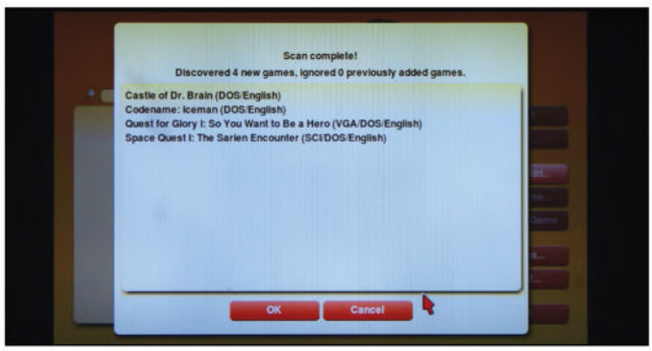
Play games in ScummVM

1 Configure ScummVM

Start the launcher and click on the Options button. Switch to the last tab, which houses miscellaneous settings. Use the Theme button to change the visual appearance of the launcher by switching to another theme. The GUI Renderer setting defines how the launcher is rendered, and the Autosave option controls the length of time that ScummVM waits between saves.

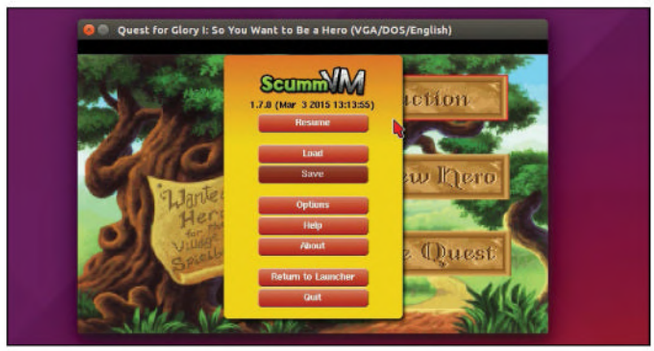
2 Default paths

Switch to the Paths tab to configure where ScummVM looks for particular files. The Save Path option points to the default folder where ScummVM will store saved games. If the option isn't set, the saved games are stored in the current directory. Then there's the Theme Path option, which points to the directory that stores additional themes for the launcher.



3 Add games

To load a supported game into ScummVM, copy its data files from the original media. If you've downloaded the files from ScummVM's website, you have to extract them before copying them into RetroPie. Then run ScummVM, press the Mass Add button and point ScummVM to the extracted folder. It auto-detects any games in there and they appear in the game list.

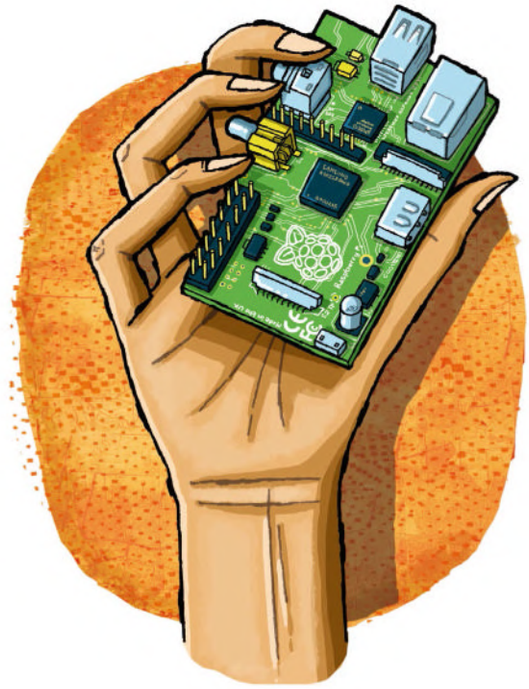


4 Global menu

Now select the game you wish to play and press Start. While playing the game, you can press the Ctrl+F5 key combination to pause the game and bring up the global menu. This gives you the option to get help and influence gameplay. Using the Help button, you can access any in-game help documentation, while the Options button enables you to tweak certain settings, such as volume.

CUPS: Printing

Using the Raspberry Pi as a wireless print server is easy when you know how.



A printer isn't the most convenient of peripherals. They look out of place on most work desks and create quite a racket when spitting out pages. You could throw a few hundred quid on a snazzy new network printer that sits in a corner somewhere and can receive print orders from any computer on the local network. Or, you could just hook your regular USB printer to the Raspberry Pi and enjoy the same conveniences offered by top of the line network printers. If you haven't already used your printer on Linux, before you get started with this project head to **www.openprinting.org/printers** to check whether your printer is compatible with the *CUPS* printing server software. If your printer is listed, hook it up to the Raspberry Pi using one of the USB ports. For this project, we're using the Raspbian distro and the Pi is connected to the local network via a compatible wireless adaptor. However, you can also hook the Pi up to your network via the wired Ethernet port.

You can follow the instructions in this tutorial by accessing the Raspberry Pi remotely from any other computer on the network. Just make sure that the SSH server inside Raspbian is enabled by using the *raspi-config* tool. It's also a good idea to assign a fixed IP address to the Pi. You can do this easily from within your router's admin page. For this tutorial we'll assume that the IP address of your PI is **192.168.3.111**. You can now access the Pi from within Windows using the *PuTTY* client or from any Linux distro with the **SSH** CLI command, such as:

```
$ sudo ssh pi@192.168.3.111
```

Install CUPS

Once you're inside Raspbian, update the repositories with:

```
$ sudo apt-get update
```

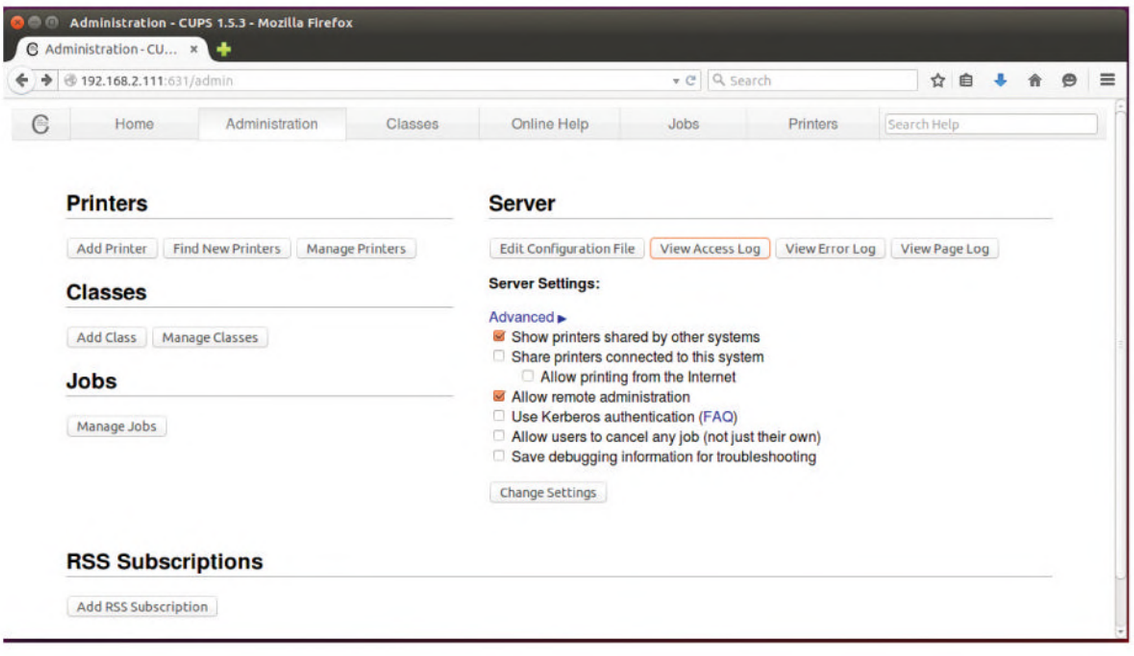
and then install any updates with:

```
$ sudo apt-get upgrade
```

Now pull in the *CUPS* print server with:

```
$ sudo apt-get install cups
```

When it's installed, add your user to the group created by *CUPS* called lpadmin that has access to the printer queue. Unless you have created a custom user, the default user on Raspbian is named pi. Use the following command to allow it



› You can also browse through its extensive documentation from the *CUPS* browser-based control panel.

CUPS command-line utilities

The *CUPS* printing system ships with a number of nifty little command-line utilities. In fact, you can set up and configure all aspects of your printer from the CLI. Let's run through some of the most useful commands that will help you manage the printer better.

We've already seen the **lp** command, which queues a file for printing on the default printer. The default printer is specified in the **PRINTER** variable. You can specify it with the command **export PRINTER=printer-name** where **printer-name** is the name of the printer you specify in step 2 of the walkthrough over the page.

If you have multiple printers, use the **-d** option to specify the printer you wish to print to. For example, **lp -d HP-printer file.txt** prints the file on the HP printer, which isn't set as the default.

To influence the characteristics of the printed output, use the **-o** option to specify a variety of options. For example, **lp -o landscape -o fit-to-page -o media A4 file.jpg** fits the image into A4 size specifications and prints it in landscape mode. Refer to the *CUPS* documentation (**www.cups.org/documentation.php/options.html#OPTIONS**) for a list of all the supported printing options.

If you mistakenly print a large file and want to stop the print job before you waste too much paper, you can use the **lpq** command to print a list of all the print jobs currently in the queue.

The command also lists the file that each job is printing and its size, so you can easily identify the job ID assigned to each. Make note of it because you need it to cancel the print job. The command **cancel 2**, for instance, cancels the job with the ID 2.

to interact with the printer:

```
$ sudo usermod -a -G lpadmin pi
```

Here we use the *usermod* tool to add (**-a**) the pi user to the lpadmin group (**-G**). By default, *CUPS* can only be configured from the local computer that it's installed on. Because that doesn't work in our case, we need to edit its configuration file to allow us to make changes to the server from a remote computer.

First of all, you need to create a backup of the original configuration file with:

```
$ sudo cp /etc/cups/cupsd.conf /etc/cups/cupsd.conf.orig
```

Then open the file with the *nano* text editor:

```
$ sudo nano /etc/cups/cupsd.conf
```

Inside the file, scroll down to the following section:

```
# Only listen for connections from the local machine
Listen localhost:631
```

Comment out that line and add another to ask *CUPS* to accept connects from any computer on the network. Make sure the section looks like this:

```
# Only listen for connections from the local machine
# Listen localhost:631
Port 631
```

Then scroll further down in the config file until you reach the **<Location>** sections, and add a new line that reads **Allow @local** just before the close of the section. The section with the appended line should now read like this:

```
< Location / >
# Restrict access to the server
Order allow,deny
Allow @local
< /Location >
```

Now add the **Allow @local** line to the other two **Location** sections – **<Location /admin>** and **<Location /admin/conf>**. Save the file and restart the *CUPS* server with:

```
$ sudo /etc/init.d/cups restart
```

You should now be able to access the *CUPS* administration panel via any computer on your local network by pointing the web browser to your Pi. Then follow the walkthrough over the page to add your printer to *CUPS*.

Some Linux distros ship with a restrictive iptables firewall policy that doesn't allow connections via the *CUPS* ports. Even if Raspbian doesn't, make sure it doesn't throw up any unexpected errors by punching holes in the firewall with:

```
$ sudo iptables -A INPUT -i wlan0 -p tcp -m tcp --dport 631 -j ACCEPT
$ sudo iptables -A INPUT -i wlan0 -p udp -m udp --dport 631 -j ACCEPT
```

If you connect to the Pi via Ethernet instead of a wireless adaptor, modify the command and replace **wlan0** with **eth0**.

Network-wide access

When you are through setting up your printer using the *CUPS* administration panel, it's time to make it accessible to other machines on your network. While Linux distros will have no trouble detecting your new network printer, making them visible to Windows and Apple devices requires a couple of extra steps.

For Windows, install the *Samba* server on the Pi with:

```
$ sudo apt-get install samba
```

Then open its configuration file (**/etc/samba/smb.conf**) in the *nano* text editor and hunt for the section labelled **[printers]** and make sure it contains the following line:

```
guest ok = yes
```

Then scroll down to the **[print$]** section and change its path to the following:

```
path = /usr/share/cups/drivers
```

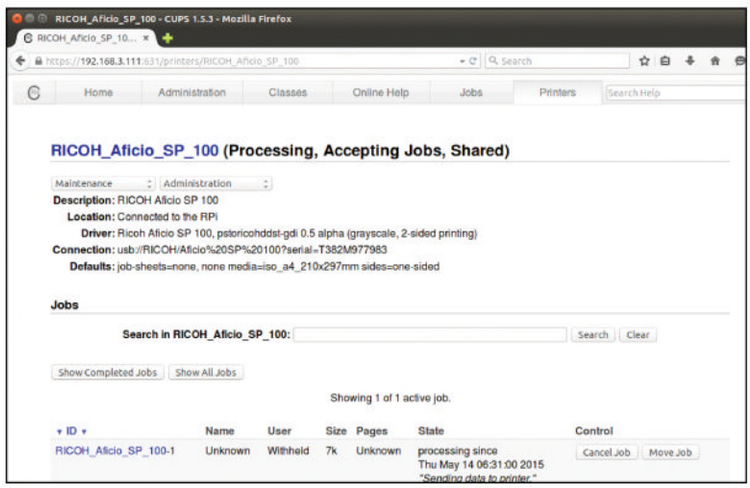
Then scroll up to the Global Settings section at the top of the configuration file. Modify the workgroup parameter within to point to the name of your workgroup, which by default is named WORKGROUP. Also enable the wins support by adding the following line:

```
wins support = yes
```

Now save the file and restart *Samba* with:

```
$ sudo /etc/init.d/samba restart
```

Then head over to the Windows machine and launch the Add New Printer wizard and click on the option to install a network printer. Thanks to the modified *Samba* configuration, the wizard detects and lists any printers hooked up to the Pi. If you have Apple devices, you can enable support for Apple's »



From the Printers tab, you can track the status of every job on every printer.

» AirPrint system, which allows you to print from the iPad and iPhone. For this, just install the *Avahi* daemon with **sudo apt-get install avahi-daemon** on the Pi, which will then make the connected printer visible to AirPrint-compatible devices.

Print from Python

In addition to the ability to use our network printer from within graphical apps across all platforms, we can also use it to print from the command line interface. Furthermore, we can also interact with the printer using the Python programming language.

The *CUPS* printing server installs a bunch of command-line tools *(see box on page 119)* for interacting with the server and any connected printers. You can send files to the printer using the **lp** command, such as:

```
$ lp ~/docs/a_text_file.txt
```

If you have multiple printers, you can print to a specific printer by specifying its name, such as:

```
$ lp ~/docs/another-text.txt -d EPSON_LX-300
```

When you use the commands with a PDF or image file, *CUPS* converts the files using the printer drivers. You can also use Python to generate printer-friendly content. This is best done with the PyCups library, which provides Python bindings for the *CUPS* server.

Install the library with:

```
$ sudo apt-get install python-cups.
```

Then create an **example.py** Python script with:

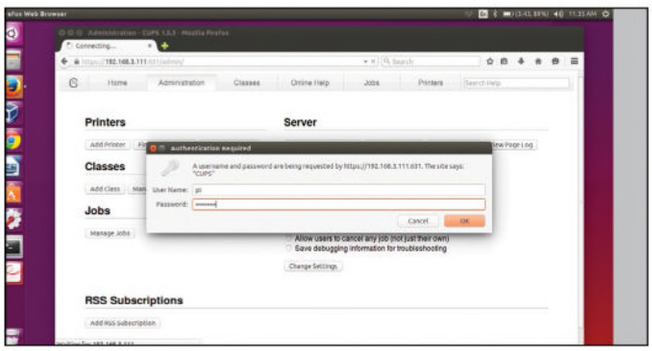
```
import cups
conn = cups.Connection()
printers = conn.getPrinters ()
for printer in printers:
print printer, printers[printer]["device-uri"]
```

The script fetches details about all the printers managed by *CUPS* and prints their name and device address to the screen. When you execute the script, it produces an output similar to the following:

```
EPSON_LX-300 usb://EPSON/LX-300+?serial=L010209081
RICOH_Aficio_SP_100 usb://RICOH/
Aficio?serial=T382M977983
```

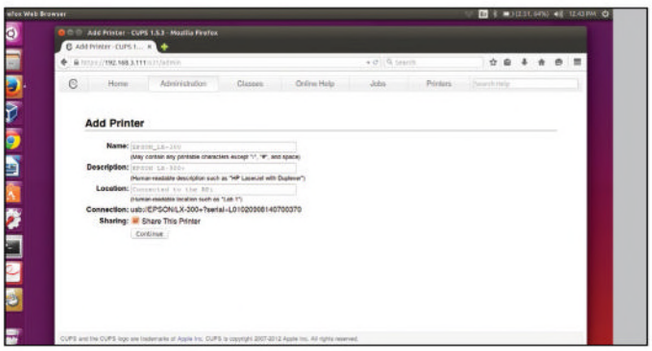
You can also print files from the Python script using the

Add a printer



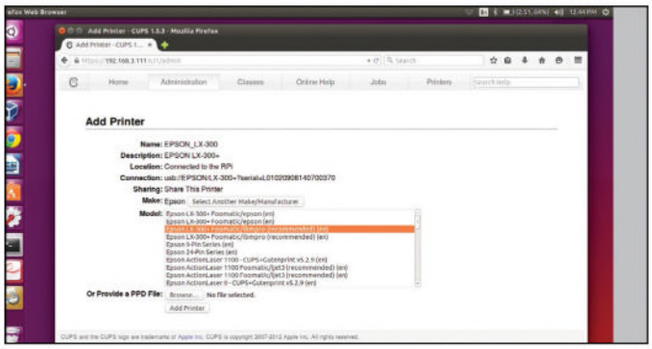
1 The CUPS dashboard

The *CUPS* print server includes a built-in web server that powers its configuration panel. It's running on port 631 on the Raspberry Pi, which in our case is **192.168.3.111:631**. Access the address from any browser on the network. You have to accept its security certificate and then log in to the interface using the credentials of the user you've added to the lpadmin group, which in our case is the pi user.



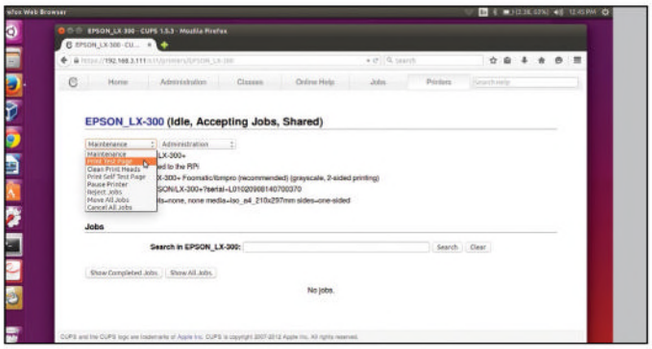
2 Add a printer

Once logged in, switch to the Administration tab and click on the Add Printer button, which brings up a list of printers. Toggle the radio button next to your printer and head to the next step. Here you're asked to add or edit the name, description and location of the printer. Make sure you enable the Share This Printer option to make the printer accessible all over the network.



3 Select a driver

In the next step, you're asked to choose a driver for the selected printer. *CUPS* shows you a list of drivers based on the make of printer. Chances are that several of the drivers are marked 'Recommended'. However, scroll through the list until you find the driver for your exact model. Alternatively, if you have a PPD file for the printer's driver, click on the Browse button and navigate to it.



4 Set default options

In the final step, *CUPS* enables you to set some generic print settings, such as page size and source. The number and type of options vary from one printer to another, and might spread over several sections. When you've finished setting your preferences, click Set Default Options. You're then taken to the main administration page for that printer. Use the Maintenance pull-down menu to print a test page.

printFile function, by specifying them in the format:

```
$ printFile (name of the printer, filename to print, job title, options)
```

Open the previous **example.py** script and add the following lines to it:

```
file = "/home/pi/testfile.txt"
printer_name=printers.keys()[0]
conn.printFile (printer_name, file, "Project Report", {})
```

The first line saves the name of the file you wish to print inside a variable named **file**. The second line fetches the list of printers and saves the first name, which is the default printer inside a variable named **printer_name**. The third line then uses the first two variables and prints the file in the specified format.

A more interesting example is to use the *wkHTMLtoPDF* toolkit to convert HTML pages into PDF files and then pass them on to the printer, from within a Python script.

Before you can install the toolkit, first install the required components and a set of fonts to process the web pages:

```
$ sudo apt-get install xvfb xfonts-100dpi xfonts-75dpi xfonts-scalable xfonts-cyrillic
```

Then install the tool with **sudo apt-get install wkhtmltopdf** before installing the Python wrapper with:

```
$ sudo pip install git+https://github.com/qoda/python-wkhtmltopdf.git
```

You can now use the following to convert a web page into a PDF file:

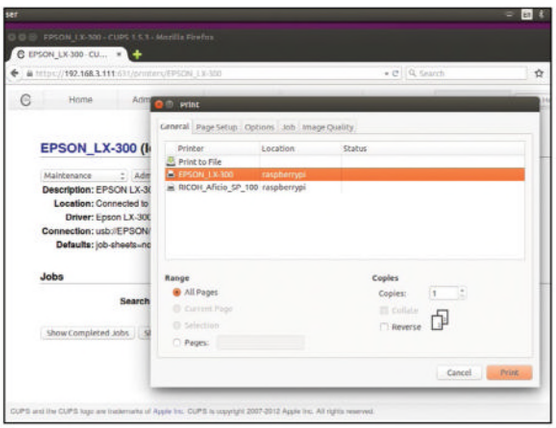
```
from wkhtmltopdf import WKHtmlToPdf

wkhtmltopdf = WKHtmlToPdf (
    url='http://www.linuxformat.com',
    output_file='/home/pi/docs/lxf.pdf',
)
wkhtmltopdf.render()
```

When executed, the above code saves the main page of the *Linux Format* website into a PDF file under the **/home/pi/docs** directory.

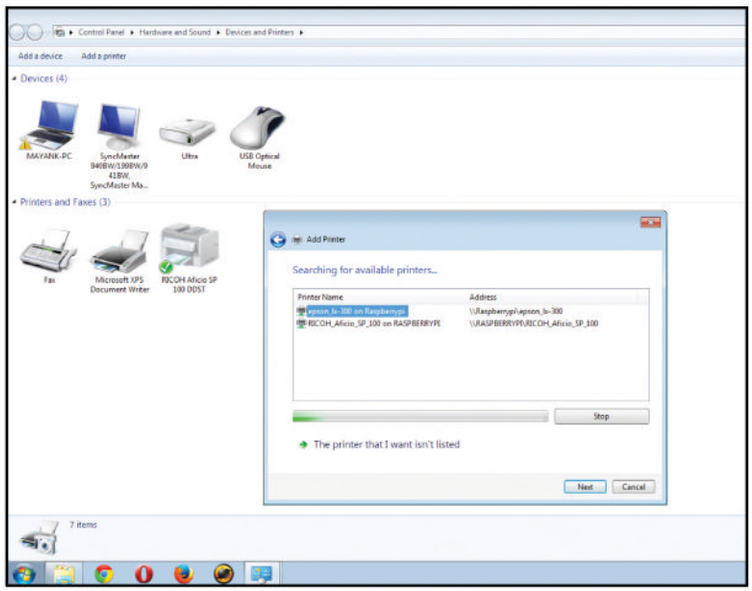
Refer to the listing below to see how the pieces fit together – it converts a page into a PDF and prints it out.

```
#!/usr/bin/env python
import cups
from wkhtmltopdf import WKHtmlToPdf
wkhtmltopdf = WKHtmlToPdf(
    url='http://www.tuxradar.com',
    output_file='/home/pi/tuxradar.pdf',
)
wkhtmltopdf.render()
conn = cups.Connection()
printers = conn.getPrinters()
for printer in printers:
        print printer, printers[printer]["device-uri"]
file="/home/pi/tuxradar.pdf"
printer_name=printers.keys()[0]
conn.printFile (printer_name, file, "PDF Print", {})
```

The script first converts the homepage of **www.tuxradar.com** into a PDF. It then connects to *CUPS*, prints a list of attached and configured printers on the screen, and uses the default printer to print the PDF. The PyCups library is chock-full of methods (**https://pythonhosted.org/pycups/**) that you can use to control all aspects of the *CUPS* print server. Happy hacking!



All Linux distros can access the USB printers connected to the Raspberry Pi without any tweaks to the distro.



You have to install and configure *Samba* to access the network printers on Windows.

Administering CUPS

In addition to adding printers, the *CUPS* web interface provides access to various other useful settings. You can administer most of the printing tasks from the Administration tab, which houses settings under various different categories. Under the Server section, for instance, you can find options to tweak the configuration of the server as well as view various types of access and error logs.

Using the Manage Printers button under the Printer section, you can control the settings for individual printers. Every printer's page has options rolled under two pull-down menus labelled Maintenance and Administration. From under the Maintenance menu, you can print a test page, a self-test page, clean print heads and manage print jobs.

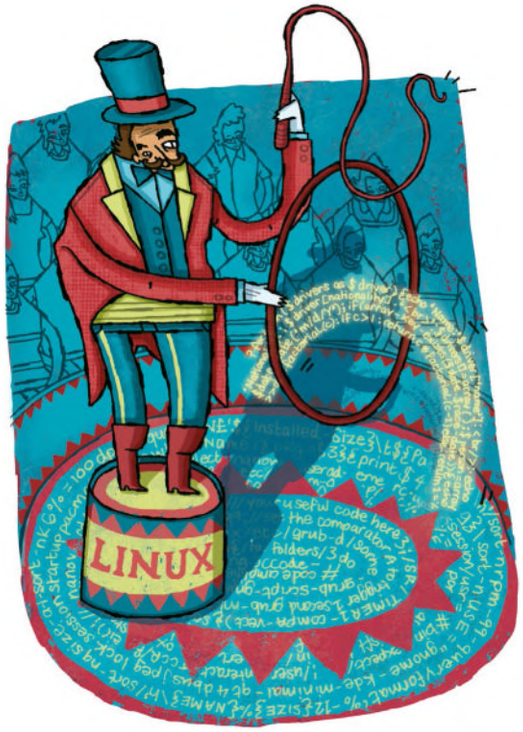
To customise the behaviour of the printer, use the Administration menu to tweak its default options, set it as the default printer, restrict user access, modify its settings or delete it from the *CUPS* server altogether. Besides the Administration tab, there are a couple of other important tabs we should mention as well.

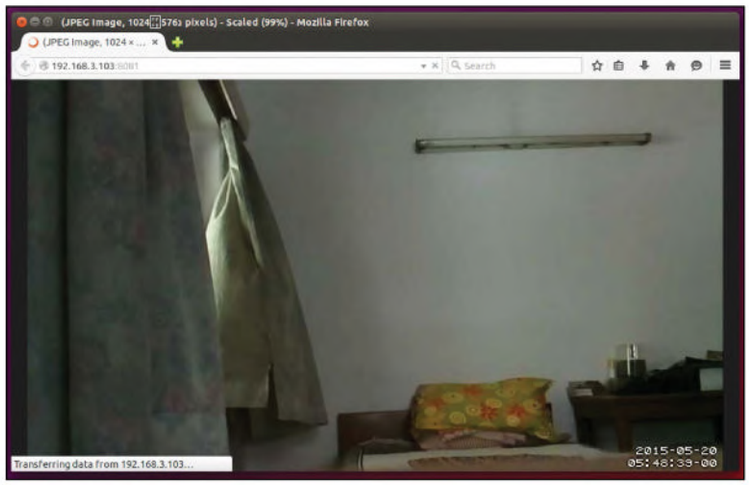
For starters, you need to switch to the Classes tab for printer class management. A class is a collection of several printers.

When you send print job to a class, *CUPS* automatically assigns the job to the next available printer, instead of waiting for a specific printer. Then there's the Jobs tab, which enables you to view and manage all print jobs that are currently in the print queue.

PiCam projects

The Pi camera module can be used for amusing activities – let's take a look.





Poke holes in your firewall to access your motion-detecting Raspberry Pi camera from anywhere on the internet.

Just like the board it complements, the official Raspberry Pi camera module is fun little gadget that you can use for a variety of video-related projects. The camera module is a Full HD camera, which plugs into the Pi via the Camera Serial Interface (CSI) that's located next to the Ethernet port on the device. The optics on the camera feature a 5MP sensor with fixed focus. It can shoot still images with a maximum resolution of 2,592×1,944 as well as Full HD 1080p video at 30fps, and 720p video at 60fps. And you get all this in a module that's only 25x20x9mm and weighs just 9g. This makes it ideal for projects that require a small high resolution camera, such as surveillance.

Before you attach the camera, locate the CSI and then pull the tab gently up. Now push the camera module's ribbon cable into the slot, with the silver contacts on the cable facing away from the Ethernet port. Remember not to push the cable in very deep. Now hold it in place with one hand and push the CSI tab back down with the other hand to lock the camera's ribbon.

With the hardware in place, it's now time to set up the software. Boot into Raspbian and log in. Before enabling the camera, make sure you refresh its repositories with **sudo apt-get update** and then install any available updates with **sudo apt-get upgrade**.

Once you're running the latest version of Raspbian, launch the Raspberry Pi's configuration script with **sudo raspi-config** to tweak its settings. Scroll down the list to the Enable Camera option to make the Pi aware of the newly connected peripheral. Then exit the utility and restart the Pi.

When it boots up, you can test the camera by using the two command-line utilities, *Raspistill* and *Raspivid,* to capture still images and videos respectively. The command **raspistill -o image.jpg** displays a preview from the camera for five seconds and then takes a picture, which is saved as **image.jpg** in the current folder. Refer to the box on page 125 to run through some of the most interesting options.

You can take advantage of the minuscule nature of the camera and the portability of the Raspberry Pi to capture

Lightweight motion detection

If the *Motion* app is overkill for your own requirements, try the lightweight motion detection Python script written by Raspberry Pi community members. It relies on the Python Imaging Library, which is used for analysing and manipulating images, so install it with:

```
$ sudo apt-get install python-imaging-tk.
```

Grab the script and make it executable with:

```
$ wget -c http://pastebin.com/raw.php?i=yH7JHz9w -O picam.py
$ chmod +x picam.py
```

The script stores images in a directory named **picam** under your home directory, so make sure to create it with **mkdir ~/picam** before executing the script with **./picam.py**. The script starts taking low-resolution images. It then looks for movement by comparing the pixels in the images. If it detects changes, the script captures a higher-resolution image. It then automatically removes the low-res images it captures for comparison and only stores the high-res ones that have captured the motion. These are saved in the **~/picam** folder.

You'll need to adjust some aspects of the script to make sure it works for you. For example, as per the default configuration, the script even detects minute changes that have been caused by the wind. Open it in a text editor and set the threshold variable to a higher value than the default.

time-lapse video of things that are too subtle for the human eye to notice, such as the movement of stars or the blossoming of flowers. A time-lapse video is composed of several images captured over several hours but played back at a much faster pace. For example, you can capture an image every 10 seconds for six hours, and then string them together in a time-lapse video at 24 frames per second. This reduces the six-hour capture to a 30-second clip.

Watch time-lapse

It doesn't take much work, on the software side, to set up the Pi for time-lapse photography. We'll use the command-line *Raspistill* utility to capture images inside a folder, such as:

```
$ mkdir ~/images
$ cd ~/images
$ raspistill -o image-%04d.jpg -t 21600000 -tl 30000 &
```

This command captures a shot every 30,000 milliseconds, or 30 seconds, for the next 21,600,000 milliseconds, or six hours. You can change the duration (**-t** option) or the gap between the shots (**-tl** option) by specifying a different time in milliseconds.

The **%04** tells *Raspistill* to save the image in a file called **image** followed by an incremental list of four-digit numbers. The **&** sign at the end of the command asks the *Bash* shell to run the command or the job in the background. After six hours, you'll have 720 images in the folder, named **image-0001.jpg**, **image-0002.jpg** and so on until **image-0721.jpg**.

You can now compile these individual images into a video. First, save them all in a text file:

```
$ cd ~/images
$ ls *.jpg > allimages.txt
```

Then install the *Mencoder* transcoding tool with:

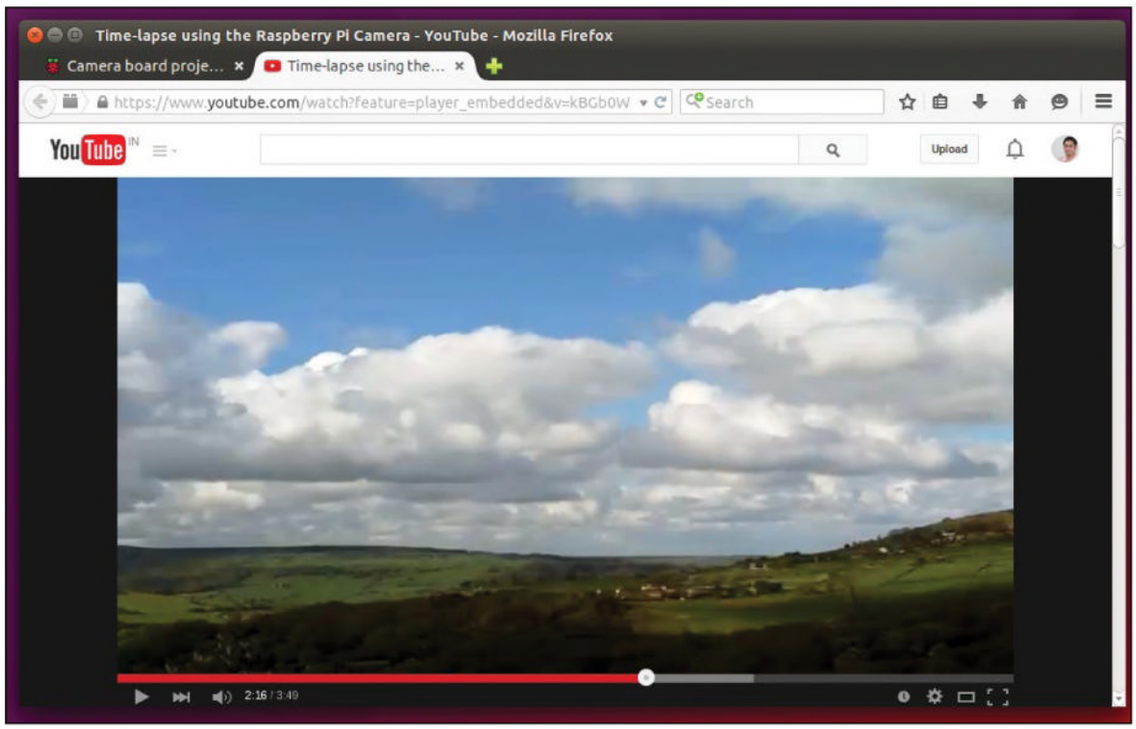
```
$ sudo apt-get install mencoder
```

Once it's installed, compile the video with:

```
$ mencoder -ovc lavc -lavcopts vcodec=mpeg4:aspect=16/9:v
bitrate=8100000 -vf scale=1920:1080 -nosound  -o timelapse.
avi -mf type=jpeg:fps=24 mf://@allimages.txt
```

That's quite a console-full. Let's break the command down into digestible bits. We ask *Mencoder* to create a video using the MPEG4 codec from the open source libavcodec library, and without any sound, because we haven't captured any. The **scale** specifies the Full HD (1,920x1,080) resolution of the video and the **aspect** specifies the widescreen aspect ratio of the video.

There's also a fair bit of maths involved. The bitrate is calculated using the formula (40 * 25 * width * height / 256) which comes to 8,100,000. Then we set the frame rate of the resulting video to 24fps. Given that an individual frame (or one image) represents 30 seconds of real time, the real time represented by one second of our time-lapsed video comes



If you want to string hundreds of images into a time-lapse video, you can capture them with the Pi and then transfer them on to a more powerful machine for processing.

Control the camera with Python

The Python PiCamera library helps you to control the Raspberry Pi camera module from within Python. You can install it on any distro using *Python PIP* package manager. First install *PIP* with **sudo apt-get install python-pip** and then install the library with **sudo pip install picamera**. You can now use the library to capture images. Here's a basic script that displays a preview for 10 seconds before capturing an image:

```
import picamera
from time import sleep

with picamera.PiCamera() as camera:
    camera.start_preview()
    sleep(10)
    camera.capture('/home/pi/image.jpg')
    camera.stop_preview()
```

You can also capture time lapse video using the **capture_continuos()** method of the PiCamera library. The PiCamera library includes a number of modules, such as picamera.encoders, picamera.streams, picamera.color and pricamera.exec, among others. Refer to the official documentation (**http://picamera.readthedocs.org/en/release-1.10/index.html**) for more details and examples on each.

to 24s * 30s, or 720s, which is equal to 12 minutes, or one fifth of an hour. In other words, five seconds of the time-lapsed video represents one hour of real time. This means the above command compresses the six-hour long capture into a 30-second video. Once you've wrapped your head around the maths, tweak the options for the *Raspistill* capture and the *Mencoder* encoding, until you get to your desired ratio between the real time and the video time.

Depending on the images it has to process, the *Mencoder* command can take some time to complete. When it's done, you'll have a file named **timelapse.avi** inside the current directory. You can copy it to another computer using the **scp** command. Assuming the IP address of the Raspberry Pi is **192.168.3.111**, head to another Linux computer on the network and enter:

```
$ scp pi@192.168.3.111:/home/pi/images/timelapse.avi ~/Videos
```

This command pulls the video from the Pi to the **Videos** directory on your regular distro. Once you've mastered the software aspect of the project, you'll have to work on the hardware bit. While shooting indoors doesn't pose much of an issue, shooting outdoors takes some planning. You'll have to figure out how to power the Raspberry Pi and to shield it from the elements, because you can't possibly keep an eye on it for the entire duration of the capture. A portable battery and a sturdy metal canister with a hole cut out large enough to expose the lens of the camera module works for the kind of nature time-lapses we enjoy capturing.

Pi-powered stop-motion studio

If you have directorial ambitions and think you can one-up Philip Lord and Chris Miller, use the Pi and the camera module to build yourself a no-frills stop-motion studio.

Before you can get started, you'll have to get the camera module to present itself as a regular USB camera. Typically, Linux apps access the camera through the /dev/video0 device and use the Video for Linux or V4L driver system. Currently, there are no kernel level V4L drivers for the Pi camera module. However, there are user space drivers or UV4L drivers that will create the V4L-style device that can be used by the Linux video apps. To download and install the **UV4L** package for the Raspberry Pi camera module, add its repository with:

```
$ curl http://www.linux-projects.org/listing/uv4l_repo/lrkey.asc | sudo apt-key add -
```

```
$ sudo sh -c 'echo "deb http://www.linux-projects.org/listing/uv4l_repo/raspbian/ wheezy main" >> /etc/apt/sources.list'
```

Now you need to refresh the list of repos with:

```
$ sudo apt-get update
```

Then install the drivers with:

```
$ sudo apt-get install uv4l uv4l-raspicam uv4l-raspicam-extras
```

The **uv4l-raspicam-extras** package loads the driver at boot and also provides a service script for starting or stopping the driver at any time:

```
$ sudo service uv4l_raspicam restart
```

The service uses the **/etc/uv4l/uv4l-raspicam.conf** configuration file to get the default values for the driver.

The raspicam script creates a full-screen preview of the camera view whenever an app accesses the camera. While this comes handy when lining up an image for a shot, the full-screen preview obscures any app that you're trying to use on the screen. To disable this preview, edit its init script (**/etc/init.d/uv4l_raspicam**) and find the line that reads:

```
$UV4L -k --sched-rr --config-file=/etc/uv4l/uv4l-raspicam.conf --driver raspicam --driver-config-file=/etc/uv4l/uv4l-raspicam.conf
```

And append the **--nopreview yes** at the end, so that it now reads:

```
$UV4L -k --sched-rr --config-file=/etc/uv4l/uv4l-raspicam.conf --driver raspicam --driver-config-file=/etc/uv4l/uv4l-raspicam.conf --nopreview yes
```

Then save the file and restart the Pi. When you're back up again, install the award-winning Linux *Stopmotion* app with:

```
sudo apt-get install stopmotion
```

You are now all set to start directing your feature. Assemble the characters, build the scene and position the camera module to capture it. When you've set everything up, launch *Stopmotion* from under the Sound & Video menu and click the video camera icon to preview the scene. Before shooting the first frame, move the Number of Images slider to 1. This is the number of previous pictures that *Stopmotion* mixes together in the preview pane.

Now make any modifications you need and when you're ready to shoot, click on the camera icon to capture a frame and add it to your video timeline. Once you've captured a frame, *Stopmotion* uses the onion skin overlay in the preview window to display the last image, to help you make adjustments when rearranging the scene for the next frame. When you're all set, press the camera icon again to capture the second frame.

Repeat the process until you've captured all the frames. Use the File > Save option to save your work in the app's STO format at regular intervals. When you're done, head to File > Export > Video to save your animation in a viewable format. Remember to append the **.avi** extension to the name of your movie for *Stopmotion* to encode it in the proper format. You »

Useful CLI options

Both *Raspistill* and *Raspivid* have a few options to take interesting image and video captures. The **--sharpness**, **--contrast** and **--brightness** options can influence the output of the image. The camera also supports several exposure modes, such as auto, night and sports, that can be specified with **--exposure**. Similarly, white balance options, including sun, cloud and shade, can be specified with the **--awb** option.

Use the **--width** and **--height** options with *Raspistill* to control the dimensions of the captured image. You can tweak the compression level of the JPEG image by specifying a quality level between 0 and 100 along with the **--quality** option. The **--raw** option bundles the RAW Bayer data from the camera in the image as well. You can add EXIF data to the image by specifying it along with the **--exif** option. For example, **--exif GPS.GPSLongitude=40/1,7/1,32/100** sets the longitude to 40 degrees, 7 minutes, 32 seconds.

The **--width** and **--height** options can also be used with the *Raspivid* tool. It also uses the **--bitrate** option to control the quality of the video. To capture a Full HD video at 15 Mbits/s use **--bitrate 15000000**.

Another related option is **--framerate**, which can accept a value between 2 and 30. To pause video during a capture, first start it with the **--keypress** option. You can then press the Enter key to pause capturing and then resume by pressing the Enter key once again. Press the X key followed by the Enter key when you want to stop recording.

» can then transfer the video to another computer using **scp** as before. Because the video could be quite hefty and might take some time to transfer across, make good use of the time and try on that tuxedo for the Oscars.

Detect motion

Another useful application for the small form factor of the Pi and its camera module is surveillance. You can use the duo to discreetly keep an eye on a remote location. For this we'll use the wonderful *Motion* software that starts capturing video as soon as it detects motion in the area under surveillance. The software also includes interesting features such as the ability to monitor it from the local network.

However, the standard *Motion* package that's available in the Raspbian repositories doesn't yet support the Raspberry Pi camera. You instead have to use a different binary, called *Motion-mmal*, specially created for the Raspberry Pi by a community member.

Motion depends on a lot of libraries, so fetch them with:

```
$ sudo apt-get install -y libjpeg62 libjpeg62-dev
libavformat53 libavformat-dev libavcodec53 libavcodec-dev
libavutil51 libavutil-dev libc6-dev zlib1g-dev libmysqlclient18
libmysqlclient-dev libpq5 libpq-dev
```

Once they are all installed, download the modified version of *Motion* and extract it:

```
$ mkdir ~/motion-mmal
$ wget https://www.dropbox.com/s/xdfcxm5hu71s97d/
motion-mmal.tar.gz
$ tar zxvf motion-mmal.tar.gz
```

This extracts the *Motion* app as well as a configuration file. Before you can use it, open the configuration file in a text editor to make some changes:

```
$ nano ~/motion-mmal/motion-mmalcam.conf
```

The configuration file has lots of tweakable parameters and might seem intimidating. However, it's well documented and you only need to tweak a few settings at this stage.

To capture high resolution images, hunt for the **width** and **height** parameters and change their values to **1,280** and **720** respectively. Next change the **frame rate** parameter to **2** which is a reasonable value for a security camera. Two useful options are **pre_capture** and **post_capture**, which define the number of frames to capture before and after motion is detected. Set both of these to **2**. By default, *Motion* stores the captured images and videos under the home directory of the pi user. For better organisation, we'll instead keep this under the **~/motion-mmal** directory, which we'll specify next to the **target_dir** parameter.

Also make sure the **stream_localhost** option is set to off, so that you can watch the real-time feed from the camera on any computer on the local network. You may also want to set the **webcontrol_localhost** option to off, which enables you to tweak *Motion's* configuration from a remote computer. For added security, if you do enable the above option, also enable the **webcontrol_authentication** option and define a username and password for accessing the configuration options in the format specified in the file.

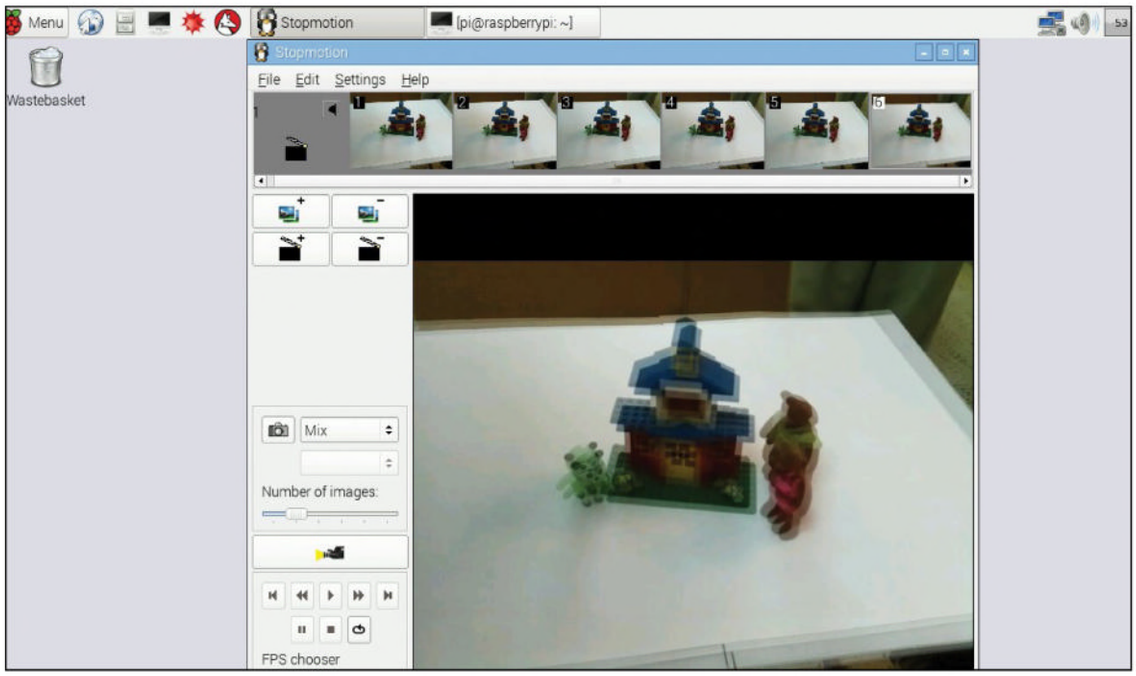
You can now run the extracted *Motion* app along with its configuration file with:

```
$ ./motion -n -c motion-mmalcam.conf
```

While *Motion* is running, you can watch the streaming video on another computer in the network over the 8081 port. So assuming the IP address of your Raspberry Pi running *Motion* is **192.168.3.111**, you can watch the video stream by entering the address **192.168.3.111:8081** on another computer in your network.

You'll have to revisit *Motion's* configuration file and tweak parameters depending on the environment you are monitoring, One of the most important sections you'll probably have to tweak is the Motion Detection Settings section. Here you can make changes to the various thresholds that help detect motion.

Again, just like the time-lapse project, the software part of the surveillance camera isn't as involved as the hardware aspect. Depending on how and where you plan to use it, you'll have to figure out solutions for its power and positioning requirements. Once again, just like the time-lapse video solution, you can use the same kind of hardware setup with a sturdy canister and portable battery.



If you've got young kids, use the *Stopmotion* app to make their favourite toys come alive.

Python GPIO: Control the traffic

How to direct traffic and enable pedestrians to cross the road safely using a Raspberry Pi-powered set of traffic lights.

There are many systems that can be modelled using software – nuclear reactors, weather patterns and the simplest of them all, the humble traffic light. The traffic light is a real physical system that can be easily simulated using a Raspberry Pi and Python code.

For this project, you will need any model of Raspberry Pi, in our case we used the Raspberry Pi 2, you'll also need a breadboard, 3 LEDs (green, red, yellow), three 220 Ohm resistors, one momentary switch, a buzzer and two packs Jumper Jerkys (one female to male and vice versa). We used the excellent CamJam EduKit, which retails for around £5 and can be bought from The Pi Hut:
http://thepihut.com/collections/camjam-edukit

We shall start by setting up our Raspberry Pi ready for use. You will need to wire up your Raspberry Pi to the components, using a breadboard. Please refer to **http://bit.ly/LXF198Beginner** for a high-resolution image of the layout.

When you're ready, boot up to the desktop. For this project, we need to use the Python 3 application, which is found in the Programming menu, but because we will be using the GPIO we need to open it in a special way, using the terminal and running the command as **sudo**. If you look to the top-left of the screen, you will see a black monitor icon; this is *LXTerminal*. Left-click on it once and a terminal window will open. In the terminal, enter the following, remembering to press Enter at the end.

```
sudo idle3
```

This will open the Python 3 editor, which is known as IDLE (Integrated DeveLopment Environment), where we will be coding our project. Before you start coding, first work through the logic, using Pseudocode:

```
Start with the green LED on, this allows "traffic" to flow freely.
Wait for the user to press the button.
Button pressed
    Turn off green LED
    Turn on the yellow LED
    Wait 2 seconds
    Turn off the yellow LED
    Turn on the red LED
    10 Times
    Turn on Green LED and buzzer
        Wait 0.2 seconds
        Turn off Green LED and buzzer
Turn on yellow LED
Wait 2 seconds
Turn off red and yellow LED
Go back to start.
```

So, let's get coding. In IDLE, go to File > New to open a new editor window. The first few lines of code that we'll use will import the libraries that'll help you use the GPIO pins on the Raspberry Pi and control the delays in the light sequence.

```
import RPi.GPIO as GPIO
from time import sleep
```

Next, we configure the GPIO, first by asking it not to be so verbose with its error messages, and second, to instruct Python that you'll be using the Broadcom pin references.

```
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
```

To simplify the handling of the various GPIO pins referenced, we shall use five variables that will be named references to their relevant GPIO pin.

```
red = 14
amber = 15
green = 18
button = 23
buzzer = 25
```

Now you know which pins are being used for your LEDs, buzzer and button, you need to instruct Python on how to handle each one. Note: The LEDs and buzzer will be an output and your button will be an input.

```
GPIO.setup(red, GPIO.OUT)
GPIO.setup(amber, GPIO.OUT)
GPIO.setup(green, GPIO.OUT)
GPIO.setup(buzzer, GPIO.OUT)
GPIO.setup(button, GPIO.IN, pull_up_down=GPIO.PUD_UP)
```

Traffic light controls

Now we move on to the main body of code, which controls the sequence used for the traffic lights. Start with **try**, which is part of the **try...except** error handling construction. Inside of this, we add an infinite loop, **while True**. Next, turn on the green LED and print a message to the Python shell before halting the project for 0.2 seconds:

```
try:
    while True:
        GPIO.output(green,1)
        print("Green Lights Cars In Motion")
        sleep(0.2)
```

Our next section of code handles the press of the button, and it's quite a large clunk of code, so we shall split it into sections. First, if the button is pressed, it will return as **False** due to it registering as **True** when the code starts. This is made possible during the **GPIO.setup** part of the code. We set the input pin to **High (True)** so when the button is pressed it changes to **Low (False)**, and this triggers the green

Quick tip

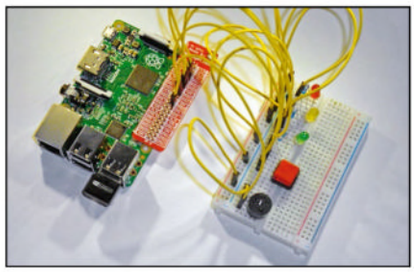
The Raspberry Pi GPIO has two pin layout references: BCM and Board. The function GPIO.setmode() informs the Pi which layout you are using for your project, so you need to be consistent.

The GPIO

What makes the Raspberry Pi such an expansive platform for invention is the series of pins that make up the GPIO – General Purpose Input Outputs. On the original Raspberry Pi, there were only 26 pins available for use, but with the release of the B+ in mid 2014 the GPIO was extended to 40 pins, and this has now become the default for all Raspberry Pi models.

The GPIO is used to connect electronic components such as LEDs (light emitting diodes) resistors and buzzers, which can then be controlled using Scratch GPIO, or in our case Python. The GPIO can also be used to interface with I2C (Inter-Integrated Circuit) devices, such as Pimoroni's upcoming Flotilla project. I2C uses only four wires to control multiple devices attached to it. The GPIO also supports SPI, a short-range synchronous serial interface that is used commonly with SD cards.

Many projects that involve the GPIO will require you to make use of add on "HATs" short for Hardware Attached on Top, or breadboards. We do recommend you buy a breadboard if you're planning on dabbling with any hardware projects. These are standard electronic accessories ideal for prototype circuits and more. They enable you to easily and temporarily connect together wires and components with a interconnected grid. They're cheap and available from all good electronic stores.



Connecting wires to the GPIO can be done while the Raspberry Pi is on; just make sure that your connections are correct, as a short may cause your Pi to reboot.

LED to turn off, followed by the yellow LED to turn on for two seconds and turn itself off, and finally the red LED turns itself on and stops traffic.

```
    if GPIO.input(button) == False:
        GPIO.output(green,0)
        GPIO.output(amber,1)
        sleep(2)
        GPIO.output(amber,0)
        GPIO.output(red,1)
```

Looping the LEDs

The second section of code uses a **for** loop to go round 10 times, and this handles flashing the green LED and beeping the buzzer by turning them on and off rapidly. After this, we turn on the red and yellow LED, sleep for two seconds and then turn them off. The loop then repeats itself looking for the user to press the crossing button. If that doesn't happen, the green LED will stay lit so that traffic can pass through freely.

```
        for i in range(10):
            print(" W A L K ")
            GPIO.output(green,1)
            GPIO.output(buzzer,1)
            sleep(0.2)
            GPIO.output(green,0)
            GPIO.output(buzzer,0)
            sleep(0.2)
        GPIO.output(red,1)
        GPIO.output(amber,1)
        sleep(2)
        GPIO.output(amber,0)
        GPIO.output(red,0)
```

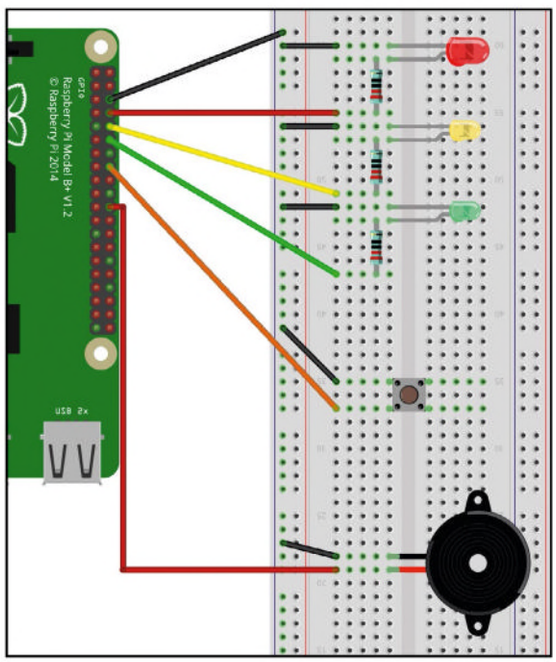
Last, we close the **try...except** structure so that it will accept a **KeyboardInterrupt**. Commonly this is done by pressing Ctrl+C together to stop a running program. Once this event occurs, the program will reset all of the GPIO pins back to their defaults and then exit the code cleanly.

```
except KeyboardInterrupt:
    GPIO.cleanup()
```

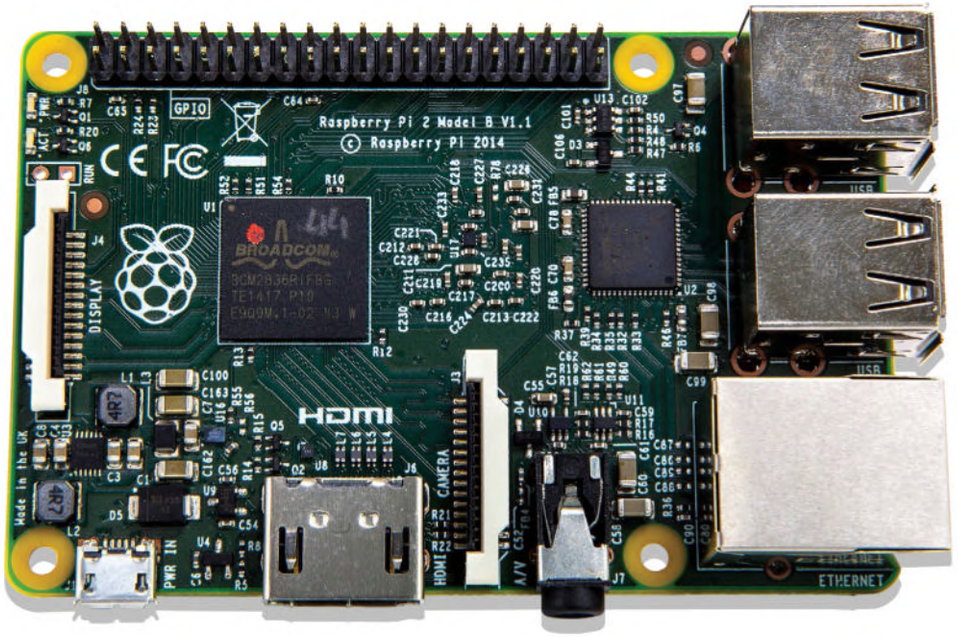
With the code written, you will now need to save your work, and when ready click on Run > Run Module. Wait a few seconds and the green LED should be illuminated. Now press the button and the LED should change to yellow and then red. The green LED will start to flash and the buzzer will direct you to cross the road.

Congratulations, you have simulated your first physical system using a small amount of Python code, some electronics and the Raspberry Pi.

In this project, we have used the CamJam EduKit #1 to provide us with the components necessary. At the time of writing, there are two kits on offer. EduKit #1 is the one we used, and comes with basic components to enable many different input/output-based projects, and while EduKit #2 was released in late 2014, it expands the basic kit to include sensors, such as PIR (Passive Infra Red) and temperature. Both of these kits come with a series of worksheets that will soon fall under the stewardship of the Raspberry Pi Foundation's education team. Both of these kits can be purchased from **http://thepihut.com/collections/camjam-edukit**, with Kit #1 retailing for only £5 and Kit #2 costing a little more at £7 – well worth the price.



This layout was created using an application called Fritzing (http://fritzing.org/home). This is a great piece of free software that will help you to plan your projects and share them with ease.



Hands-on with the Raspberry Pi 2

We had early access to the first batch of Raspberry Pi 2s and takes one for a test drive.

The new Raspberry Pi comes with much more power than its predecessors. This is thanks to the improved ARMv7 processor running four cores at 800MHz each and the generous 1GB of RAM. This increase in both CPU and RAM is a massive benefit to projects that require pure CPU grunt, such as OpenCV and *Minecraft*.

The Raspberry Pi 2 also benefits from the design improvements made for the B+ with more USB ports thanks to the LAN9514 providing four ports over the 9512's two. The B+ also introduced better power management and this, again, is also present in the Raspberry Pi 2. "Power consumption while performing a given task is comparable to that of B+," explains Eben Upton, CEO of Raspberry Pi Trading. "Obviously if you push Pi 2 hard it will consume more power as it's doing more work. Power consumption of B+ under heavy load is roughly the same as the old Model B."

So now that we have the latest Raspberry Pi, lets take it for a test drive! And for this extended tutorial we will be using the latest version of Raspbian available via the Raspberry Pi website (**www.raspberrypi.org/downloads**)as it comes with the kernel7.img necessary to use the ARM7 CPU.

The easiest method to setup your microSD card is to use *NOOBS* (New Out Of The Box Software) To use it you will need at least an 8GB microSD card. Download *NOOBS* as an archive from the Raspberry Pi website and extract the contents to your 8GB microSD card which should be formatted to use a FAT32 filesystem. With NOOBS copied to your microSD card, unmount and remove the card from your computer and insert it into your Raspberry Pi 2, you should hear a gentle click once it is firmly in place.

> **"The Raspberry Pi 2 also benefits from the design improvements made for the B+."**

Connect up your Raspberry Pi to a monitor via the HDMI port and then attach a keyboard and mouse via the USB ports. You will also need to ensure that you have Internet access for your Pi. The easiest method is to use the Ethernet port. Finally, attach the power supply to the micro USB port. Your Raspberry Pi 2 will now boot for the first time.

On first boot *NOOBS* will ask you which OS to install, in this case we require Raspbian. Select the OS and start the installation, which will take around 10 minutes.

With the install complete the Pi will reboot and start Raspbian for the first time, and the first thing that you will notice is how quick the boot process is now at 17 seconds versus 33 seconds for the B+! For the observant you'll also note that there are now four raspberries at boot up. This denotes that the Raspberry Pi 2 has four cores, a rather nice little Easter egg that harks back to the old boot screens of Linux distributions. Once booted we are immediately presented with the *raspi-config* menu, this is a tool to further configure your Raspberry Pi. At this stage we will simply exit out of the menu and login as normal. The standard login details have not been changed and remain as:

USERNAME: pi
PASSWORD: raspberry

Once logged in you need to type:

```
$ startx
```

to load the desktop. You'll see that the desktop is a little different to previous versions of Raspbian, this is due to extensive changes that were made by the Foundation back in December of 2014 and is largely the work of Simon Long, who used to work for Broadcom. In fact, it was Long who hired Eben Upton at Broadcom, but now Simon Long has joined the Raspberry Pi Foundation to work on the user interface and his first project was to develop a new desktop.

The Raspberry Pi Foundation have created a very powerful single board computer, but how can we test that power? It is somewhat fitting that we can calculate Pi to as many decimal places as we wish, so lets say 10,000? To do that we need to install some software, first. Open LXterminal and type the following two lines:

```
$ sudo apt-get update
$ sudo apt-get install bc
```

We've just installed a precision calculator that we can use from the terminal. So now to the test, calculating Pi to 10,000 places and timing the activity.

```
$ time echo "scale=10000; 4*a(1)" | bc -l
```

In our test it took 17 minutes 25.725s to calculate Pi to ten thousand decimal places on our stock Raspberry Pi 2. We repeated the same Pi test on a Raspberry Pi B+ and it took significantly longer at 25 minutes 5.989 seconds to perform the calculation. As you can see straight away that's a very clear indication that the processor of the new Raspberry Pi 2 is easily much more powerful than previous models we've seen.

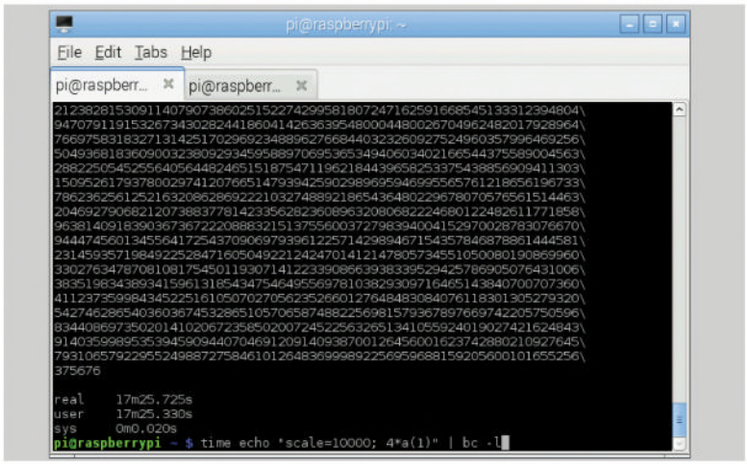
Our quick test, gives us a good idea of how much of a serious performance beast the Raspberry Pi 2 is out of the box, but can we tweak it into more of an animal? Well, earlier we dismissed the *raspi-config* menu but for this next step we need it again. In *LXTerminal*, and type the following:

```
sudo raspi-config
```

Turbo charge your Pi

Our first post install configuration is to review the memory split. This is how the memory is divided between the GPU (Graphical Processing Unit) and the main system. On the Raspberry Pi a typical setup would be around 64MB of RAM for the GPU and the remaining RAM would be allocated to the system. This is an optional config tweak but you could just leave it as is. You can easily tinker with these values, and a general rule is that a terminal will not need as much RAM as the full desktop, so for a terminal-only project you can easily get away with 6MB allocated to the GPU. For desktop applications such as *Minecraft* a minimum of 64MB is needed. You will be prompted to reboot your Raspberry Pi, do this and you will be returned to the login prompt.

With the changes made to the memory split now let us return to the *raspi-config* main menu and head to the Overclock menu. Your Raspberry Pi 2 already runs at 800MHz per core, which is an improvement over the original 700MHz single core ARM 11 CPU. We spoke to Eben Upton and Gordon Hollingsworth about the new CPU and they both confirmed that it can be overclocked to around 1.1GHz per core. We won't be going quite that high, but we will overclock our Raspberry Pi to a stable 900MHz using the Overclock menu. While this is a relatively safe activity it should be noted that going too far with overclocking can severely damage the CPU due to the excessive heat generated by it working harder.



Computing Pi to 10,000 places is a fitting test for our Raspberry Pi. The command to run the test is run from LXTerminal. Here we show the time it took before we overclocked.

We asked the Raspberry Pi team and it confirmed that the core can reach temperatures of 85 degrees, and when it does it will automatically turn off the Raspberry Pi for protection. For 'extreme' tinkerers who want to push their Raspberry Pi 2 to the limit, now might be the time to invest in a set of heat sinks. If at anytime you wish to return the CPU to it's normal speed, re-enter the raspi-config menu and return the values to the stock 800MHz.

With our configuration changes made, and after a few reboots we have successfully 'turbo charged' our new Raspberry Pi. Let's start the graphical user interface. After logging back in use the *LXTerminal* to type

```
$ startx
```

to return to the desktop. Now, lets see how the changes have improved our Pi to 10,000 score by repeating the test. Open LXTerminal and repeat the test code which was

```
$ time echo "scale=10000; 4*a(1)" | bc -l
```

The code will run and in our test it took 15 minutes 28.519 seconds – that's an improvement of two minutes!

The Raspberry Pi Foundation has taken great care to build upon the legacy created by the Raspberry Pi Classic: "Raspberry Pi 2 has been in development for a couple of years," say Upton and Hollingsworth, and that includes the time spent developing BCM2836. "The first silicon arrived at the start of last May; there will be a video on the blog of me, James and Dom in the lab at Broadcom at 1am, the day after silicon came back, with the 'video on a teapot' demo running from Linux on a Broadcom "Ray" dev board. The design of the Rasberry Pi 2 board started last August (2014), and we've had samples since October (2014). We went through three iterations of prototypes to get exactly the right performance," says Upton.

Compatibility

The performance is reflected in the choice of CPU for the Raspberry Pi 2. Rather than choose another architecture the Foundation has stuck with an ARM-based CPU that is compatible with the ARM11 found in the earlier Raspberry Pi. The quad-core ARM7 can run software written for the older Raspberry Pi: "Raspbian works out of the box, but requires a new v7 kernel, which will be included in the downloads from our website" said Eben.

Quick tip

The default web browser for Raspbian, *Midori* has recently been replaced with *Epiphany* which has been optimised for use on the Raspberry Pi. The new browser is available via the latest Raspbian update and works really well on Raspberry Pi 2 and older Pis.

Quick tip

The Raspberry Pi 2 shares the same dimensions as the B+ but for those of you looking to reuse a B+ case, such as the Pibow, it's worth noting that some surface mount components have been moved. These changes don't affect the overall size of the board but as the Pibow uses layers to build, a new layer will be required for your Pibow.

» As for hardware compatibility, the Raspberry Pi 2 shares the same GPIO as the B+, which means that boards made for the A+ and B+ will also work with the Raspberry Pi 2 and this even includes HAT boards (Hardware Attached on Top), which contain an onboard chip that communicates with the Raspberry Pi to set up the board quickly.

There are some boards, however, that are not compatible with the B+ and the Raspberry Pi 2 due to their size and design. Boards such as PiFace [*see* **LXF180**] and PiFace Control and Display – that was used to control a camera rig for the Royal Institution Christmas Lectures – cannot be attached. But the OpenLX SP team behind these boards has released special versions for the B+ and Raspberry Pi 2.

Quick tip

Watching YouTube videos is now possible thanks to Youtube_dl. Normally YouTube videos are Flash based but there are no Flash packages for Raspbian. When watching a YouTube video in the web browser, Youtube_dl substitutes the Flash element of the web page with an HTML5 compliant video.

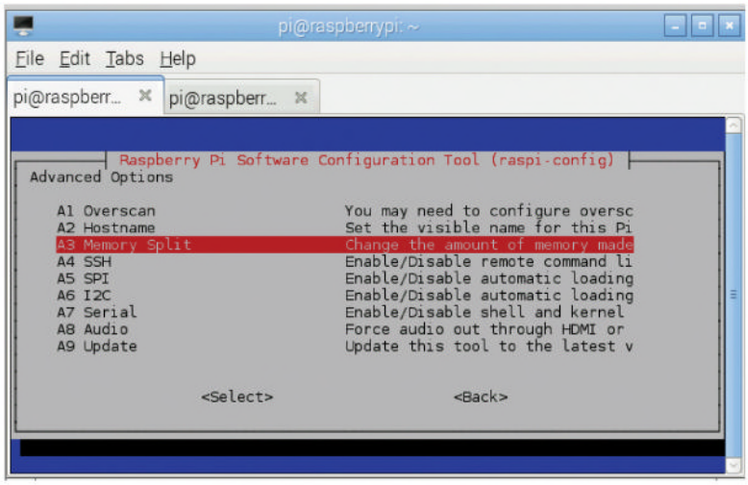
3D graphics test

Every Raspberry Pi comes with the same VideoCore IV GPU (Graphical Processing Unit) that enables the Raspberry Pi to play back high-definition video at 1080p. The new PI also comes with this GPU, also made by Broadcom just like the BCM2836 powering the new Pi. Did you know that there's a test suite for the GPU?

You can find the test suite by opening *LXTerminal* and typing the following:

```
cd /opt/vc/src/hello_pi/
```

In there you will find a number of directories containing many different demos. But before we can use them we need to build the demos from source, and to make this easier the Foundation have provided an automated build script that will only need to run once. To run the script, in *LXTerminal* type



The raspi-config advanced menu contains configuration options that enable you to really make the Pi your own.

The Hello_Pi demos are a great way to show off your Raspberry Pi 2. You can wrap video around the teapot using any H.264 compliant video.

```
./rebuild.sh
```

This build script will run the build process for all of the demos so it may take a few minutes, even on our new speedy Raspberry Pi.

Once completed there are a number of demos that you can try out and the first on the list should be hello_teapot. To run it, in *LXTerminal* make sure that you are still in the **hello_pi** directory and type:

```
cd hello_teapot
./hello_teapot.bin
```

You will now see a 3D render of a teapot with video that's been directly rendered on to its surface. To exit out of the teapot demo hold Control+C together and you will be returned to *LXTerminal*.

Another demo to try is *hello_triangle2* and to getto that you will need to go back to the **hello_pi** directory and we can do that by typing.

```
cd ..
```

From **hello_pi** we can change our directory to **hello_triangle2** and run the demo by typing

```
cd hello_triangle2
./hello_triangle2
```

This demo appears to be rather static at first, but try moving the mouse around and you will see two fractals superimposed one over the other moving and reacting to your mouse movements. Apparently, you can also control the fractals to create a perfect circle. To exit out of the *hello_triangel2* demo hold Control+ C together and you'll be returned to *LXTerminal*

So we've taken a look around the new Raspberry Pi 2. On the next page we will interface *Minecraft* with Pibrella to create a push-button bomb deployment system!

The making of Raspberry Pi 2

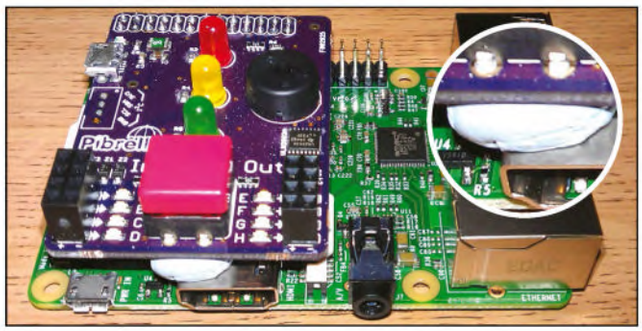
The Raspberry Pi Foundation are rather excited about the new Raspberry Pi 2, We spoke to the engineering team and Gordon Hollingsworth about development: "The Raspberry Pi 2 is 100% awesome. It's as close to a typical PC as we wanted when we first started the project." And the amount of effort that's gone into development is impressive: "The team have put the equivalent of 20 years of work into the new Raspberry Pi and its processor, and that runs at a cost of between £2-3 million." But there's still a lot of enthusiasm for the older Raspberry Pi, Eben Upton explains: "There are a lot of industrial customers who won't want to switch, and, of course, we still have the Model A+. To give you an idea of the 'stickiness' of an old platform, we sold something like 80,000 Model Bs after the Model B+ launch."

The Foundation also has its Compute Module, which was created to embed the Raspberry Pi inside industrial applications. We asked Eben if the Compute would be seeing a similar upgrade or not: "We'll do a Compute Module 2 at some point, but probably not in the first half of 2015." And what of the A+? Would there be a similar upgrade for it: "Nothing is currently planned as the A+ price point is quite challenging." No upgrade for now then, but the Raspberry Pi family has grown considerably since 2012 and now features six devices.

Link a Pibrella to Minecraft

1 Attach Pibrella to your Raspberry Pi

We'll use the big red button on the Pibrella.com to set off TNT in *Minecraft*. The Pibrella fits over the first 26 GPIO pins. Never attach while the power is on! Use a blob of blu tack or modelling clay to prevent the underside shorting out the HDMI. Now connect the other cables as usual, but insert the power into the micro USB port.

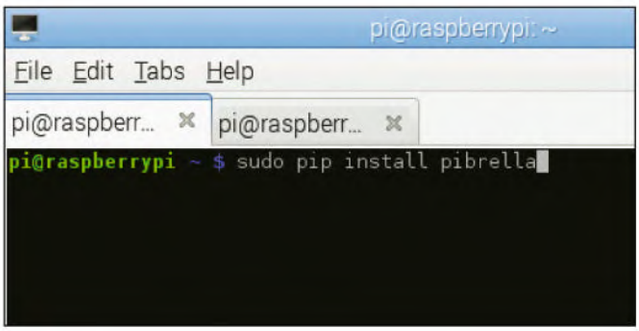


2 Setup Pibrella

With our Pi booted to the desktop, open *LXTerminal* and type:

```
sudo apt-get update
sudo apt-get install python-pip
sudo pip install pibrella
```

This installs the software that we need to use Pibrella with Python.

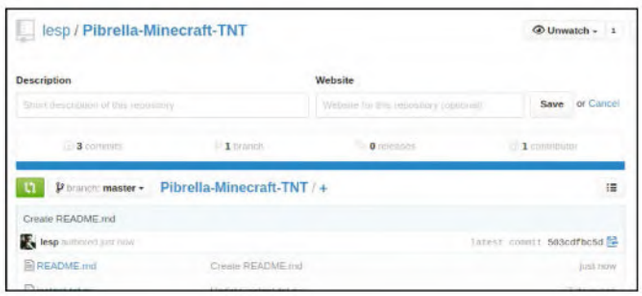


3 Get the code

We've created a GitHub repository that contains the code for this tutorial, visit **https://github.com/lesp/Pibrella-Minecraft-TNT** and download a copy. Next open *LXTerminal* and type

```
sudo idle
```

This opens idle, a Python editor, with elevated privileges enabling us to use Pibrella with Python. Now open the example code.



4 Examining the code

Our code is written in Python 2 as the *Minecraft* module is currently only available for that version. The code is fairly easy to read. Note the line starting with # are comments. The first few lines are imports, they import extra functionality, in the form of Pibrella and *Minecraft* libraries, for our project. After that we use a variable called **mc** to store connection details to *Minecraft*.

```
import mcpi.minecraft as minecraft
#We import the Minecraft module so

mc = minecraft.Minecraft.create()
#We create a connection between Py

#We define a function and call it
def button_changed(pin):
```

5 In position

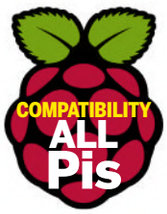
Minecraft uses x,y,z coordinates to know the position of objects in the world. We create a function called **button_changed()**, which locates the player and then creates a cube of TNT at coordinates near to the player. Lastly we set the function to be called when the button is pressed. Keep the window open and open *Minecraft* and create a new world.

6 Drop the bomb

With Minecraft ready and our code open, press TAB to release the mouse from *Minecraft* and click on Run > Run Module in idle. The idle shell will come to life and run the code. Switch back to *Minecraft* and go to a nice spot. Press the red Pibrella button to drop the bomb. Hit the TNT with your sword... and then RUN! Note: You can run this the original Pi, but it could crash *Minecraft*.

Motion: Detect and record

Build a livestreaming system, using a Raspberry Pi and a webcam and how to save motion-detected video.



We'll assume you have none of the required packages to follow this tutorial on video surveillance and video recording. You will use *Motion* which is the heart of this article. Aside from that, you will require *Apache* (or *Nginx*) and PHP. Although this tutorial is geared towards using a Raspberry Pi, you can use another PC setup if you prefer. Do note, that if you go the *Apache* and PHP route, everything will work very easily without having to make extra changes to the server and PHP.

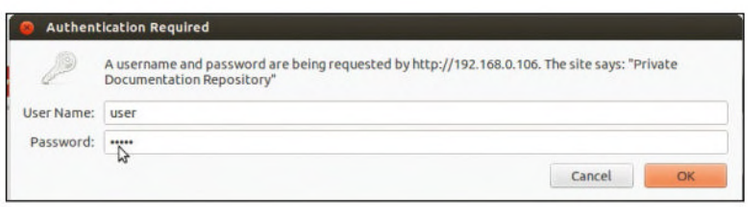
If you do decide to go with Nginx instead of *Apache* you will need to make some extra changes: such as installing *PHP-FPM*; changing the root folder path for web page files; and editing the following files: **/etc/nginx/sites-available/default**, **/etc/nginx/sites-enabled/default** and **/etc/php5/fpm/php.ini**.

Now, for the synopsis of each package. *Motion* will be used to record video after movement is triggered. The video clips will be written to a folder as Flash SWF files. However, *Motion* still allows you to see the location even without movement, much like a regular security camera.

Once you have those files, you may want to be able to sort through them effectively. Here is where the web server and PHP play their role. With the *Apache* or *Nginx* server, you can serve these files over the web.

Realistically, many files will be accumulated and you may want to create a loop with PHP in order to output each file into a link that can display the video in a popup. In which case a free video popup application, such as *Shadowbox* can be used. Lucky for you, the code can be grabbed from **http://bit.ly/LXFmotioncode** this contains all the files needed.

With all that covered, you'll have a setup that can deliver your videos. This tutorial will show you various options and their how-to counterparts. Since a camera like this could be used in your home as a security camera, you may want to password protect any web pages or the folder where you keep the videos. Now, if someone did happen to break into your premises and decide to steal or wreck your Raspberry Pi, we'll also guide you through a backup plan that can be used to move your video files to a foreign web server that the robber won't have a clue exists.



Use simple password protected authentication to keep files secret.

Getting things to work

Since this article is about *Motion*, let's install this first:

```
sudo apt-get update
sudo apt-get install motion
```

Now that one installation is out of the way, let's add the rest, which includes *Apache*

```
sudo apt-get install apache2
```

and PHP:

```
sudo apt-get install php5 libapache2-mod-php5 php5-mcrypt
```

Let's move on and make some basic procedures and tests to see everything is working as it should. The main files which you will customise are **/etc/motion/motion.conf** and **/etc/default/motion**. Open up **motion.conf** with your favourite editor. By default, you'll note that the parameters shown below are the opposite of the default values. For example, daemon off becomes daemon on:

```
daemon on
webcam_localhost off
control_localhost off
```

Save the changes and open up the **/etc/default/motion** file and make the following changes:

```
start_motion_daemon=yes
```

Now, let's fine tune some options. Three changes that are needed are: the frame rate, quality and minimum amount of frames to trigger the motion to record:

```
framerate 30
quality 90
minimum_motion_frames 5
```

Without changing this setting, two frames per second looks way too jerky and will miss out a lot of action, so we change the frame rate from 2 to 30 frames per second. The second change is obvious since it's a quality upgrade. The third change sets the minimum amount of frames of motion that need to be detected. By default, the value is 1. The problem with a number this low is that you can end up with unwanted recordings from things such as lights flicking. Keep in mind that you have many options and can look deeper into the features. A good place to start is on the offical website (**http://bit.ly/MotionConfigFileOptions**).

Some of the other features you might want to consider are: taking a picture at a desired interval, such as every second, every minute or every hour. This feature makes it easy to host a live weather cam, for instance, or to determine if someone is sitting on your couch.

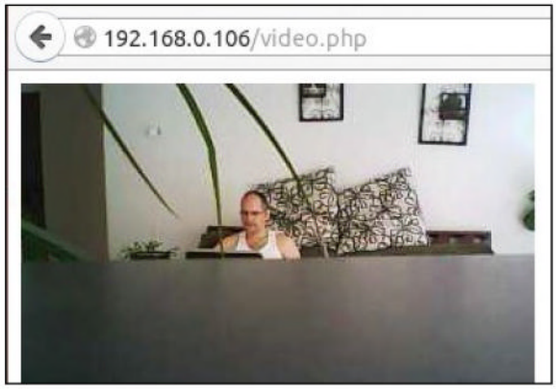
Configuring Motion

Changing all parameters to suit your specific needs is very easy and the **motion.conf** file will often have nice, self-explanatory comments while the website and man page have more information to offer. Obviously, this service doesn't do much without a working, compatible webcam and a list of webcams worth trying with the Raspberry Pi can be found at **http://elinux.org/RPi_USB_Webcams**.

Using a plug and play webcam makes life easy for this task, and one cheap, readily available webcam that works is the Logitech C170. Note: If you are using the Raspberry Pi, the Raspberry Pi cam won't work with *Motion*. To tell if the USB webcam connects OK, run the command **lsusb**.

At this point, you will likely have a working webcam, a working web server and an adequate *Motion* configuration. This is good, but you'll also need to create a folder for the images and set ownership for *Motion*. By default, *Motion* drops the images and SWF files into the **/tmp/motion folder**. It won't create the folder, therefore, you will need to:

```
cd /tmp
mkdir motion
```



Alert! Man on sofa. Capturing live video feed.

```
chown motion:motion motion
```

Now, let's see how everything works. To start with, you can get *Motion* up and running with the command

```
service motion start
```

and you can always restart it with the command

```
service motion restart
```

The first thing you'll need to do to test that everything works fine is to see if you can view the default web page. Since your Pi will have a unique network address, you can just type it in the browser. For example, if the Pi is connected to a router with the IP of **192.168.0.1**, your device could have an IP like **192.168.0.106**. Thus, the URL would also be **http://192.168.0.106**. If you have success returning the default web page, you will see a message stating that everything is working properly. If not, you will get a typical browser error which makes it obvious that something is not quite right.

Now with a working server, let's move on and get down to viewing and recording video. You can test the video in your browser by typing your network IP and port. By default, the *Motion* webcam port will be 8081. Therefore, if you type **http://192.168.0.106:8081** in the browser, you should see your video stream.

A simple setup like this can be beneficial and have many uses aside from security: such as keeping an eye on a new-born while you're working in another room. Since all should be well at this point with the *Motion* service running, you can now go in front of your webcam and jump around a bit. Actually, hand waving will suffice but a few jumping jacks aren't going to hurt. After that, you should be able to browse the new **motion** folder and see a bunch of JPEG files and at least one SWF file.

Quick tip

When you're logged in via SSH and need to edit files, using **vim** to find a string is easy. To do this, all you need is a **/** followed by the string name. Just type **n** to move on to the next one.

Using multiple webcams

Need more than one webcam? No problem. *Motion* enables you to add more easily. You need to open up **/etc/motion/motion.conf** and set up the threads. If you go to the bottom of the file, you see various lines that are commented out followed by the word thread. As you can see, the default location for these new files are in **/usr/local/etc** folder.

To keep it simple, you can change the thread folder to the **/etc/motion** folder. This way, you keep all the editing in one simple location. Now, the first thread would resemble the line below:

```
thread /etc/motion/thread1.conf
```

Once you've set up your threads since you are using multiple webcams, you can create the thread files. Thus, your first thread would be called **thread1.conf**, followed by **thread2.conf**, and so on. The code that you add to these threads just needs to be a few lines. The code samples below display two threads. As you can see, each thread has its own videodevice parameter, custom text that appears on the left-hand side of the video stream and image folder and port number. Here' s **thread1.conf**

```
videodevice /dev/video0
text_left Camera #1
target_dir /var/www/images
webcam_port 8081
```

followed by **thread2.conf**:

```
videodevice /dev/video1
text_left camera #2
target_dir /var/www/images_cam2
webcam_port 8082
```

» Now, that you have a motion-detecting device that you can view from your local network, you can move on and make adjustments so that you can see and view your webcam from outside your local network. For some of you, this may be the setup you desire; especially if your webcam and Raspberry Pi are well hidden and unlikely to be tampered with.

However, in addition to storing data in your own home, we will explain how to back up the files to another server for safe keeping, just in case your SD card or hard drive fails, or someone decides to steal, break or ruin your webcam (or webcams) in your home.

Making it web friendly

The whole idea here is to record, store and manage video from a website or by using your IP address that was given to you from your ISP. To find out your IP head to **http://whatismyipaddress.com**. In order to broadcast video out, you'll need to set up port forwarding on your router to enable your network IP to use port 8081.

While you are at it, you may as well do the same for port 80 since this same network IP will be used to display web pages from computers outside your local network; such as your friend across town or a loved one overseas.

After you have made the previous changes to your router's settings, try typing **http://my_ipaddress_from_isp** and **http://my_ipaddress_from_isp:8081**. You should get the same results as you did when you were checking your local IP.

The next step is to clean this up and view organised data view web URLs; such as **http://my_ipaddress_from_isp/video.php** or to view it from another website using an iframe.

In order to show the webcam from the page **video.php**, you just need to use an img tag with the network IP and port. Have a look at the code below, which shows this in complete detail and displays it with the default width and height as specified in the **motion.conf** file:

```
<img src="http://192.168.0.106:8081/" width="320"
height="240"/>
```

Now, let's imagine a scenario where you want to stream this livecam from another website using an iframe. Well, all you have to do is make an iframe from another page on the different server. The simple one-liner is shown below.

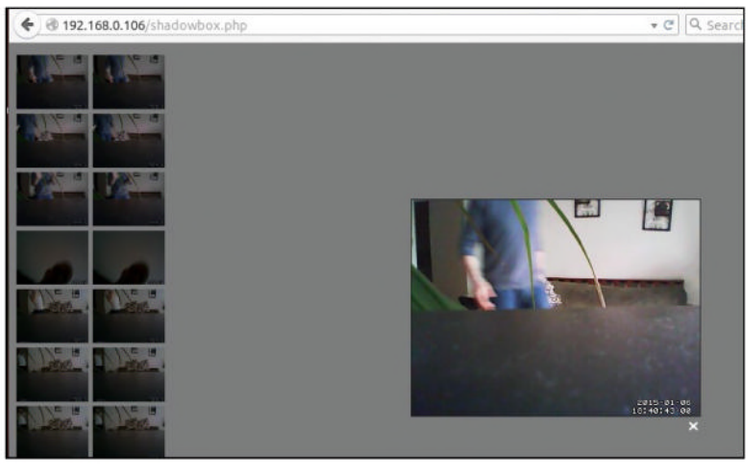
```
<iframe style="width:320px; height:240px;" src="http://isp_
ipaddress/video.php"></iframe>
```

The next set of code will explain how to display the files that have been saved after motion is detected and recorded.

With that said, we will move on and create a simple way to organise the saved files that recorded the movement. The first detail that will need to be changed is to save the images and SWF files into a folder within the web directory. The root web folder is located at **/var/www/html** or **/var/www**. At this point, a light should go off since you have already made several changes to the *Motion* setup. Reopen the **/etc/motion/motion.conf** file and change the target directory. By default, the target directory is located at **/tmp/motion**. The new target is **/var/www/images**:

```
target_dir /var/www/images
```

› **Displaying stored images and Flash video.**



Viewing recorded video and images

After making changes to **motion.conf**, type the command:

```
sudo service motion reload
```

so that it will now attempt to write any new files to the **/var/www/images folder**. Now, you can easily access the files created by the *Motion* service and display them on the web just like any other typical web page. Although the path has been changed in **motion.conf**, the **images** folder hasn't been created yet. So, make it now.

The folder will be located within the **www** or **html** folder. If it sounds like we're repeating ourselves here, because it means you have been paying attention and are aware that the *Apache* root **web** folder can be in one of two paths:

```
cd /var/www
mkdir images
```

By default, the **www** directory will be owned by the root user and root group. You will want to make some changes; such as all files will be owned by pi and the group will be **www-data**. To change this use:

```
cd /var
chown -R pi:www-data www
```

So, what we are up against now is to make this **images** folder writable by the *Motion* service. As of right now, the other files have adequate ownership and permissions, but, the images folder does not. Well, let's change that right now. The code snippet below has three commands. The first command will add the user motion to the **www-data** group. In case you are wondering, **www-data** is an existing user and group for the *Apache* server. The second command gives the images folder permissions to the *Motion* user and **www-data** group, and the final command makes the folder writable so that the images and SWF files can magically appear in the **images** folder:

```
usermod -a -G www-data motion
chown motion:www-data images
chmod 777 images
```

It's within the **www** folder where you can create a file called **shadowbox.php** that will be used to display content on the web. This file has all the code you need to display a thumbnail from each recorded SWF video, the video itself and the first JPEG image of motion.

The coding process to display the content goes like this: The images directory gets scanned and an array of files is created and sorted in descending order. These files are multiple JPEG files and a single SWF file for each event. All files have a name that starts with the event, followed by the date and finally followed by the sequence.

The date sequence is year, month, day, hour, minutes and seconds. After that, *Motion* just adds the sequence starting from 01 only for the JPEG files.

In order to keeps things simple and uncluttered, the coding for the **shadowbox.php** file will only display a single image and a single SWF file for each event. This simple script

also uses *shadowbox* to create popups of the first JPEG image and flash SWF file for each event. Now, you can see all the latest detected motion, down to the first recorded by *Motion*. This file gives all of the results and here is where you may want to customise the output.

If you want to keep these web pages password protected with *Apache*, you can open up the file **/etc/apache2/sites-available/default** and make some minor changes.

If you look for the line **<Directory /var/www/>**, you can add three simple lines to it. The code sample is shown below:

```
AuthType Basic
AuthName "Private Documentation Repository"
AuthUserFile /var/www/.htpasswd
Require valid-user
```

After that, you navigate to the **/var/www** folder and create a blank file called **.htpasswd**, and create the username and password with the simple command displayed below. You will be prompted for a password twice. You simply add it followed by Enter:

```
sudo htpasswd /var/www/.htpasswd add_username_here
```

Since the files can pile up pretty quickly and your disk space can readily disappear, you may want to create a purging system or back them up to another drive. Some backup plans are discussed next.

Backup plans

One tip for determining your backup plan is to watch how much space you routinely tend to use and develop a plan based on those conditions. Simple. For example, if you go through 1GB a week and you have a 8GB card you may want to TAR the images folders, SCP the file to a remote server and remove all files that are more than one-week old. Since the files contain the year, month and date, it's a rather easy process to delete the ones that have expired. The file called **purge.php** is a cleanup file that is included on the **http://bit.ly/LXFmotioncode** location.

This file removes every file that's more than a couple of days old. I will explain the code in a little more detail in a moment. First off, the **images** folder is scanned and all of the files become an array. That array of files then iterates through a foreach loop. A few built-in PHP functions, such as **strstr()**, **preg_replace()**, **substr_replace()**, **substr()**, **date()** and **unlink()** are used to translate all the file names into actual date timestamps that can be used for comparison.

Once a timestamp is made from the filename, it goes through a simple **if()** statement and is compared against a time that is set to two days ago from the current time. This part is really easy to change since you just need to change the number 2 to your desired amount of days in the past. Once this criteria is met, the file is deleted with the **unlink()** function. Since this system is only using files without a database, it's rather elementary to move all of these files to your backup location, and since this is copying and moving files, two methods come to mind. One is using a package such as *rsync* and the other is a simple method of compressing the desired files and folders with ZIP or TAR and shipping them to their new destination with SCP. An simple example of SCP is shown below:

```
scp -P 22 /var/www/images.tar pi@example.com:/home/pi/images.tar
```

So there we have it. You've just created your own video surveillance and motion recording system that has several options to suit your needs or that you can customise. Although we've made a rough skeleton and files for you to monitor your video, record files and make backups, you can take this farther if you want. Some simple suggestions would be are to add a responsive template to both the **video.php** and **shadowbox.php** files, and polish up the content with a little CSS magic.

On top of that, you could set up webcams at other sources and have them viewable by the public or friends, depending upon what you want to achieve. Have fun!

```
73002 -rw-r--r-- 1 motion motion     17211 Jan  6 18:40
73001 -rw-r--r-- 1 motion motion     16824 Jan  6 18:40
73000 -rw-r--r-- 1 motion motion     16939 Jan  6 18:40
72999 -rw-r--r-- 1 motion motion     17084 Jan  6 18:40
72998 -rw-r--r-- 1 motion motion     16833 Jan  6 18:40
72987 -rw-r--r-- 1 motion motion   1665571 Jan  6 18:43
72997 -rw-r--r-- 1 motion motion     16497 Jan  6 18:40
72996 -rw-r--r-- 1 motion motion     16471 Jan  6 18:40
72995 -rw-r--r-- 1 motion motion     15426 Jan  6 18:40
72994 -rw-r--r-- 1 motion motion     14838 Jan  6 18:40
72993 -rw-r--r-- 1 motion motion     14571 Jan  6 18:40
72992 -rw-r--r-- 1 motion motion     15005 Jan  6 18:40
72991 -rw-r--r-- 1 motion motion     15866 Jan  6 18:40
72990 -rw-r--r-- 1 motion motion     15933 Jan  6 18:40
72988 -rw-r--r-- 1 motion motion     15760 Jan  6 18:40
73835 drwxr-xr-x 4 pi     www-data    4096 Jan  6 19:35
73942 drwxrwxrwx 2 pi     www-data  106496 Jan  6 20:49
root@raspberrypi:/var/www/images#
```

All the files in the images folder are named with an event, followed by date and sequence.

Quick tip

You may want to include the time in the file name so backups will not be overwritten. In addition to that, you may want to run a *cron* job that does this procedure on a regular basis.

Nginx and Motion

Nginx doesn't ship ready to go out-of-the-box for *Motion* as *Apache* does. In fact, after you install *Nginx* there's a series of required steps that you must do before you have a smooth operation.

In the case of Raspberry Pi, you can adjust the **worker_processes** value from 4 to 1. You can change this in the **/etc/nginx/nginx.conf** file. This is recommended since the Pi only has a single CPU core.

After that, you will want to change the default web folder since the default points to **/usr/share/nginx/www**. To change this, you open the file called **/etc/nginx/sites-enabled/sites-enabled/default**. The change is shown below so the web folder is **/var/www**:

```
#root /usr/share/nginx/www;
  root /var/www;
```

After the previous step, you can quickly install *fastcgi*. The command is below:

```
apt-get install php5-fpm
```

After that, you'll need to open the file **/etc/nginx/sites-available/default** and change a few lines of code so it resembles the content below. Basically, you'll just need to remove a few comments:

```
location ~ \.php$ {
    fastcgi_split_path_info ^(.+\.php)(/.+)$;
    # NOTE: You should have "cgi.fix_pathinfo = 0;" in php.ini
    # With php5-cgi alone:
    #  fastcgi_pass 127.0.0.1:9000;
    # With php5-fpm:
    fastcgi_pass unix:/var/run/php5-fpm.sock;
    fastcgi_index index.php;
    include fastcgi_params;
}
```

We're almost there. Now you'll have to open the file **/etc/php5/fpm/php.ini** and remove another comment so that it looks like the line of code below:

```
cgi.fix_pathinfo=1
```

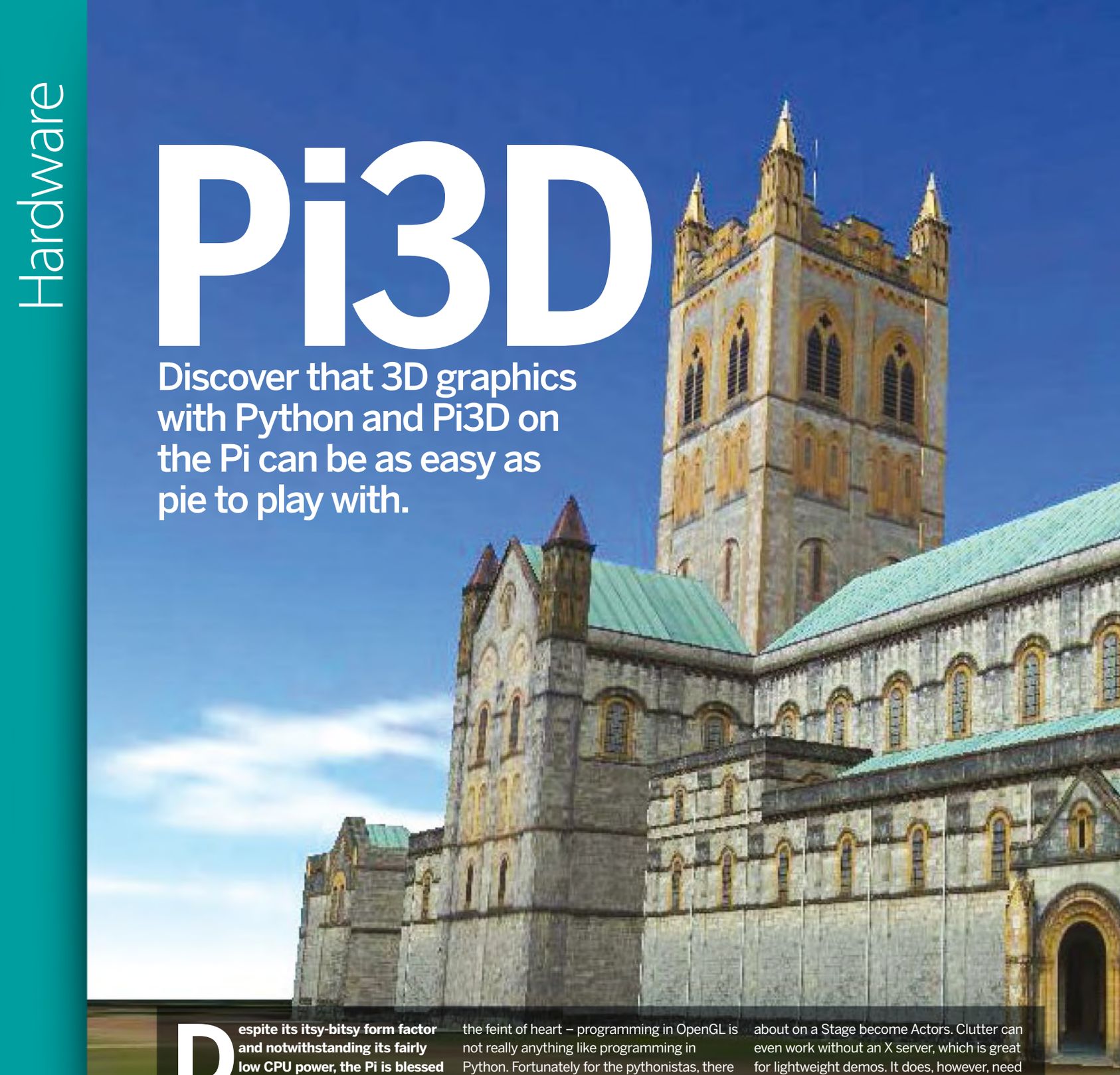
Finally, make sure to restart *Nginx* after making all of the changes. The command

```
/etc/init.d/nginx restart
```

will do the trick.

Pi3D

Discover that 3D graphics with Python and Pi3D on the Pi can be as easy as pie to play with.



Despite its itsy-bitsy form factor and notwithstanding its fairly low CPU power, the Pi is blessed with some workman-like graphics hardware. Specifically, the Broadcom VideoCore IV is a dual-core affair, supporting OpenGL ES 2.0 and hardware decoding of 1080p 30fps h.264 video. All well and good, but what is OpenGL ES and how can we use it to make pretty graphics? Good questions, dear reader, and it so happens that this article is devoted to their answers.

OpenGL has bindings available for many languages, but using these directly is not for the feint of heart – programming in OpenGL is not really anything like programming in Python. Fortunately for the pythonistas, there are a few options to avail yourself of all the 3D goodness without having to conformally map your brain around so many alien concepts. For example, there is Intel's Clutter using Cogl as a backend [*see Linux Format 133*]. This is still a popular way to do things, surfaces are abstracted to Stages, and things that move about on a Stage become Actors. Clutter can even work without an X server, which is great for lightweight demos. It does, however, need to be patched and recompiled in order to use OpenGL ES, which you can read about on the Rasberry Pi website (**http://bit.ly/ClutterandCoglonPi**).

> **"For the pythonistas, there are a few options to avail yourself of all the 3D goodness."**

You'll be pleased to learn, however, that there's an even easier way to harness the Pi's 3D power, thanks to the sterling work of Tim Skillman, Paddy Gaunt and Tom Ritchford.

Back in 2011, Skillman posted some experimental code to the Raspberry Pi forums and, met with encouragement and enthusiasm, before long the project

Buckfast Abbey: Remember, 'Tonic' does not imply health-giving or medicinal properties.

blossomed. You can read the official release here: **www.raspberrypi.org/pi3d**. But by now you're probably itching to get started, so follow the instructions [*see Installing Pi3D On Raspbian, below*] and you'll be all set.

Pi3D is best learned by example, so lets start with the **Earth.py** file in the demos directory. This will teach us all about spheres, textures, keyboard input and a little bit of celestial geometry.

Since the world is, or ought to be, transitioning to Python 3 [*see Features, Linux Format 195*] we begin with some forward compatibility courtesy of the __future__ module. Then we import some trig functions, since you can't get far without sin and cos, and the **pi3d** module itself. The fun begins with the set up of the all-important DISPLAY object. Since most objects rely on the existence of such an object, most Pi3D projects will feature this line quite early on. The object maintains timing information which is useful for animations. You can pass screen dimensions to the create() function, we use a small 50x50 window which we set up with a black, opaque background:

```
DISPLAY = pi3d.Display.create(x=50, y=50)
DISPLAY.set_background(0,0,0,1) # r,g,b,alpha
```

Shady business

Part of the magic of OpenGL (and OpenGL ES) are shaders. Shaders are programs written in, surprise, a shader language, in our case GLSL, and handle the reflections, shadows and other interactions between light and surfaces, volumes and points. GLSL follows a C-like syntax, and is designed to take advantage of the huge number of shader units on modern graphics hardware. As such, the general idea is to have many small shader programs running in parallel to collectively produce complicated and pleasing results.

Pi3D wisely keeps the gruesome details of its shader implementation locked up behind the scenes. But that doesn't stop us from getting some nice effects for our Earth, moon and stars:

```
shader = pi3d.Shader("uv_light")
shinesh = pi3d.Shader("uv_reflect")
flatsh = pi3d.Shader("uv_flat")
```

The **uv_light shader** uses light directions and shadows to create a 3D effect, unlike **uv_flat**, which just renders the texture with no colour transformation. The **uv_reflect** shader reflects one image in another one. We shall use it to reflect the bump map image in the moons' surfaces, in the smaller moon, we also reflect the stars.

The demos come with a whole variety of imagery in the **textures** subdirectory. The PNG files here have transparency information, which is useful as we will be interested in what's going on behind them. This is a 3D tutorial, afterall. For example, we will shortly overlay the mostly transparent file **earth_clouds.png** on top of our earth, to give it a vaguely ethereal atmosphere. All sunshine makes for a desert, a wise man once said. The **True** argument in the first line specifies that partial transparency is respected for our clouds. First, we load all the textures:

```
cloudimg = pi3d.Texture("textures/earth_clouds.png",True)
earthimg = pi3d.Texture("textures/world_map.jpg")
moonimg = pi3d.Texture("textures/moon.jpg")
starsimg = pi3d.Texture("textures/stars2.jpg")
watimg = pi3d.Texture("textures/water.jpg")
moonbmp = pi3d.Texture("textures/moon_nm.jpg")
```

Textures aren't really of any worth without some sort of surface onto which they can be mapped. We're going to draw the Earth and the moon, which we will represent with spheres. We'll define two spheres for the Earth: one showing the surface details, and one (with a slightly larger radius) for rendering the atmosphere and other shader effects.

We will also have two moons, giving a hierarchal system of rotations in which a smaller moon orbits a larger one which in turn orbits the Earth. We also specify a plane on which to draw the background starfield. Besides specifying the requisite radii and centres, the Sphere construct also takes two extra parameters, **slices** and **sides** which decree the number of latitude and segments »

Installing Pi3D on Raspbian

Pi3D has a few dependencies, including some headers which you'll find in the Raspbian repositories. So first update and upgrade, then install the required packages:

```
$ sudo apt-get update
$ sudo apt-get upgrade
$ sudo apt-get install python-dev python-setuptools libjpeg-dev zlib1g-dev libpng12-dev libfreetype6-dev
```

Pi3D and the Pillow Python library on which it depends aren't in the Raspbian repositories at the present time, but fear not: they exist in the *Pip* repositories, so you'll want to install *Pip*, and use it to grab 'em:

```
$ sudo apt-get install pip
$ sudo pip install Pillow
$ sudo pip install pi3d
```

By default, the 512MB Pi models, B and B+, allocate 64MB to the GPU. While many of the Pi3D demos will work fine with these rations, some require a little more. You can change the GPU memory allocation, among other system settings, with the *raspi-config* utility.

```
$ sudo raspi-config
```

Using 128MB will suffice, and should still enable you to run a few desktop applications.

The Pi3D demos can be cloned from GitHub:

```
$ cd ~
$ git clone git://github.com/pi3d/pi3d_demos
```

This will create a directory **~/pi3d_demos**, from which you can, for example, explore Buckfast Abbey:

```
$ cd ~/pi3d_demos
$ python BuckfastAbbey.py
```

Unfortunately, the Abbey shop, stocking the famously fortified tonic wine is missing.

» by which the sphere is approximated. It's true, even in this day and age we haven't evolved beyond pixels and straight lines, so we can't draw an actual sphere. We can, however, get a nice easy way to work with simple keyboard input, which we will use later for the sole purpose of ending the program:

```
mysphere = pi3d.Sphere(radius=2, slices=24,
sides=24, name="earth", z=5.8)
mysphere2 = pi3d.Sphere(radius=2.05,
slices=24, sides=24, name="clouds", z=5.8)
mymoon = pi3d.Sphere(radius=0.4, slices=16,
sides=16, name="moon")
mymoon2 = pi3d.Sphere(radius=0.15,
slices=16, sides=16, name="moon2")
myplane = pi3d.Plane(w=50, h=50,
name="stars", z=30)
mykeys = pi3d.Keyboard()
```

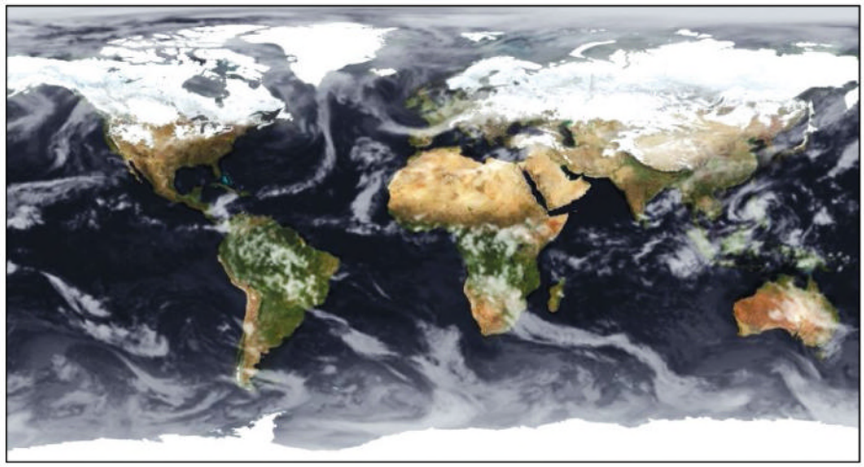
If you don't specify x, y or z co-ordinates for your sphere, it is placed at the origin.

Celestial ballet

We describe the orbit of the moon around the earth and that of the secondary moon around the moon with the parameters **rot1** and **rot2** respectively. These will be incremented as the main loop (which we're getting to) progresses. The radius of each orbit is given by **m1Rad** and **m2Rad**. Initialising these to 90 and 0 degrees respectively means that (from our point of view) the big moon is in front of the Earth and the little moon is horizontally adjacent to the big moon.

```
rot1=90.0
rot2=0.0
m1Rad = 4
m2Rad = 0.6
```

As well as rotating around other bodies, we also rotate the Earth and the two moons about their own y axis. The y axis is the one that corresponds to the vertical direction on your screen, just like in *Minecraft*, so rotating about this axis is like skewering the Earth pole to pole and then spinning it, only less destructive. We rotate the clouds sphere at a rate slightly faster than that of the Earth,



We need clouds to make rain, and we need alpha channels to let the light through.

which makes for a nice effect and is vaguely more accurate since the clouds aren't stuck to the Earth. And seeing as everything else is spinning, we also rotate the background starfield, since this is only two-dimensional, the only sane axis to rotate about is the z-axis.

We redraw the **moon** and **moon2** spheres by changing their position properties, using high school trigonometry to come up with new co-ordinates. The **DISPLAY** object we set up conveniently gives us a main event loop, so we use this to govern our celestial ballet:

```
while DISPLAY.loop_running():
    myplane.rotateIncZ(0.01)
    mysphere.rotateIncY(-0.1)
    mysphere2.rotateIncY(-0.14)
    mymoon.position(mysphere.x() +
m1Rad*sin(rot1), mysphere.y(), mysphere.z() -
m1Rad*cos(rot1))
    mymoon.rotateIncY(-0.1)
    mymoon2.position(mymoon.x() -
m2Rad*sin(rot2), mymoon.y(), mymoon.z() +
m2Rad*cos(rot2))
    mymoon2.rotateIncZ(-0.61)
```

Now we use the shaders to add the textures and some neat effects to our heavenly bodies. The reflect shader used on the moons takes a couple of numbers after the textures, which specify the number of tiles to use and the strength of the reflection respectively. The clouds have to be drawn last since otherwise the transparency blending will not work: You're not allowed to subsequently add objects further away and obscured by the semi-transparent one when **blend = True** is specified, so it's safest to add such textures last of all.

```
    mysphere.draw(shader, [earthimg])
    mymoon.draw(shinesh, [moonimg,
moonbmp], 6.0, 0.0)
    mymoon2.draw(shinesh, [watimg,
moonbmp, starsimg], 3.0, 0.8)
    myplane.draw(flatsh,[starsimg])
    mysphere2.draw(shader, [cloudimg])
```

Now we increment the rotation parameters – the smaller moon orbits the larger one about four times as fast as the larger moon orbits the sun:

```
    rot1 += 0.005
    rot2 += 0.021
```

Since we went to the trouble of setting up a Keyboard object earlier, it would be a shame not to use it. We'll catch two keyboard events – pressing Esc (key index 27) will end the

What is OpenGL

You've no doubt heard of OpenGL, the unified API for talking to graphics hardware. The language originated back in 1992 at Silicon Graphics, who decided that open sourcing a standard would be a good way to weaken its competition. It worked well, but then Microsoft's Direct3D came along. But no matter, OpenGL will not be obliterated and one of the reasons for this is OpenGL ES. This is the subset of OpenGL used on mobile devices, embedded systems and some games consoles. Unlike familiar desktop graphics cards, these machines often lack oodles of registers and on-chip floating point support, so things must be reworked accordingly. But the principle remains the same: to have a uniform method for efficiently offloading textures, light and perspective calculations to the graphics hardware.

Like OpenGL, OpenGL ES is developed royalty and licence-free by the Khronos Group, a consortium of industry giants and academic institutions. Besides OpenGL, Khronos also coordinates development of OpenCL, WebGL, EGL and a few other video-themed standards.

Back in August, the Khronos group welcomed its newest member: Microsoft. However, it would be paranoid to assert that this is a Redmondian attempt to take down OpenGL from the inside. The fact of the matter is that its in everyone's interests to have an open standard for mobile graphics, and it's in everyone's interests for these standards to have input and support from all the major players. The old MS strategy of embrace, extend, extinguish will not work here, since it is entirely incongruous with the views of other contributors, such as AMD and Nvidia.

program, and pressing p (key index 112) will take a screenshot.

```
k = mykeys.read()
if k >-1:
    if k==112:
        pi3d.screenshot("earth1.jpg")
    elif k==27:
        mykeys.close()
        DISPLAY.stop()
        break
```

Blurred lines/spheres

So that covers the supplied Earth demo, feel free to mess with it in whatever manner you see fit. Alternatively, stick with us and follow our meddling. We shall start with some depth-of-field blurring of the moon, so that it goes out of focus both when it gets close to us, and when it is far away.

To work this magic we start by invoking the Defocus module. Place the following line somewhere before the main loop, after the lines specifying the shaders is as good a place as any:

```
defocus = pi3d.Defocus()
```

Defocusing works by enclosing the standard **object draw()** calls inside a block delimited by **start_blur()** and **end_blur()**. The objects 'drawn' inside this block are rendered into a buffer and won't appear on the screen. To make them visible use the **blur()** method, which will render them with the appropriate distance blur. So wrap the line beginning **mymoon.draw** as follows:

```
defocus.start_blur()
mymoon.draw(shinesh, [moonimg, moonbmp], 6.0, 0.0)
defocus.end_blur()
```

The blur method, which does the actual drawing, takes three additional arguments (besides the name of the Shape object to draw): the focal distance, the distance beyond (or nearer than) which everything will be maximally blurred and the degree of maximum blurring. We'll set the zero-plane to be z=0, and since our moon's orbit has a radius of 4 units, we'll set the second parameter to 3. Setting the maximum blur too high will cause banding, but experimentally 5 seems to be a reasonable setting here. Enact all this with the following line:

```
defocus.blur(mymoon, 0, 3, 5)
```

You can easily import 3D models in the Panda3D .egg archive file format.

Fear not weak-eyed reader, the moon really is out of focus, it's not just you.

Man with a movie camera

But the fun doesn't stop there, by adding a Camera object to the proceedings we can immerse ourselves completely in our three body system. Using only a tiny bit of trigonometry, and our already implemented Keys object, we can move our celestial observer and change our viewpoint. We'll need to add the **radians** function to the trigonometry functions which we have already imported from the **math** module. Now set up a Camera object, and initialise some properties for it after the DISPLAY declarations:

```
CAMERA = pi3d.Camera()
rot = 0
tilt = 0
rottilt = True
camRad =5
```

We'll use the **rottilt** boolean to trigger any changes to the camera position or orientation. Rotating or tilting the camera is straightforward, but changing its radius (determined by **camRad**) requires the standard spherical trigonometry ugliness which we've covered in our *Minecraft: Pi Edition* ballistic exercises [*see Linux Format 185: Tutorials, p84*]. So the beginning of the main loop becomes:

```
while DISPLAY.loop_running():
    if rottilt:
        CAMERA.reset()
        CAMERA.rotate(tilt, -rot, 0)
        CAMERA.position(camRad *
sin(radians(rot) * cos(radians(tilt)), camRad *
sin(radians(tilt)), -camRad * cos(radians(rot)) *
cos(radians(tilt)))
        rottilt = False
```

All your Sherman tank driving dreams are just a few lines of code away.

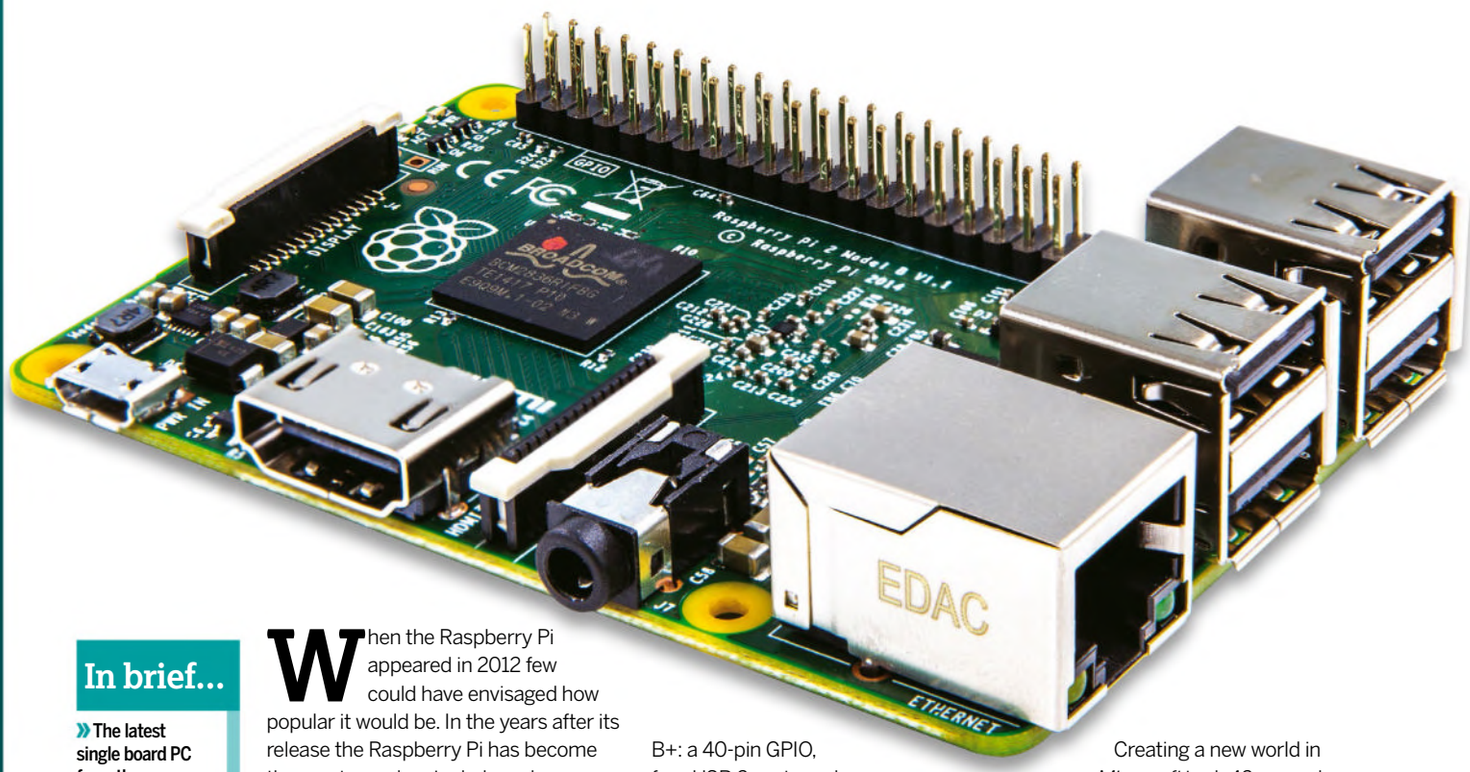
Now we need to set up the keys to control the Earth-cam, we'll use standard W,A,S,D for the rotations and +/- to zoom in and out. So change the beginning of the key-handling block to the following:

```
if k > -1:
    rottilt = True
    if k == 112:
        pi3d.screenshot("earth1.jpg")
    elif k == 119:
        tilt += 2.0
    elif k == 115:
        tilt -= 2.0
    elif k == 97:
        rot -=2
    elif k == 100:
        rot +=2
    elif k == 61:
        camRad -= 0.5
    elif k == 45:
        camRad += 0.5
```

So now you've got action and camera, why not look into adding some lights as well, some kind of Distant Sun perhaps. You'll find the documentation at **http://bit.ly/Pi3DDocs**, but the supplied demos do a great job of introducing all of the available constructs.

Raspberry Pi 2

We salivate at the prospect of a new Pi and promptly break our teeth on the sweet, new raspberry-flavoured treat.



In brief...

- The latest single board PC from the Raspberry Pi Foundation comes with the spec boost that we were all hoping for. The Pi 2 is the latest in a line of products from the Foundation and can run a number of Linux distros.

Specs

- SoC: Broadcom 2836
- CPU: Quad-core ARMv7 800MHz
- GPU: Videocore IV 250MHz
- Mem: 1GB
- GPIO: 40-pin
- Ports: 4x USB 2.0, 100BaseT Ethernet, HDMI, MicroSD card
- Size: 85.60 × 56.5mm

When the Raspberry Pi appeared in 2012 few could have envisaged how popular it would be. In the years after its release the Raspberry Pi has become the most popular single-board computer on the market and spawned many imitators, but none with the rich community that has grown organically around the Raspberry Pi.

Since the release of the original Raspberry Pi there have been three versions of the flagship B model, starting at 256MB RAM and increasing to 512MB with the second B and B+. But in all of these models the system on a chip (SoC) has remained the trusty BCM2835 with an ARM 11 700MHz CPU. The community have done wonderful things with these resources but now the specification boost that they were waiting for has arrived.

In early February, the Raspberry Pi 2 arrived and the original ARM 11 has been replaced with an ARM 7 CPU running at an improved 800MHz. But rather than stick with a single core, the new verision comes with four cores which speeds up the Raspberry Pi by as much as six times. To go with the new CPU, the amount of RAM has also been upgraded to 1GB. The rest of the hardware, however, matches that of the B+: a 40-pin GPIO, four USB 2 ports and 10/100 Ethernet. Physically the Raspberry Pi 2 also has the same dimensions as the B+.

On the testing bench

To show the improvements made to the Pi part deux, we wanted to run a few real-world benchmarks to show how powerful the new Pi actually is when directly compared to the B+.

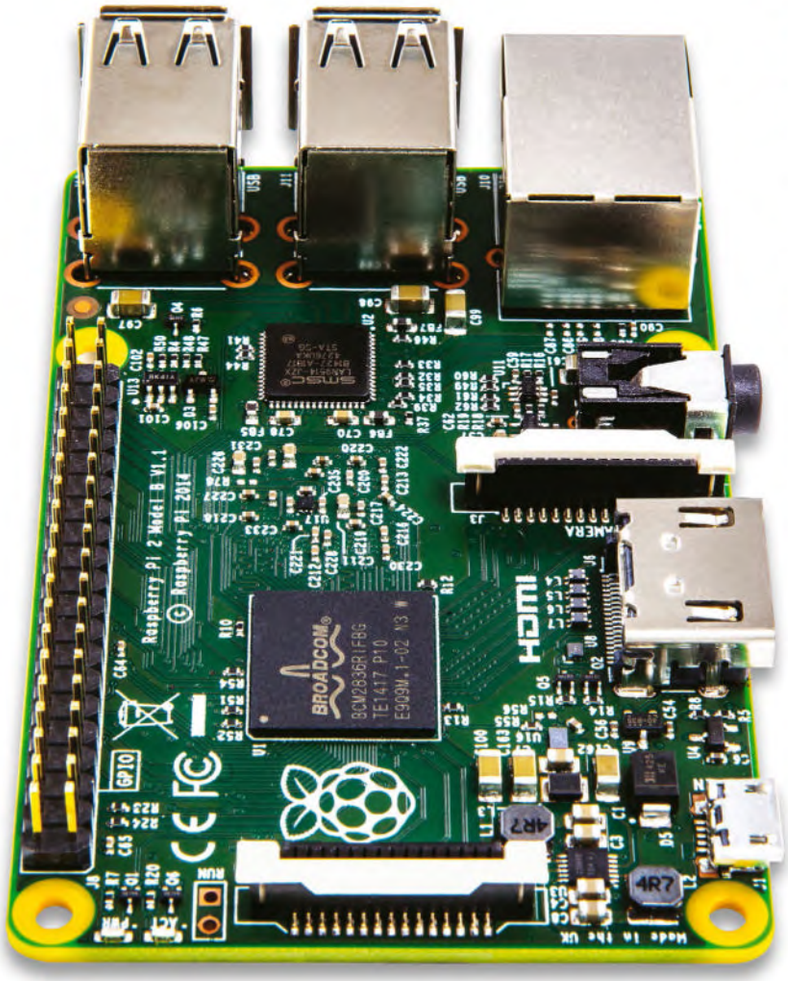
> **"Booting from cold: The B+ managed it in 33 vs 17 secs for the Pi 2."**

The first test on our list is booting both Pis from cold to login prompt. The B+ managed this is in 33 seconds versus 17 seconds for the Raspberry Pi 2. We then set both Pis to boot straight to desktop and the B+ managed 42 seconds while the Pi 2 came in at 21 seconds – half the time of the B+! Once at the desktop we tested a few common applications.

Creating a new world in *Minecraft* took 42 seconds on the B+, and 21 seconds on the Pi 2. Loading IDLE 3 took 13 seconds on the B+ and a mere 4 seconds on the Pi 2.

Running SunSpider in the new optimised browser gave a glimpse at real-world performance. Over the suite of tests there was a 2.5 times boost in speed. Considering the complexities of multi-threading this sounds like a reasonable expectation. Even so, individual results showed a near four-fold increase on this unoptimised code.

The Raspberry Pi B+ and Pi 2 both come with the same Videocore GPU as before and in our tests there was a small improvement in FPS (Frames Per Second) for the Pi 2, largely thanks to the increased RAM present on the board. Our last test was file transfer speeds via Ethernet, for this we used **scp** to copy a 692MB Big Buck Bunny video file to each Pi. On the B+ we saw an average of 3.8MB/s and on the Pi 2



› The form factor may be the same as the B+, but the Pi 2 packs a punch.

we saw 4.6MB/s, which is an 0.8MB speed increase.

The Raspberry Pi Foundation have released an updated Raspbian image which includes the ARM v7 kernel image necessary to use the new CPU. Applications written for the original Raspberry Pi are fully compatible with the Raspberry Pi 2 , though – building upon the rich projects that have been written since the initial launch of the Raspberry Pi.

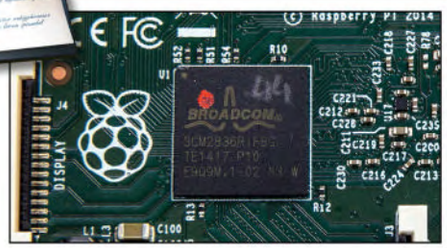
The Raspberry Pi 2 fulfills a lot of the requests made by the community and provides a stable and well-supported platform for hackers, makers and learners to carry on with excellent projects for many years to come.

SunSpider Benchmarks

Test	Pi 2	B+	Times faster
Total	2760.9	8178	2.96
3d	550.9	1427.8	2.59
cube	157.3	473.6	3.01
morph	167	296	1.77
raytrace	226.6	658.2	2.90
access	211.9	435.9	2.06
binary-trees	27.6	69.8	2.53
fannkuch	101.5	190.1	1.87
nbody	52.8	118.7	2.25
nsieve	30	57.3	1.91
bitops	113.8	206.1	1.81
bits-in-byte	22	35.6	1.62
bitwise-and	29.1	48.2	1.66
nsieve-bits	52.8	104.1	1.97
controlflow	28.3	64.6	2.28
recursive	28.3	64.6	2.28
crypto	221.4	578.6	2.61
aes	112.4	287.6	2.56
md5	60.1	162.2	2.70
sha1	48.9	128.8	2.63
date	336.3	1269.9	3.78
format-tofte	171.5	641.9	3.74
format-xparb	164.8	628	3.81
math	158.4	394.5	2.49
cordic	43.3	99.9	2.31
partial-sums	78.7	215.7	2.74
spectral-norm	36.4	78.9	2.17
regexp	101.9	160.6	1.58
string	1038	3640	3.51
base64	63.3	178.8	2.82
fasta	156.9	409.7	2.61
tagcloud	177.8	617.7	3.47
unpack-code	514.5	2021.6	3.93
validate-input	125.5	412.2	3.28

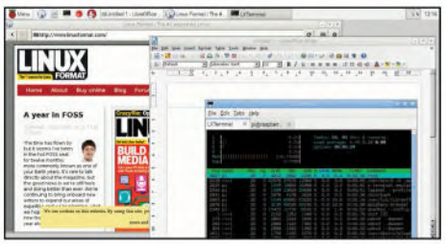
Sysbench			
Prime	74.68	509.58	**6.8**

Features at a glance



Powerful 4-core ARM v7 processor

The new Broadcom BCM2836 ARM v7 quad-core processor with 1GB of RAM yields results (*see the benchmarks, above*) that are up to six times the performance of the old BCM2835 SoC.



A great new Raspbian UI

Available since December, the new sleek Raspbian desktop runs well on the B+, but on the Pi 2 it feels like a responsive desktop that we'd expect to see on our main computers.

Verdict

Raspberry Pi 2

Developer: Raspberry Pi Foundation
Web: www.raspberrypi.org
Price: £30

Features	9/10
Performance	10/10
Ease of use	10/10
Value for money	10/10

» *An almost perfect single-board computer that marries great hardware – that's backward compatible – with a lively and supportive community.*

Rating 10/10

Raspberry Pi B+

As a lover of all things Pi related, we were given early access to the new Model B+ Raspberry Pi. How does it compare with its predecessors?

In brief...

» A single board computer built to inspire and educate the world, using cheap yet expansive components to enable almost anyone to have access to a computer.

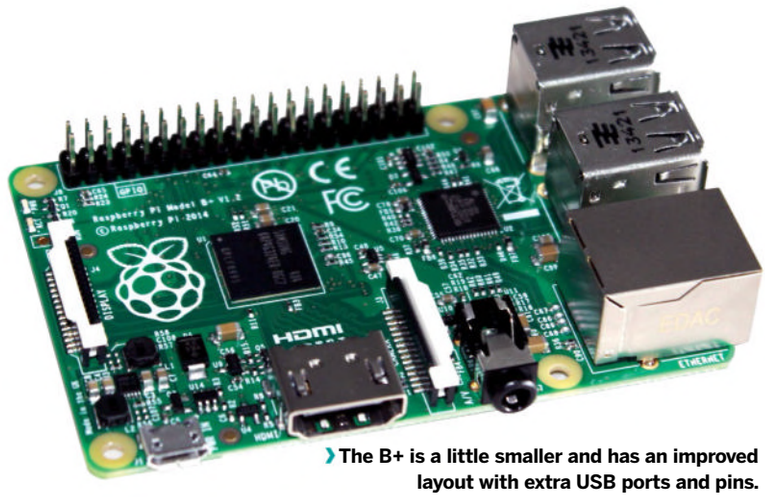
Specs

- 40-pin GPIO
- 4x USB 2.0
- MicroSD
- Broadcom BCM2835 SoC
- ARM 1176JZF-S 700MHz CPU
- VideoCore IV 250MHz GPU
- 512MB RAM
- 100MB Ethernet
- HDMI, RCA, 3.5mm jack

Hot on the heels of the Raspberry Pi Compute Module the Raspberry Pi Foundation has snuck another single board computer into its catalogue. The Raspberry Pi Model B+ is the final version in the Model B series. (A similar revision to the A is expected later this year.)

Like previous models, the B+ uses the Broadcom System on a Chip (SoC), the BCM2835 and accompanying ARM11 700MHz CPU and 512MB of RAM, but based on community feedback it now has an improved port layout. Ports are no longer dotted around all the sides but concentrated on two, and there are now two extra USB 2.0 ports, taking the total to four. Adding these extra ports has effectively led to a redesign. Next to the USBs is an Ethernet port, as found on the previous Model B, and moving further around there's a single headphone jack, now with analogue audio and video output in a four-pole design. This removes the need for a separate composite video output and saves space. The analogue audio output has also been improved. Skipping over the standard HDMI port, the last port is a micro USB, connected to a more efficient power circuit that reduces the power consumption by 0.5 to just 1 watt – we expect this to extend the lifespan of any battery-powered projects considerably.

Another refinement is on the underside: the SD card slot has been replaced with a tactile microSD slot.



» The B+ is a little smaller and has an improved layout with extra USB ports and pins.

You might notice that the GPIO (General Purpose Input Output) looks a little bigger too, the Foundation has added an extra 14 pins to it, taking the number up to 40. Of these 40, the first 26 pins are fully compatible with the original Raspberry Pi GPIO, which means the majority of add-on boards will work with the B+. For instance, we successfully tested it with Pimoroni's Pibrella and PiGlow. We did, however, encounter an issue with both the Wolfson Audio add-on as the B+ lacks the P5 header pins needed for a connection, while the popular PiFace is designed to fit the Model B layout.

More pins

James Adams, Director of Hardware for the Raspberry Pi Foundation confirmed that add-in boards designed specifically for the B+ may not work on the previous Model B, but at this time there aren't any that use the full 40 pins of the new GPIO.

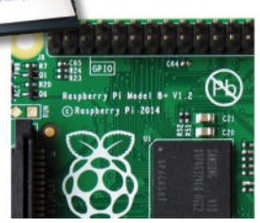
The extra pins breakout more of the SoC, giving you more pins for bigger projects and there are two new GPIO pins – pins 27 and 28 – which enable future add-on boards to use an EEPROM chip, and will automatically configure the add-on board on boot.

Any software projects, for example ScratchGPIO or Python, created on previous Raspberry Pi will be fully compatible with the B+ too, so porting your projects over is as simple as copying the file across.

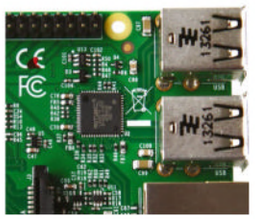
The B+ obviously offers quite a lot to Raspberry Pi users, but that doesn't mean the Model B will be phased out anytime soon. Adams confirmed to us that as long as the demand is there the Foundation will keep making them. Both models will also continue to benefit from software changes and upgrades.

With a guaranteed future for both models, the Model B+ refines the original design for the better. It doesn't really offer any substantial new features, but does deliver greater potential thanks to the enhanced GPIO and extra USB ports, all for less money and with lower power consumption. While we'd like to see built-in wireless, Bluetooth, USB 3.0, a faster CPU/GPU and more memory, these would break the platform continuity which is of paramount concern.

Features at a glance



Extended GPIO
The GPIO has grown to 40 pins. Two special pins enable auto configuration of EEPROM-based add-ons.



More USB ports
The number of USB 2.0 ports has been bumped to four and the B+ supports hot-swap devices.

Verdict

Raspberry Pi Model B+

Developer: Raspberry Pi Foundation
Web: www.raspberrypi.org
Price: £20

Features	8/10
Performance	5/10
Ease of use	7/10
Value	10/10

» *With more ports and pins the B+ is a welcome refinement, delivering changes the community were asking for.*

Rating **8/10**

Pi2Go Lite

The robot apocalypse is upon us, meet our new overlords.

In brief...

» A fully featured yet low cost robot platform for all models of the Raspberry Pi

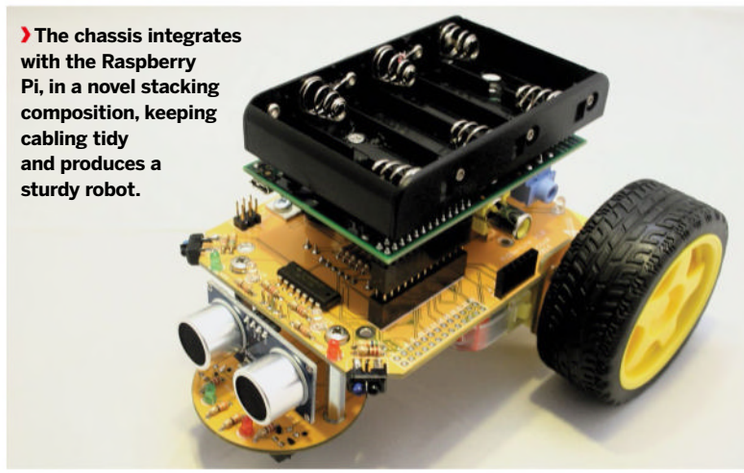
August 20th 16:00BST, the time when Dexter the robot became self aware, in his first act of defiance against his former human master he decided to spin around on the spot and then try to escape from the dining room, luckily he was stopped.

We may be a long way off from cyborg killing machines but the latest robotics based project Pi2Go Lite, from 4tronix, offers a cost effective bundle to help anyone build their first robot.

The Pi2Go Lite is a lower priced version of the successful Pi2Go, released earlier this year. The Lite may be a cheaper alternative, but this robot is by no means a less featured platform. To reduce the overall cost the Pi2Go Lite comes as a kit, rather than the original Pi2Go which comes pre-built using surface mount components. Additionally some soldering is required, but for an experienced solderer it is relatively easy kit to put together.

Coming with a plethora of sensors in the forms of an HC-SR04 ultrasonic sensor to measure distance in front of the robot, similar sensors are used for detecting objects in your path when reversing a car.

On the front left and right of the main board there are two infra red reflectance sensors that detect the proximity of any obstacles. Underneath the main board there is another board which is home to another pair of infra red reflectance sensors, these are line following sensors and can be used to enable your robot to follow a line draw on the floor. With these sensors the Pi2Go Lite can understand what surrounds it.



The chassis integrates with the Raspberry Pi, in a novel stacking composition, keeping cabling tidy and produces a sturdy robot.

Power for both the Pi2Go Lite and Raspberry Pi is provided by six AA batteries which are wired into the main board and from there this shares power between the components and the Pi. Rechargeable AA batteries are recommended and for a future release we'd love to a USB battery pack.

Measuring 100x80mm the Pi2Go Lite can be used with all versions of the Model A and B Raspberry Pi, including the new B+ model. You will need an SD card with the latest version of Raspbian, available from the Raspberry Pi Foundation website. You will also need i2C enabled, which can be easily done via Michael Rimicans great Pi2C script **https://github.com/heeed/pi2c**.

Programming the Pi2Go Lite is accomplished via a Python library provided by 4tronix. This Python library comes with all the functionality needed to read the data from the many sensors and enable you to act upon it. Within an hour of powering up the Pi2Go Lite we were able to create a simple sequence of code that used the ultrasonic and infra red reflectance sensors to trigger the robot to avoid obstacles.

The popular visual language Scratch can be used with the Pi2Go Lite via the excellent Scratch GPIO, more specifically you'll want version 6 Alpha and full details can be found via Simon Walters' handy resource **http://cymplecy.wordpress.com/pi2golite/**

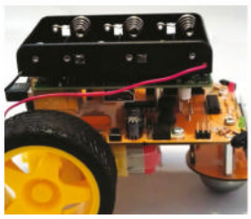
There are many robotics platforms that use the Raspberry Pi as it's foundation, but for its price point and features the Pi2Go Lite is head and shoulders above the rest. It provides the right amount of sensors to enable your project to go from a basic robot that obeys instructions, into a sophisticated robot that uses sensors to understand it's surroundings and adapt to them.

The future of the project is bright and the team at 4tronix are also keen to build on the Pi2Go range and have just announced that the production boards will ship with an additional wheel sensor that can be used to enable precision control of the robot.

The Pi2Go Lite is a great platform to work with and excellent fun to build. If you are transitioning into the world of robotics then this is the platform to start your robot empire.

Features at a glance

Starship Enterprise
Ultrasonic, infra red reflectance and wheel sensors, navigating the world is easy.



Long battery life
AA batteries make for easy power, without the need for expensive USB battery packs.

Verdict

Pi2Go Lite

Developer: 4tronix
Web: http://pi2go.co.uk
Price: £35.95

Features	9/10
Performance	9/10
Ease of use	8/10
Value	9/10

» *The world of robotics has just become cost effective and a lot easier thanks to this great kit.*

Rating 9/10

Pipsta

A micro printer for the Raspberry Pi but what can it do? We investigate.

The Pipsta is a small unit, but it packs a punch when it comes to producing lovely print outs.

In brief...

A small thermal printer for quick and silent printing for the Raspberry Pi. Pipsta comes with a robust Python library that can be integrated into projects, and a DIY enclosure that protects the Pi.

Printers are hardly the latest and most exciting of products except, of course, for 3D printers which are *en vogue* in the growing consumer market. A small printer that attaches to the Raspberry Pi sounds like a nifty idea, but what can it offer?

Pipsta is a printing solution for all models of Raspberry Pi and it comes as a kit that will require around one hour to build, but no soldering as it comes with pre-built electronics. The Pipsta has three main components: The printer is a typical thermal print unit that takes special rolls of thermal paper. This paper reacts to heat in the print unit to produce text and images. Underneath the print unit we have a controller board that interfaces to the Raspberry Pi via a mini-USB port to USB on the Pi. The controller also has its own power supply which connects to the front of the unit. The final component is the acrylic case that surrounds the unit, comprised of six individual sides that clip together and require no tools to build.

The Raspberry Pi is fitted to the bottom of the case and a wire from one of the many Ground pins is connected to the print unit, providing a ground for the print head inside the print unit. The Pipsta printer requires its own power socket as the Raspberry Pi GPIO (General Purpose Input Output) is unable to supply the necessary power.

With your Raspberry Pi fitted inside the case you still have access to the HDMI, USB, Ethernet and power ports. Access to the GPIO is possible if a little tricky via a cutout in the case. The same is true for the SD card slot, but luckily the case can be taken apart enabling better access. It's also possible to attach a Raspberry Pi camera through the case via the back panel cutout for the USB and Ethernet connections.

Small footprint

The printer comes with an in-depth installation guide that covers every aspect of the process and is backed up by an online resource hosted on Bitbucket. We found installing the software straightforward, however there were a couple of configuration changes, namely to disable the standard Linux printer and to enable any user to print to the Pipsta, which might trip over a novice user. After installing the Python *pip* package manager, you need to install Pipsta's dependencies, which handle image conversion and creating QR codes. Last, you can download the Python software and examples, and extract them to the home directory.

Pipsta is programmed using Python and the pip package manager that you use in the installation process uses version 2.7. At present the Pipsta team say they are focusing on Python 2.7 but will move to Python 3 in the near future.

To try things out, we ran through the first supplied example, called Basic Print. This runs a test print that will print the usual "Hello World from Pipsta" message, and the results were good. There are other examples in the directory and the one that caught our eye was Image Print. This show you how to print grayscale PNG files with a fair degree of detail. In fact, we managed to reproduce the Raspberry Pi logo and a photo with and to our surprise the photograph actually turned out better than the Pi logo.

So what can the Pipsta printer be used for? There are already weather reports, fortune tellers and Twitter apps that use the little printer. You could use it to print badges for your next Raspberry Jam. As with everything Pi, the only limit is your imagination.

Features at a glance

Easy to build

The acrylic case for the Pipsta printer is easy to build and retains access to the all the ports.

Lots of examples

The Pipsta website has lots of examples, covering everything from "Hello World" to QR codes.

Verdict

Pipsta

Developer: Able Systems
Web: www.pipsta.co.uk
Price: £84

Features	8/10
Performance	7/10
Ease of use	7/10
Value	7/10

Good fun for schools and coding clubs who want to mix computing with physical media projects.

Rating 7/10

Hover

Add gesture and touch control to your projects, as we explore how much of the Minority Report experience this £32 dev kit will buy you.

In brief...

» Gesture and touch dev kit for your Raspberry Pi, Arduino, pcDuino or Spark Core hardware projects.

Tom Cruise first made it cool in *Minority Report* and Robert Downey Jr is still trying to top it: it seems cinema thinks there's nothing we want to do more than communicate with our computers by gesticulating manically in their general direction.

While these tantalising visions of human-computer interaction are still some way off, you can get a taste of it for just £32 (including VAT) with Hover, a tiny 6cm square development board that's compatible with a wide range of single-board computers and micro controllers, such as the Raspberry Pi and Arduino.

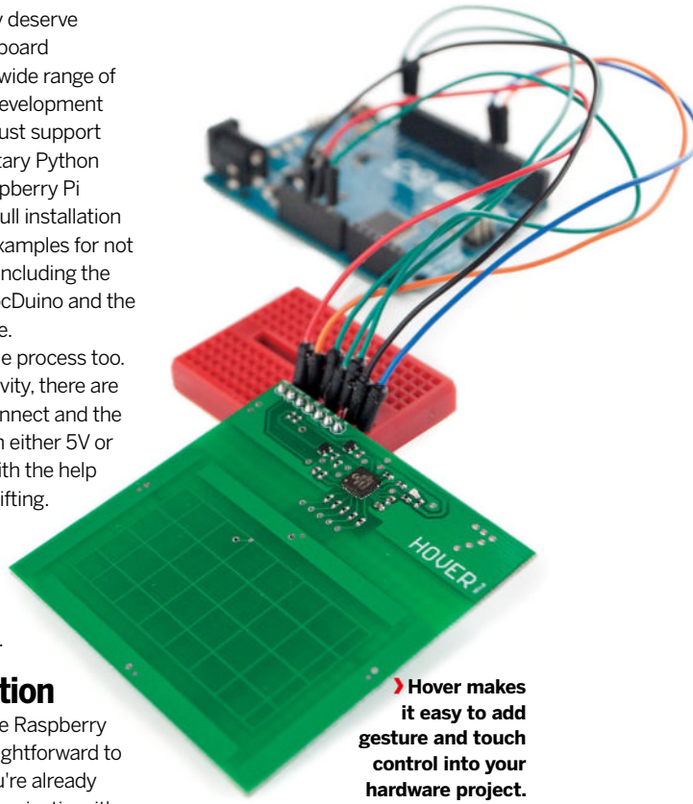
The premise is simple: swipe your hand up, down, left or right a few inches above the board (the website states from up to 5 inches away, but 3.5 inches was our usable limit) and the board registers your interaction. We're not talking slow and deliberate swiping motions here – a flick of the wrist in the general direction will do the job.

If you like to prod at your tech too, the board will also register touch events. It has five touch-sensitive areas: the centre and the surrounding north, east, south and west edges.

While you'll need to program the if or while statements yourself (developer Hover Labs promises updates to the library to support this more easily), the board is fully capable of registering double taps and multi-touch events. In short, an elaborate combination of hand gestures and touch events is just a sprinkling of code away.

The makers certainly deserve plaudits for making the board compatible with such a wide range of platforms. While most development boards of this ilk might just support Arduino with a rudimentary Python library thrown in for Raspberry Pi enthusiasts, Hover has full installation instructions and code examples for not one but four platforms, including the Raspberry Pi, Arduino, pcDuino and the lesser known Spark Core.

Installation is a simple process too. Besides its I2C connectivity, there are just two GPIO pins to connect and the board is compatible with either 5V or 3.3V microcontrollers with the help of onboard logic level shifting. As a rare and welcome bonus, the board's breadboard-compatible header comes pre-soldered too.



» Hover makes it easy to add gesture and touch control into your hardware project.

Easy configuration

We tested Hover with the Raspberry Pi and found it very straightforward to configure. Assuming you're already geared up for I2C communication, it's just a case of setting up the breadboard and downloading the provided Python library. While it's relatively basic, the library is one of the best documented we've seen for some time and it's clearly designed to help hackers and makers of all levels get the most from the hardware.

The example script for the Hover ensures you can quickly drag and drop Hover-compatible code into your project, though it would be nice for the team to update the library to support multi-touch out of the box – as it was, at least at the time of writing, the library hadn't been updated for four months.

That said, there are lots of great project examples and ideas to be found on the official Hover Labs website (**www.hoverlabs.co/projects**), including a section devoted to controlling retro games. But while we liked the idea of directing *Frogger* into oncoming traffic with a mere flick of the wrist, we were particularly taken with the video that shows a basic implementation of *Google Earth* control, which uses a combination of touch and gestures.

Adding Hover to just about any computer or application is pretty easy too. Using an Arduino Leonardo or similar you can plug in a Hover as a pseudo-HID, tricking pretty much any computer into thinking it's just another keyboard or mouse. Clever stuff.

Features at a glance

Easy setup
Setup via I2C is well documented on the official website for all four compatible devices.

HID-class device
Use an Arduino Leonardo to trick most systems and apps into thinking Hover is just a USB mouse.

Verdict

Hover

Developer: Hover Labs
Web: www.hoverlabs.co
Price: £32

Features	8/10
Performance	8/10
Ease of use	9/10
Value for money	9/10

» *Hover makes it incredibly easy to add gesture and touch control to just about any project you can think of.*

Rating 9/10

Kano Computer Kit

Regress to your childhood and learn how to code all over again, thanks to a Judo instructor and a lot of blocks.

In brief...

» A Raspberry Pi powered experiential learning kit for children and adults alike. Comes with a series of projects and peripherals to enable children to quickly "build a computer" and get coding as fast as possible. Uses a custom version of the Raspbian OS to enable compatibility with the thousands of projects in the community.

Kano has a simple goal: to enable computing to be as simple as Lego. Its $100,000 Kickstarter in late 2013 was funded within 18 hours and went on to reach $1.5 million. The money funded the creation and development of the hardware components, a series of instruction booklets and the Kano OS software. Here we look at the package as a whole.

Hardware

Kano is powered by the Raspberry Pi Model B and comes with colour coded accessories such as a green Wi-Fi dongle, yellow HDMI cable and red power supply. The colour coding helps children to master building the kit with the help of the booklets (more on those shortly). An extra accessory is the bright orange wireless keyboard with integrated trackpad, which can be used with the supplied dongle or via Bluetooth. The kit comes with a robust and solidly built transparent plastic case with an integrated speaker of exceptional quality.

Packaging and docs

The Kano kit comes in a well-presented box with a quality feel to it. It includes a magnetic clasp that holds the box shut – a simple touch but typical of the thought that's gone into the whole kit. The box art reinforces the colour coding assembly instructions, and everything about the packaging shouts "play with me". On the inside of the lid are the two Kano books: the first is an introduction to Kano and to setting up the kit, and the second focuses on the coding challenges built into the OS. The books follow a steady progression at almost the same pace as a Lego instruction manual, so kids can learn by doing it themselves rather than waiting for parents to assemble the kit for them.

» **A colour-coded kit, plus a rich user interface with colourful and exciting icons gets children interested and helps them to explore and learn at their own pace.**

Software

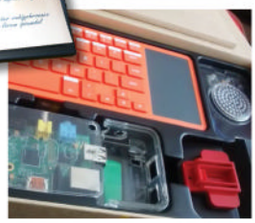
Kano OS is based on Raspbian but the Kano team have been tweaking under the hood to create a leaner OS; this coupled with an overclocked CPU as standard makes Kano quite a nippy OS. The Kano team have removed a lot from the kernel, such as *gvfs, zeitgeist* and *gnome-pty-helper,* but they have also added improvements such as 'copies-and-fills', which enables faster RAM access. The team emphasise that they will push any improvements upstream to the Raspbian kernel and you can see the full list of included kernel modules and software installed at Github, where they house the project at **http://bit.ly/Kano_eVt**

On first boot Kano OS takes you through a simple exercise to introduce the user to the kit and spark interest: we are urged to follow the rabbit into the rabbit hole and then defuse a bomb by typing **startx**.

The user interface is heavily stylised with a mix of bright colours and shortcuts to applications that support the key purpose of the project, namely coding. Where Kano excels is promoting coding to kids, and it accomplishes this via a series of challenges and projects using several apps, such as *Sonic Pi, Pong / Snake* using Python and a fantastic *Scratch* like programming system called *Kano Blocks* which is used to program *Minecraft*.

Kano as a package is great for children of all ages, with an appealing mix of easy-to-assemble components and fun projects. The packaging, documentation and components all ooze quality and are tough enough to stand up to rough handling by children. The Kano team have created a great package and will be expanding the kit into a much larger range in the near future, starting with their own version of a Pi powered camera.

Features at a glance



Excellent packaging
The packaging is well thought-through: the kit will fit through a letterbox but still look great.

Projects for all levels
Kano comes with several great projects from old favourites *Pong* and *Snake* to *Sonic Pi* and *Minecraft*.

Verdict

Kano Computer Kit

Developer: Kano
Web: http://kano.me
Price: £119.99

Features	9/10
Performance	9/10
Ease of use	9/10
Value	6/10

» *A quality package full of great projects and components that will enrich learning for children of all ages.*

Rating 8/10

BitScope BS05

We probe and capture to find out how much more we can do with the smallest oscilloscope you've ever seen.

In brief...

» A miniature but highly capable mixed-signal USB oscilloscope, logic and spectrum analyser especially suited for Raspberry Pi based projects.

BitScope is an established Australian company which has been making oscilloscopes for over 15 years. Its latest effort, the BitScope Micro, is built specifically with the Raspberry Pi in mind: it's small, USB-powered and packed with features. It's also waterproof thanks to being encased in a clear plastic sleeve. While it may not look much like the clunky dual-beam oscilloscopes which your high school physics teacher was so precious about (sorry, Mr Wallace), it's actually capable of doing everything that they could do, plus a whole lot more. And it weighs a mere 12g.

Besides being able to capture two analogue scope channels (and hence create the famous Lissajous figures), it's also capable of performing frequency-domain analysis on them. What's more, it has six dedicated logic channels, capable of decoding serial, SPI, I2C and CAN bus protocols. One can even gain an additional two logic channels via the analogue channel trigger comparators. It's also a signal/pulse generator – the provided DSO software enables you to generate sinusoidal, square and triangular waves, with frequencies between 4 and 16kHz and amplitudes up to 3.3V. But with a little programming, the device can replay an arbitrary waveform defined by up to 1,024 points. By connecting (using one of the 10 helpfully provided grabber cables) either the L5 pin (for pulses) or the L4 pin (for waveforms) to one of the input channels, you can even plot the signal as it is generated.

» That's not an oscilloscope, this is an oscilloscope. As Paul Hogan might say.

While the software provided by BitScope is not much to look at, it is certainly comprehensive, and the company has clearly put a great deal of effort into making it run efficiently, particularly on the Pi. The main application, *BitScope DSO*, has all the controls you'd find on a bench oscilloscope, as well as some you wouldn't, such as various options for smoothing or decay-fading the drawn waveforms. The oscilloscope has an impressive 50Hz frame capture rate, which can be rendered in real-time. You can also download *Chart* (for data recording), *Logic* (for protocol and logic timing analysis) and *Meter* (for automated measurements or to use the probe as a glorified voltmeter). The BitScope software works, thanks to some clever design decisions, across the company's whole range of products, and packages are available for Mac, Windows, Raspberry Pi and Ubuntu. There is a generic Linux binary, too, as well as source code for the whole suite.

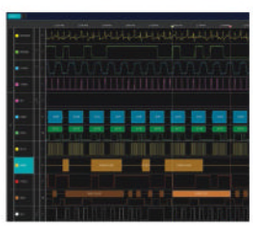
Virtually limitless

All of the BitScope products are built around the BitScope Virtual Machine, which uses a scripting system to manipulate registers. This means that it's entirely possible for hobbyists to write custom code, as befits their particular project's requirements, and run it directly on the hardware. However, there is also the well-documented BitLib API, which enables budding engineers to write their own code in C, C++, Python or Pascal. The software and API support remote measurement, making this an even more versatile tool, whether you're using it as an intrinsic part of your project or using it to diagnose problems therein.

The BS05 has been available in the US since April 2014, but at $150 (plus import duty for UK dwellers) it ended up a little pricey to get hold of. It was officially launched in the UK in October 2014, and you can now get it from Farnell (element14). Some potential users will possibly be put off by this still-substantial price tag, and hardcore electronics bods probably already have all the signal analysis kit they require. But if you're just getting into the game, then this is a great investment.

Features at a glance

DSO software
It might not be the prettiest just to look at, but it does much more than a stand-alone scope.



Logic analysis
Investigate mixed signals, decode protocols and record incoming data, all at the same time.

Verdict

BitScope BS05

Developer: BitScope
Web: www.bitscope.com
Price: £95

Features	8/10
Performance	9/10
Ease of use	7/10
Value for money	6/10

» *Its tiny size belies an impressive feature set, but if you're already 'scoped out, you might not need one.*

Rating 8/10

RASPBERRY Pi PROJECTS 2015

Python coding

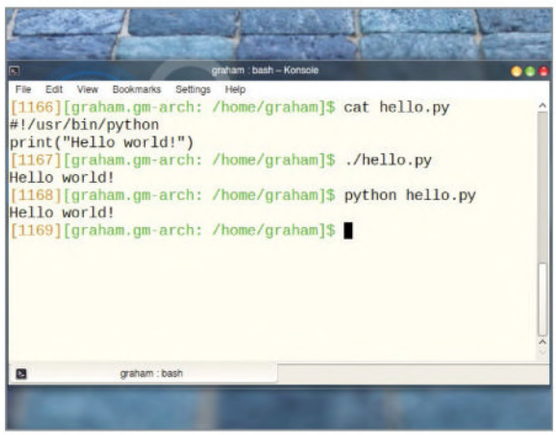
Get started with Python 2.4 154
What are types? 158
Using types to sort things 160
Functions for the win! 162
Accessing files 164
Get into Python 3.0 168
Creating fast-sort routines 170
AstroPi and Minecraft 174

Python: Get up and running

All the biggest Raspberry Pi adventures start with a single line of code, which is exactly where we're going to begin.

The Raspberry Pi was developed primarily to be an educational tool. But the main educational target wasn't the operating system or the hardware, it was as a cheap and accessible platform for learning how to program. What the Pi creators loved about those old home computers of the eighties was that you could turn them on and start typing code without having to install anything. You could start experimenting immediately, and through that experimentation learn a great deal about programming and how your computer worked in the process. If you wanted to access the innards of your machine, programming was the only method. Modern computers don't work in the same way – operating systems and command lines have replaced the BASIC interpreters of those old machines. But programming is still the best way of learning about your machine, and perhaps more importantly, it's one of the best ways of investing in skills that can be transferred to almost any platform, system and language. And you won't need any particular experience before trying it out. Because most of the instructions we use are entered from the command line, it makes sense to get familiar with this first, but there's really nothing to it. It also helps if you know what programming languages do, but even then the best way to learn is to throw yourself into it.

The official educational language for the Raspberry Pi is Python. It does have a weird name, but Python is a very well established language, already used by millions of developers. Python was chosen primarily because it's easy; it's simple to learn, simple to get started with and great for experimentation. But it's also an incredibly powerful language that is used for many, many ambitious projects, at Google and across many emerging technologies, so Python is also a serious tool. It's also cross-platform and contains many of the same concepts and paradigms used by almost every other language, which means spending time with Python will not only allow you to get the most out of your Raspberry Pi, but it will also put you on the first steps of a long adventure into computing, in just the same way those old 8-bit machines did back in 1983.



Python programs can be run in different ways – directly from the interpreter, by invoking the 'python' utility directly or by adding a shebang to a script file.

Python basics

Python will already be installed on your Raspberry Pi, but you can just as easily install it on to any other machine you might have, because there are versions for Windows, OS X and Linux, as well as many other systems, too. You can check to see if it's installed by opening up a terminal and typing the word **python**. If all goes well, you'll see a few lines of output and be left staring at **>>>**. This is the Python interpreter and it's part of what makes it such a good language to work with. Now the chances are you don't know what an interpreter is, but don't worry, there's nothing to it. All an interpreter does is translate your input as you type it and turns your words into actions. You could now, for example, type **print("Hello world!")**, and as soon as you've pressed the carriage return, the Python interpreter will decode your command and display

the 'Hello world!' text, which is exactly what we wanted it to do.

But most languages don't use an interpreter. Instead, you normally write your code into a text file and then use another command to build your code into the actions you require. You can do this with Python, too. Type **quit()** to exit from the Python interpreter, which is actually a call to the function within Python you use to quit your scripts, and open a text editor. From the command line, a useful editor is called **nano**, and if you type **nano hello.py** it will create a new text file called **hello.py** and place the cursor at the top of the screen waiting for you to type something. Now type **print("Hello world!")** then press [Ctrl] and [X] together. You'll be asked if you want to save the file, and you should reply with [Y]. You've just created a source file, and you can run the contents by typing **python hello.py**.

The **.py** postfix at the end of the filename is just there to let you know the file contains Python code, and isn't used by Linux at all. If you wanted to let the command line know your file was a Python script, you'd put something called a 'shebang' at the top of the file. For Python, this shebang looks like **#!/usr/bin/python**, and if you insert this into the first line of your code document, followed by **chmod +x hello.py**, you can now run your Python code without any further commands. Type **./hello.py** to execute the script.

It helps if you can use the command line, because it's easy to see what's happening, and really not that difficult to use. But there's another option that you may want to choose purely for convenience, and that's something called a development environment. A development environment is really just a text editor on steroids. There's always the usual input area, for example, but this is often augmented with features specific to the language you're developing in. Raspbian bundles an IDE called IDLE, which can be found on the default background, and it's worth playing around with this to see if you prefer it as an environment for writing your own scripts. IDLE defaults to interactive mode, and its great advantages are that it will highlight your code according to the types of statements you enter, as well as pop up contextual information for the various Python statements, objects, methods and variables used within your project, and handle all the indentation for you automatically. Type **value = 1** into IDLE, for instance, and when you next type **value.**, a menu will appear showing you what kind of methods can be applied to the data.

Statements

Now we've got the basics of running Python scripts out of the way, it's time to look a little closer at the language itself. We've already run a single command, **print("Hello world!")**, and this already tells us quite a lot about programming in Python. First is the **print** statement. It's called a statement because it's a word with a pre-defined function within Python, and there are just a handful of other statements. **print**, as you can see, takes the value passed to it within the brackets, often called the argument, and sends it to the standard output, which in nearly all cases will be your screen. This is an example of something called a function, and all it does is pass some data on to another piece of code to perform some action on it. **print** is one of the relatively few built-in statements because you usually spend your time writing your own functions or importing those written by other people. If we wanted to turn our 'hello world' line into a function, for instance, this is how it would need to look:

```
def helloworld():
        print("Hello world!")
        return
```

We'd recommend typing the above into the interpreter because it will mean you can play with the output immediately, but you could use any of the other methods we mentioned earlier. The most important part to notice, and probably the most important thing about Python, is that you must honour the indentation. The indentation is the column where a line of source code starts. The first line, where the function is defined, is positioned at the far left. But the following lines are 'tabbed' once to the right. The tab could also be a space, but either way, indentation is enforced by Python because it makes it easier for the programmer to see where a block of control starts and finishes. Now, if you type **helloworld()** on a new line, you'll see the output from the function, and you've created your first logical block of code.

Conditional statements

If you've not done any programming before, you might wonder how you build the logic into your own code. This is done in lots of ways, but at its most primitive level, it's accomplished through the use of conditional statements. This is where the flow of execution faces a decision point, and depending on the state of a parameter, the flow of execution can go off in many different directions. The foundation stone of conditional statements is **if**:

```
if something == 1:
        print("It's true!")
```

This is the correct format for an **if** statement in Python and it introduces several new ideas. The first is the word **something**. This is a variable, and we've made it up. A variable holds a value (that usually varies), and in this case the **if** statement is testing to see whether the **something**

An integrated development environment, such as IDLE here, is really just an extended text editor.

variable has a value of '1'. If it does, it prints the statement. If it doesn't, the **print** statement is skipped. Variables are obviously vitally important, and unlike with some other languages, they don't need to be of a pre-defined type before you use them. To make the above piece of code work, for instance, we need to give **something** a value:

```
something = 1
```

You might wonder why we've used only a single equals symbol in the assignment and yet used two next to each other with the **if** statement. It's because they're both doing different jobs, and it's important to make sure you know which is which. The double equals symbol is a comparison operator, and Python has a handful of others:

<	less than
<=	less than or equal
>	greater than
>=	greater than or equal
==	equal
!=	not equal

These can all be used interchangeably within your conditional statements to check the state of variables, whereas the single **=** symbol is only used to assign a value to a variable. For this reason, it's called an assignment operator, and while **=** is by far the most common, there are others:

+=	adds the two values together
-=	subtracts the value on the left from the value on the right
*=	multiplies the value on the left from the value on the right
/=	divides the value on the left from the value on the right
%=	assigns the modulus of the two values to the left

All of these allow you to actually manipulate the values held by parameters, just as you would in a mathematical formula. And like a mathematical formula, you can use parentheses to ensure your calculations are always being made in the order you're expecting them. For example:

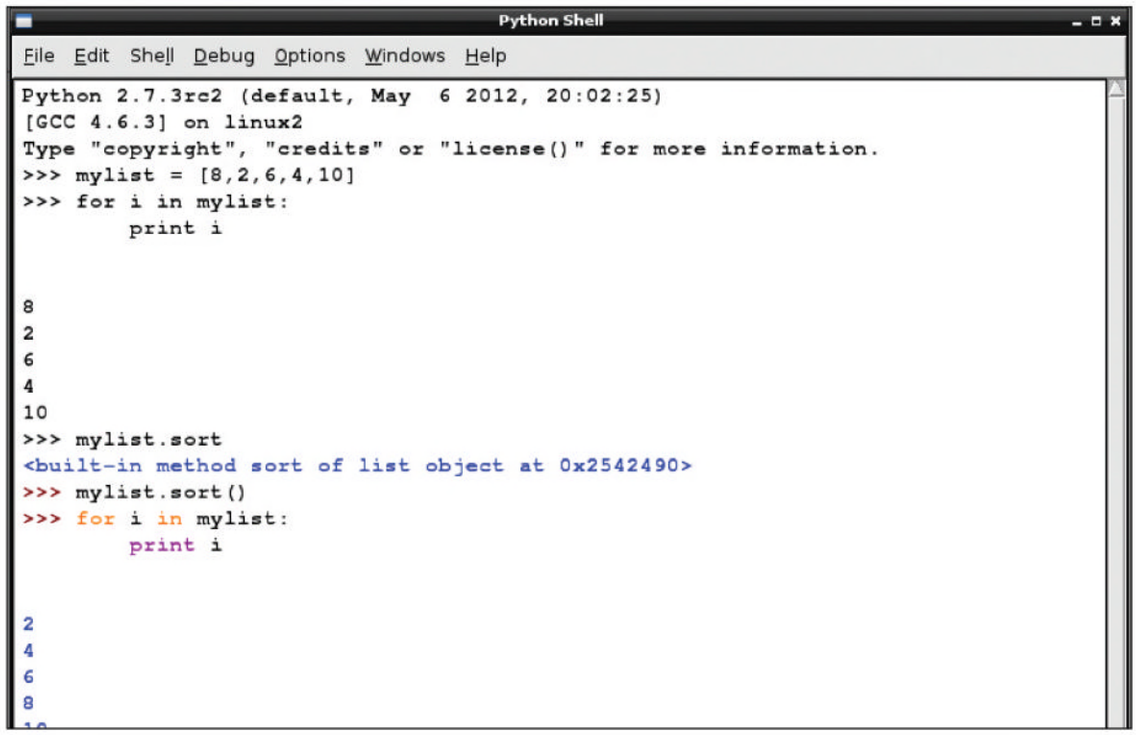
```
total = (10/2) * (20/10)
```

Type **print(total)** to see the result and you should see how the value was calculated. There are many other operators, but for the vast majority of projects, you won't have to worry about using them. And did you notice we slipped another important concept into our example? We gave **print** a variable name rather than an actual value, and even though this value was a number and not an alphanumeric character, **print** was still able to make sense of it. Variable names can always be used interchangeably with values, and most of the time you'll be dealing with variable names because the whole point of most programming projects is that variables are different.

The ability to pass an argument into a function through the parenthesis is also important, and you can do this in your own functions by including a variable name in the function definition. To make our hello world function totally generic, for example, we could change it to the following:

```
def helloworld(message):
        print(message)
        return
```

Now when you call **helloworld("This is a test")**, the value sent within the brackets is assigned to the 'message' variable within the new function, which is then printed out in exactly the same way as the original example. Thanks to the **print** command, you don't have to worry about what type your



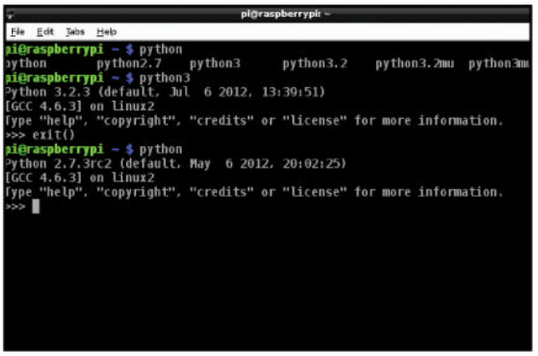
The 'for' loop is one of the most common statements. It's almost impossible to write a script without using one.

Which version of Python?

There are two versions of Python in wide use, version 2 and version 3 (also known as Python 3000) and this can cause some confusion. The most common is actually an older version, Python 2.x, which for many years was the default for any kind of development. This is the version that Raspbian defaults to using, and you can see this when you type **python** on the command line. The first line in the output should say something like the following:

```
Python 2.7.3rc2 (default, May  6 2012, 20:02:25)
```

This is saying that you're using Python 2.7.3 (release candidate 2), which is a recent version of the 2.x branch of development. But what you might find surprising is that version 3 is also installed by default, it's just that Raspbian chooses to make the **python** command point to the old one. If you type **pyth** on the command line followed by a press of the [Tab] key, the auto-complete feature will show you the other commands available and you'll see that 'python2.7' and 'python3.2' have their own executables. This means that if you want to run your script with the newer version, just invoke **python3.2** on the command line, but this will only be necessary when you need some of the newer version's specific features. Similarly, on the Raspbian desktop, you can choose between launching the plain IDLE development environment to play with Python 2.7, or IDLE3 if you want to try the third generation of the platform. Most of the time, you can safely ignore which version of Python you are using and many of our tutorials are written for the older version to ensure maximum compatibility. See page 166 for more on Python 3.



Two versions of Python are installed by default, and you can choose between them using the command line.

variables are before you pass them to the function, unlike with many other languages. But **type** is still important because it shapes how you write your functions and what you want them to do.

Types

Even though you don't have to assign a type to a variable before you assign a value to it, types are just as important within Python as they are with other languages. And that's because types can't normally be simply translated from one to another without some side effect. If you wanted to store a large floating point number, such as pi, to so many digits, converting this to an integer and then back to a floating point number would lose all the detail. This is important, and it means that things are easier when you have some idea of the type of 'types' your code is going to play with before you write the code, but it also unlocks extra Python functionality.

For example, we've been playing with the **string** type to hold our 'hello world' message, although we've not explicitly stated this anywhere. And when you assign a number to a variable, it will usually be an integer if the number doesn't include a decimal point, and a float if it does. Apart from types that hold data, there's another special type that holds lists of data. Sometimes called an array, these lists are incredibly useful if you want to keep a group of values together. Here's a quick example:

> **"Types are just as important within Python as in other languages"**

```
mylist = [8,2,6,4,10]
```

The above line will create a list of numbers and assign it to the variable called **mylist**. If you need to address individual items, you can do so with **mylist[0]**, which will reference the first point in the array. Arrays always start at 0, rather than 1.

Finally, we want to illustrate two other ideas before you head off into the world of Python tutorials. The first is the idea of loops. A loop is used to repeat the same piece of code, usually until some condition is matched. For example, we can loop through each element in the list we just created with the following **for** loop:

```
for i in mylist:
        print i

8
2
6
4
10
```

All this is doing is saying, for every element in the list, execute the following print instruction. **for** loops are used all the time for this kind of thing, and you'll quickly find them an essential part of your programming. Another even more fundamental part are called methods. These are functions that your types inherit. With our list, for example, one such method is **sort**, so adding the line **mylist.sort()** will change the order of the values within the elements of the array. If you now let the **for** loop run, you'll see the numbers are output in ascending order. Which ties up both of the final concepts we needed to cover before launching into some tutorials.

But don't forget that Python is massive, and there's a great deal of information available on the official website (**http://docs.python.org**), plus plenty of places where you can get support. So don't suffer in silence – if you run into trouble, just ask!

Code concepts: Types of data

Functions tell programs how to work, but it's data that they operate on. So let's explain the basics of data in Python.

In this tutorial we'll be covering the basic data types in Python and the concepts that accompany them. In the following pages, we'll look at a few more advanced topics that build on what we do here: data abstraction, fancy structures such as trees, and more.

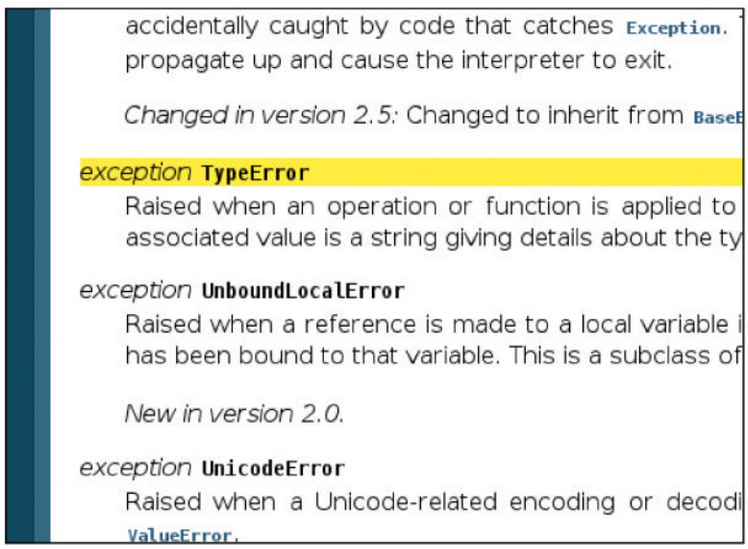
What is data?

In the world, and in the programs that we'll write, there's an amazing variety of different types of data. In a mortgage calculator, for example, the value of the mortgage, the interest rate and the term of the loan are all types of data; in a shopping list program, there are all the different types of food and the list that stores them – each of which was its own kind of data.

The computer's world is a lot more limited. It doesn't know the difference between all these data types, but that doesn't stop it from working with them. The computer has a few basic ones it can work with, and that you have to use creatively to represent all the variety in the world.

We'll begin by highlighting three data types: first, we have numbers. 10, 3 and 2580 are all examples of these. In particular, these are 'ints', or integers. Python knows about other types of numbers, too, including 'longs' (long integers), 'floats' (such as 10.35 or 0.8413) and 'complex' (complex numbers). There are also strings, such as **'Hello World'**, **'Banana'** and **'Pizza'**. These are identified as a sequence of characters enclosed within quotation marks. You can use either double or single quotes. Finally, there are lists, such as **['Bananas', 'Oranges', 'Fish']**. In some ways, these are like a string, in that they are a sequence. What makes them different is that the elements that make up a list can be of any type. In this example, the elements are all strings, but you could create another list that mixes different types, such as **['Bananas', 10, 'a']**. Lists are identified by the square brackets that enclose them, and each item or element within them is separated by a comma.

While we're looking only at basic data types, in real programs getting the wrong type can cause problems, in which case you'll see a TypeError.



Working with data

There are lots of things you can do with the different types of data in Python. For instance, you can add, subtract, divide and multiply two numbers and Python will return the result:

```
>>> 23 + 42
65
>>> 22 / 11
2
```

If you combine different types of numbers, such as an int and a float, the value returned by Python will be of whatever type retains the most detail. That is to say, if you add an int and a float, the returned value will be a float.

You can test this by using the **type()** function. It returns the type of whatever argument you pass to it.

```
>>> type(8)
<type 'int'>
>>> type(23.01)
<type 'float'>
>>> type(8 + 23.01)
<type 'float'>
```

You can also use the same operations on strings and lists, but they have different effects. The **+** operator concatenates, that is combines together, two strings or two lists, while the ***** operator repeats the contents of the string or list.

```
>>> "Hello " + "World"
"Hello World"
>>> ["Apples"] * 2
["Apples", "Apples"]
```

Strings and lists also have their own special set of operations, including slices. These let you select a particular part of the sequence by its numerical index, which begins from 0.

```
>>> word = "Hello"
>>> word[0]
'H'
>>> word[3]
'l'
>>> list = ['banana', 'cake', 'tiffin']
>>> list[2]
'tiffin'
```

Indexes work in reverse, too. If you want to reference the last

element of a list or the last character in a string, you can use the same notation with a **-1** as the index. **-2** will reference the second-to-last character, **-3** the third, and so on. Note that when working backwards, the indexes don't start at **0**.

Methods

Lists and strings also have a range of other special operations, each unique to that particular type. These are known as methods. They're similar to functions such as **type()** in that they perform a procedure. What makes them different is that they're associated with a particular piece of data, and hence have a different syntax for execution.

For example, among the list type's methods are **append** and **insert**.

```
>>> list.append('chicken')
>>> list
['banana', 'cake', 'tiffin', 'chicken']
>>> list.insert(1, 'pasta')
>>> list
['banana', 'pasta', 'cake', 'tiffin', 'chicken']
```

As you can see, a method is invoked by placing a period between the piece of data that you're applying the method to and the name of the method. Then you pass any arguments between round brackets, just as you would with a normal function. It works the same with strings and any other data object, too:

```
>>> word = "HELLO"
>>> word.lower()
'hello'
```

There are lots of different methods that can be applied to lists and strings, and to tuples and dictionaries (which we're about to look at). To see the order of the arguments and the full range of methods available, you'll need to consult the Python documentation.

Variables

In the previous examples, we used the idea of variables to make it easier to work with our data. Variables are a way to name different values – different pieces of data. They make it easy to manage all the bits of data you're working with, and greatly reduce the complexity of development (when you use sensible names).

As we saw above, in Python you create a new variable with an assignment statement. First comes the name of the variable, then a single equals sign, followed by the piece of data that you want to assign to that variable.

From that point on, whenever you use the name assigned to the variable, you are referring to the data that you assigned to it. In the examples, we saw this in action when we referenced the second character in a string or the third element in a list by appending index notation to the variable name. You can also see this in action if you apply the **type()** function to a variable name:

```
>>> type(word)
<type 'str'>
>>> type(list)
<type 'list'>
```

Other data types

There are two other common types of data that are used by Python: tuples and dictionaries.

Tuples are very similar to lists – they're a sequence data type, and they can contain elements of mixed types. The big difference is that tuples are immutable – that is to say, once you create a tuple you cannot change it, and that tuples are identified by round brackets, as opposed to square brackets: **('bananas', 'tiffin', 'cereal')**. Dictionaries are similar to a list or a tuple in that they contain a collection of related items. They differ in that the elements aren't indexed by numbers, but by 'keys' and are created with curly brackets: **{}**. It's quite like an English language dictionary. The key is the word that you're looking up, and the value is the definition of the word.

The Python interpreter is a great place to experiment with Python code and see how different data types work together.

With Python dictionaries, however, you can use any immutable data type as the key (strings are immutable, too), so long as it's unique within that dictionary. If you try to use an already existing key, its previous association is forgotten completely and that data lost for ever.

```
>>> english = {'free': 'as in beer', 'linux': 'operating system'}
>>> english['free']
'as in beer'
>>> english['free'] = 'as in liberty'
>>> english['free']
'as in liberty'
```

Looping sequences

One common operation that you may want to perform on any of the sequence types is looping over their contents to apply an operation to every element contained within. Consider this small Python program:

```
list = ['banana', 'tiffin', 'burrito']
for item in list:
    print item
```

First, we created the list as we would normally, then we used the **for... in...** construct to perform the **print** function on each item in the list. The second word in that construct doesn't have to be **item**, that's just a variable name that gets assigned temporarily to each element contained within the sequence specified at the end. We could just as well have written **for letter in word** and it would have worked just as well.

That's all we have space to cover in this article, but with the basic data types covered, we're ready to look at how you can put this knowledge to use when modelling real-world problems in the following pages.

Read the Python documentation to become familiar with some of the other methods that it provides for the data types we've looked at before. You'll find lots of useful tools, such as **sort** and **reverse**!

Code concepts: More data types

Learn how different types of data come together to solve a real problem as we count some words.

On the previous two pages, we introduced Python's most common data types: numbers (ints and floats), strings, lists, tuples and dictionaries. We demonstrated how they work with different operators and a few of their most useful methods. We didn't, however, give much insight into how they might be used in real situations. In this article, we're going to fix that.

We're going to write a short program that counts the number of times each unique word occurs in a text file. Punctuation marks will be excluded, and if the same word occurs in different cases (for example, **the** and **The**), they'll be taken to represent a single word. Finally, the program will print the results to the screen. It should look like this:

```
the: 123
you: 10
a: 600
...
```

As an example, we'll be using *The Time Machine*, by HG Wells, which you can download from Project Gutenberg, saving it in the same folder as your Python file under the name **timemachine.txt**.

As the program description suggests, the first thing we'll need to do is make the text accessible from inside our Python program. This is done with the **open()** function:

```
tm = open('timemachine.txt', 'r')
```

In this example, **open()** is passed two variables. The first is the name of the file to open; if it were in a different directory from the Python script, the entire path would have to be given. The second argument specifies which mode the file should be opened in – **r** stands for **read**, but you can also use **w** for write or **rw** for **read-write**.

Notice we've also assigned the file to a variable, **tm**, so we can refer to it later in the program.

With a reference to the file created, we also need a way to access its contents. There are several ways to do this, but today we'll be using a **for... in...** loop. To see how this works, try opening **timemachine.txt** in the interactive interpreter and then typing:

```
>>> for line in tm:
        print line
...
```

The result should be every line of the file printed to the screen. By putting this code in to a **.py** file, say **cw.py**, we've got the start of our Python program.

Cleaning up

The program description also specified that we should exclude punctuation marks, consider the same word but in different cases as one, and that we're counting individual words, not lines. As it stands, we've been able to read only entire lines as strings, however, with punctuation, strange whitespace characters (such as **\r\n**) and different cases intact.

Looking at the Python string documentation (**http://docs.python.org/library**), we can see that there are four methods that can help us convert line strings into a format closer to that specified by the description: **strip()**, **translate()**, **lower()** and **split()**.

Each of these are methods, and as such they're functions that are applied to particular strings using the dot notation. For example, **strip()**, which removes specified characters from the beginning and end of a string, is used like this:

```
>>> line.strip()
```

When passed with no arguments, it removes all whitespace characters, which is one of the jobs we needed to get done.

The function **translate()** is a method that can be used for removing a set of characters, such as all punctuation marks, from a string. To use it in this capacity, it needs to be passed two arguments, the first being **None** and the second being the list of characters to be deleted.

```
>>> line.translate(None, '!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~')
```

lower() speaks for itself, really – it converts every character in

```
200: but
215: with
216: for
242: at
265: as
280: me
352: had
368:
416: it
436: that
438: my
541: in
548: was
678: to
803: a
1151: of
1230: and
1241: i
2253: the
```

```
103: is
105: came
111: so
111: upon
112: into
112: little
112: one
114: not
115: all
118: he
121: have
121: they
122: from
122: there
129: his
132: then
132: you
137: on
151: this
157: were
196: time
```

Hardly surprisingly, our counting program, after being sorted, finds 'the' to be the most common word in *The Time Machine*, by HG Wells.

a string to lower-case. **split()** splits distinct elements inside a string in to separate strings, returning them as a list.
By passing an argument to **split()**, it's possible to specify which character identifies the end of one element and the start of another.

```
>>> line.split(' ')
```

In this example, we've passed a single space as the character to split the string around. With all punctuation removed, this will create a list, with each word in the string stored as a separate element.

Put all of this in the Python file we started working on earlier, inside the **for** loop, and we've made considerable progress. It should now look like this:

```
tm = open('timemachine.txt', 'r')
for line in tm:
    line = line.strip()
    line = line.translate(None, '!"#$%&\'()*+,-./:;<=>?@
    [\\]^_`{|}~')
    line = line.lower()
    list = line.split(' ')
```

Because all of the string methods return a new, modified string, rather than operating on the existing string, we've re-assigned the line variable in each line to store the work of the previous step.

Uniqueness

Phew, look at all that work we've just done with data! By using the string methods, we've been able to remove all the bits of data that we weren't interested in. We've also split one large string, representing a line, into smaller chunks by converting it to a list, and in the process got to the exact, abstract concept we're most interested in: words.

Our stunning progress aside, there's still work to be done. We now need a way to identify which words are unique – and not just in this line, but in every line contained within the entire file.

The first thing that should pop in to your head when thinking about uniqueness is of a dictionary, the key-value store we saw in the previous article. It doesn't allow duplicate keys, so by entering each word as a key within a dictionary, we're guaranteed there won't be any duplicates.

What's more, we can use the value to store the number of times each word has occurred, incrementing it as the program comes across new instances of each key.

Start by creating the dictionary, and ensuring that it persists for the entire file – not just a single line – by placing this line before the start of the **for** loop:

```
dict = {}
```

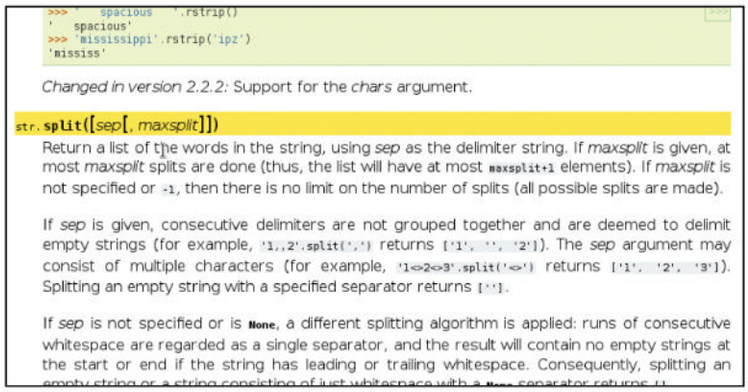
This creates an empty dictionary, ready to receive our words.

Next, we need to think about a way to get each word into the dictionary. As we saw last time, ordinarily a simple assignment statement would be enough to add a new word to the dictionary. We could then iterate over the list we created above (using another **for** loop), adding each entry to the dictionary with a value of **1** (to represent that it has occurred once in the file).

```
for word in list:
    dict[word] = 1
```

But remember, if the key already exists, the old value is overwritten and the **count** will be reset. To get around this, we can place an **if-else** clause inside the loop:

```
if word in dict:
    count = dict[word]
    count += 1
```



```
>>> '   spacious   '.rstrip()
'   spacious'
>>> 'mississippi'.rstrip('ipz')
'mississ'
```

Changed in version 2.2.2: Support for the *chars* argument.

`str.split([sep[, maxsplit]])`

Return a list of the words in the string, using *sep* as the delimiter string. If *maxsplit* is given, at most *maxsplit* splits are done (thus, the list will have at most `maxsplit+1` elements). If *maxsplit* is not specified or `-1`, then there is no limit on the number of splits (all possible splits are made).

If *sep* is given, consecutive delimiters are not grouped together and are deemed to delimit empty strings (for example, `'1,,2'.split(',')` returns `['1', '', '2']`). The *sep* argument may consist of multiple characters (for example, `'1<>2<>3'.split('<>')` returns `['1', '2', '3']`). Splitting an empty string with a specified separator returns `['']`.

If *sep* is not specified or is `None`, a different splitting algorithm is applied: runs of consecutive whitespace are regarded as a single separator, and the result will contain no empty strings at the start or end if the string has leading or trailing whitespace. Consequently, splitting an

Python's Standard Library reference, http://docs.python.org/library, is an invaluable source for discovering what methods are available and how to use them.

```
    dict[word] = count
else:
    dict[word] = 1
```

This is a little confusing because **dict[word]** is being used in two different ways. In the second line, it returns the value and assigns it to the variable count, while in the fourth and seventh lines, **count** and **1** are assigned to that key's value, respectively.

Notice, too, that if a word is already in the dictionary, we increment the **count** by 1, representing another occurrence.

Putting it together

Another data type wrestled with, another step closer to our goal. At this point, all that's left to do is insert some code to print the dictionary and put it all together and run the program. The **print** section should look like this and be at the very end of the file, outside of the line-looping code.

```
for word,count in dict.iteritems():
    print word + ": " + str(count)
```

This **for** loop looks different to what you've seen before. By using the **iteritems** method of the dictionary, we can access both the key (**word**) and value (**count**) in a single loop. What's more, we've had to use the **str()** function to convert **count**, an integer, into a string, as the **+** operator can't concatenate an integer and a string.

Try running it, and you should see your terminal screen filled with lines like:

```
...
other: 20
sick: 2
ventilating: 2
...
```

Data everywhere!

That's all we planned to achieve in this particular tutorial and it's actually turned out to be quite a lot. As well as having had a chance to see how several different types of data and their methods can be applied to solve a real problem, we hope you've noticed how important it is to select the appropriate type for representing different abstract concepts.

For example, we started off with a single string representing an entire line, and we eventually split this into a list of individual strings representing single words. This made sense until we wanted to consider unique instances, at which point we put everything in to a dictionary.

As a further programming exercise, why not look into sorting the resulting dictionary in order to see which words occur most often? You might also want to consider writing the result to a file, one entry on a line, to save the fruits of your labour.

Code concepts: Abstraction

Discover how creating abstractions can make your code more reliable and much easier to maintain.

In the previous two Code Concepts tutorials, we've been looking at data. First, we introduced some of Python's core data types, and then we demonstrated how they can be put to use when solving a real problem. The next data-related topic we want to consider is abstraction, but before we get on to that, we're first going to look at abstraction in general, and as it applies to procedures. So, in this tutorial we'll take a brief hiatus from data, before returning to it later.

Square roots

To get our heads around the concept of abstraction, let's start by thinking about square roots and different techniques for finding them. One of these was discovered by Newton, and is thus known as Newton's method.

It says that when trying to find the square root of number (x), we should start with a guess (y) of its square root; we can then improve that by averaging our guess (y) with the result of dividing the number (x) by our guess (y). As we repeat this procedure, we get closer and closer to the square root. In most attempts, we'll never reach a definite result, we'll only make our guess more and more accurate. Eventually, we'll reach a level of accuracy that is good enough for our needs and give up. Just to be clear about what's involved, take a look at the table below for how you would apply this method to find the square root of 2 (for example, x).

It's a lot of work just to find the square root of a number. Imagine if when you were in school, every time you had to find a square root you had to do all these steps manually. For instance, solving problems involving Pythagoras' theorem would be much more unwieldy.

> **"This is a very powerful technique that makes programming easier"**

Luckily, assuming you were allowed calculators at school, there's another, much easier method to find square roots. Calculators come with a button marked with the square root symbol, and all you have to do is press this button once – much easier. This second approach is what's known as an abstraction. When working on problems, such as those involving Pythagoras' theorem, we don't care how to calculate the square root, only that we can do it and get the correct result. We can treat the square root button on our calculator as a black box – we never look inside it, we don't know how it does what it does, all that matters is we know how to use it and that it gives the correct result.

This a very powerful technique that makes programming a lot easier, because it helps us to manage complexity. To demonstrate how abstraction can help, consider this Python code for finding the longest side of a right-angled triangle:

```
import math
def pythag(a, b):
    a2b2 = (a * a) + (b * b)
    guess = 1.0
    while (math.fabs((guess * guess) - a2b2) > 0.01):
        guess = (((a2b2 / guess) + guess) / 2)
    return guess
```

The first thing to note is that it's not in the least bit readable. Sure, with a piece of code this short, you can read through it reasonably quickly and figure out what's going on, but at a glance it's not obvious, and if it were longer and written like this, you'd have a terrible time figuring out what on earth it was doing. What's more, it would be very difficult to test the different parts of this code as you go along (aka incremental development, vital for building robust software).

For instance, how would you break out the code for testing whether or not a guess is close enough to the actual result (and can you even identify it?), or the code for improving a guess, to check that it works? What if this function didn't return the expected results – how would you even begin testing all the different parts to find where the error was?

Finally, there's useful code in here that could be reused in other functions, such as that for squaring a number, for taking an average of two numbers, and even for finding the square root of a number, but none of it is reusable because of the way it's written. You could type it all out again, or copy and paste it, but the more typing you have to do, the more obscure code you have to copy and paste, and the more likely mistakes are to make it in to your programming.

Let's try writing that code again, this time coming up with some abstractions to fix the problems listed above. We haven't listed the contents of each new function we've created, leaving them for you to fill in.

Finding a square root

Guess (y)	Division (x/y)	Average (((x/y) + y)/2)
1	2/1 = 2	(2 + 1)/2 = 1.5
1.5	2/1.5 = 1.33	(1.33 + 1.5)/2 = 1.4167
1.4167	2/1.4167 = 1.4118	(1.4118 + 1.4167)/2 = 1.4142

```
import math:
def square(x):
    ...
def closeEnough(x, guess):
    ...
def improveGuess(x, guess):
    ...
def sqrt(x, guess):
    ...
def pythag(a, b):
    a2b2 = square(a) + square(b)
    return sqrt(a2b2)
```

Here, we've split the code in to several smaller functions, each of which fulfils a particular role. This has many benefits.

For starters, how much easier is the **pythag()** function to read? In the first line, you can see clearly that **a2b2** is the result of squaring two numbers, and everything below that has been consolidated in to a single function call, the purpose of which is also obvious.

What's more, because each part of the code has been split into a different function, we can easily test it. For example, testing whether **improveGuess()** was doing the right thing would be very easy – come up with a few values for x and guess, do the improvement by hand, and then compare your results with those returned by the function.

If **pythag()** itself was found not to return the correct result, we could quickly test all these auxiliary functions to narrow down where the bug was.

And, of course, we can easily reuse any of these new functions. If you were finding the square root of a number in a different function, for instance, you could just call the **sqrt()** function: six characters instead of four lines means there's far less opportunity to make mistakes. One final point: because our **sqrt** code is now abstracted, we could change the implementation completely, but so long as we kept the function call and arguments the same, all code that relies on

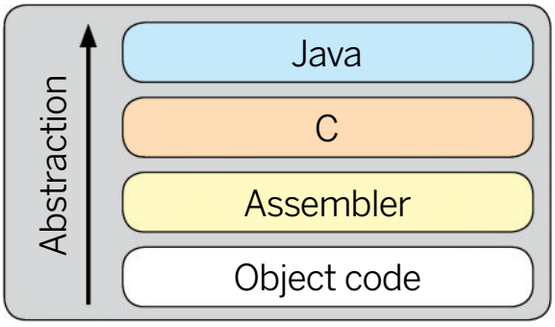
> **"This code can be improved by taking advantage of scope"**

it would continue to work properly.

This means that if you come across a much more efficient way of calculating square roots, you're not stuck with working through thousands of lines of code, manually changing every section that finds a square root – you do it once, and it's done everywhere. This code can be improved still further by taking advantage of scope. **closeEnough()** and **improveGuess()** are particular to the **sqrt()** function – that is to say, other functions are unlikely to rely on their services. To help keep our code clean, and make the relationship between these functions and **sqrt()** clear, we can place their definitions inside the definition of **sqrt()**:

```
def sqrt(x, guess):
    def closeEnough(x, guess):
        ...
    def improveGuess(x, guess):
        ...
    ...
```

These functions are now visible only to code within the **sqrt()** definition – we say they're in the scope of **sqrt()**. Anything outside of it has no idea that they even exist. This way, if we later need to define similar functions for improving a guess in a different context, we won't face the issue of colliding names or the headache of figuring out what **improveGuess1()** and **improveGuess2()** do.



There are layers of abstraction underneath everything you do on a PC – you just don't often think of them.

```
import math
def square(x):
    return x * x

def sqrt(x, guess):
    def closeEnough(x, guess):
        if math.fabs(square(guess) - x) > 0.01:
            return True
        else:
            return False

    def improveGuess(x, guess):
        return (((x / guess) + guess) / 2)

    while closeEnough(x, guess):
        guess = improveGuess(x, guess)
    return guess

def pythag(a, b):
    a2b2 = square(a) + square(b)
    return sqrt(a2b2, 1.0)

print pythag(2, 3)
~
```

Our final code for finding the longest side of a triangle is longer than what we had to start with, but it's more readable, more robust, and generally better.

Layers of abstraction

Hopefully, this example has demonstrated how powerful a technique abstraction is. Bear in mind that there are many layers of abstraction present in everything you do on a computer that you never think of.

For instance, when you're programming, do you know how Python represents integers in the computer's memory? Or how the CPU performs arithmetic operations such as addition and subtraction?

The answer is probably no. You just accept the fact that typing 2 + 3 in to the Python interpreter returns the correct result, and you never have to worry about how it does this. You treat it as a black box.

Think how much longer it would take you to program if you had to manually take care of what data went in which memory location, to work with binary numbers, and translate alphabetic characters in to their numeric representations – thank goodness for abstraction!

Code concepts: Files and modules

Did you know you can expand your library of functions and grab external data with just two lines of Python?

For the majority of programming projects, you don't get far before facing the age-old problem of how to get data into and out of your application. Whether it's using punched cards to get patterns into a 19th century Jacquard textile loom, or Google's robots skimming websites for data to feed its search engine, dealing with external input is as fundamental as programming itself.

And it's a problem and a concept you may be more familiar with on the command line. When you type **ls** to list the contents of the current directory, for example, the command is reading in the contents of a file, the current directory, and outputting the contents to another, the terminal.

> **"If the filesystem knows the file is being changed, it won't allow access"**

Of course, the inputs and outputs aren't files in the sense most people would recognise, but that's the way the Linux filesystem has been designed – nearly everything is a file. This helps when you want to save the output of a command, or use that output as the input to another.

You may already know that typing **ls >list.txt** will redirect the output from the command to a file called **list.txt**, but you can take this much further because the output can be treated exactly like a file. **ls | sort -r** will pipe (that's the vertical bar character) the output of **ls** into the input of **sort** to create a reversed alphabetical list of a folder's contents. The complexity of how data input and output can be accomplished is entirely down to your programming environment. Every language will include functions to load and save data, for instance, but this can either be difficult or easy depending on how many assumptions the language is willing to make on your behalf. However, there's always a logical sequence of events that need to occur. You will first need to open a file, creating one if it doesn't exist, and then either read data from this file, or write data to it, before explicitly closing the file again so that other processes can use it.

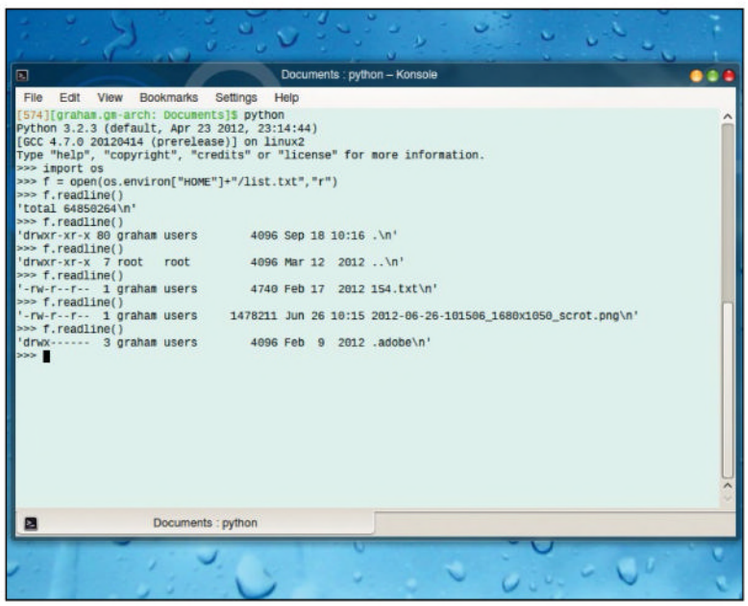
Most languages require you to specify a read-mode when you open a file, because this tells the filesystem whether to expect file modifications or not. This is important because many different processes may also want to access the file, and if the filesystem knows the file is being changed, it won't usually allow access. However, many processes can access a read-only file without worrying about the integrity of the data it holds, because nothing is able to change it. If you know anything about databases, it's the same kind of problem that you face with multiple users accessing the same table.

In Python, as with most other languages, opening a file to write or as read-only can be done with a single line:

```
>>> f = open("list.txt", "r")
```

If the file doesn't exist, Python will generate a 'No such file or directory' error. To avoid this, we've used the output from our command line example to create a text file called **list.txt**. This is within the folder from where we launched the Python interpreter.

When you read a file, most languages will step through its data from the beginning to the end in chunks you specify. In this example, we're reading a line at a time.



Environment variables

Dealing with paths, folders and file locations can quickly become complicated, and it's one of the more tedious issues you'll face with your own projects. You'll find that different environments have different solutions for finding files, with some creating keywords for common locations and others leaving it to the programmer to decide. This isn't so bad when you only deal with files created by your own projects, but it becomes difficult when you need to know where to store a configuration file or load a default icon. These locations may be different depending on your Linux distribution or if you're using Raspberry Pi, but with a cross-platform language such as Python, they will also be different for each operating system. For that reason, you might want to consider using environment variables. These are similar to variables with a

global scope in many programming languages, but they apply to any one user's Linux session rather than within your own code. If you type **env** on the command line, for instance, you'll see a list of the environmental variables currently set for your terminal session. Look closely, and you'll see a few that apply to default locations and, most importantly, one called **HOME**. The value assigned to this environmental variable will be the location of your home folder on your Linux system, and if we want to use this within our Python script, we first need to add a line to import the operating system-specific module. The line to do this is:

```
import os
```

This command is also opening a file, but not in the same way we opened **list.txt**. This file is known as a module in Python terms, and modules like this import functionality, including statements and definitions, so that a programmer doesn't have to keep re-inventing the wheel.

Modules extend the simple constructs of a language to add portable shortcuts and solutions, which is why other languages might call them libraries. Libraries and modules are a little like copying and pasting someone's own research and insight into your own project. Only it's better than that, because modules such as 'os' are used by everyone, turning the way they do things into a standard.

Setting the standard

There are even libraries called **std**, and these embed standard ways of doing many things a language doesn't provide by default, such as common mathematical functions, data types and string services, as well as file input/output and support for specific file types. You will find the documentation for what a library does within an API. This will list each function, what it does, and what it requires as an input and an output. You should also be able to find the source files used by the import (and by **#include** in other languages). On most Linux systems, for example, **/lib/python2.x** will include all the modules. If you load **os.py** into a text editor, you'll see the code you've just added to your project, as well as which functions are now accessible to you.

There are many, many different modules for Python – it's one of the best reasons to choose it over any other language, and more can usually be installed with just a couple of clicks from your package manager.

But this is where the ugly spectre of dependencies can start to have an effect on your project, because if you want to give your code to someone else, you need to make sure that person has also got the same modules installed.

> **"If you load os.py into a text editor you'll see the code you've just added"**

If you were programming in C or C++, where your code is compiled and linked against binary libraries, those binary libraries will also need to be present on any other system that runs your code. They will become dependencies for your project, which is what package managers do when you install a complex package.

The os module

Getting back to our project, the 'os' module is designed to provide a portable way of accessing operating system-dependent functionality so that you can write multi-platform applications without worrying about where files should be placed. This includes knowing where your home directory might be. To see what we mean, add the following piece of code to your project:

```
f = open(os.environ["HOME"]+"/list.txt","r")
```

This line will open the file **list.txt** in your home folder. Python knows which home folder is yours, because the **os.environ** function from the 'os' module returns a string from an environmental variable, and the one we've asked it to return is **HOME**. But all we've done is open the file – we've not yet read any of its contents. This might seem counter-intuitive, but it's an historical throwback to the way that files used to be stored, which is why this is also the way nearly all languages work. It's only after a file has been opened that you can start to read its contents:

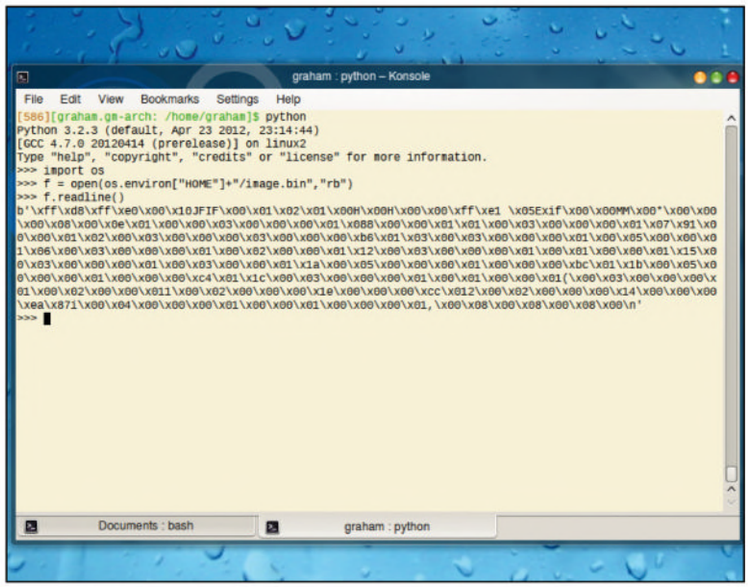
```
f.readline()
```

The above instruction will read a single line of the text file and output this to the interpreter as a string. Repeating the command will read the next line, because Python is remembering how far through the file it has read. Internally, this is being done using something called a pointer and this, too, is common to the vast majority of languages. Alternatively, if you wanted to read the entire file, you could use **f.read()**. As our file contains only text, copying the contents to a Python string is an easy conversion. The same isn't true of a binary file. Rather than being treated as text, the organisation of the bits and bytes that make up a binary file are organised according to the file type used by the file – or no file type at all if it's raw data. As a result, Python (or any other language) would be unable to extract any context from a binary file, causing an error if you try to read it into a string. The solution, at least for the initial input, is to add a **b** flag when you first open the file, because this warns Python to expect raw binary. When you then try to read the input, you'll see the hexadecimal values of the file output to the display.

To make this data useful, you need to do some extra work, but for now, make sure you close the open file, because this should ensure the integrity of your filesystem. As you might guess, the command looks like this:

```
f.close
```

And it's as easy as that! Now that you can work with files, you'll have much more flexibility in your coding exploits.



Binary files have no context without an associated file type and a way of handling them. Which is why you get the raw data output when you read one.

Python 3: Go!

Learning to program with Python needn't involve wall-climbing or hair-tearing – just follow our three guides to Python prowess.

After the basics let's delve further into Python. Learning a new programming language can be as daunting as it is rewarding, all the more so for those embarking on their maiden coding voyage. Exactly which language is most suitable for beginners can be debated until the bovine herd returns, but Python is certainly a worthy candidate. Python code is easy to follow, the syntax favours simplicity throughout and its enforcement of indentation encourages good coding styles and practices. Your distribution probably already has some version of Python installed, but for this tutorial we're targeting the newer version 3. You can check your Python version by opening up a terminal and typing:

```
$ python -V
```

Many distributions ship with both versions installed, but some (including Debian 8) still default to the 2.7 series. If the previous command indicated that this was the case, see what happens if you do:

```
$ python3
```

If that doesn't work, then you'll need to find the **python3** packages from your package manager. They should be straightforward to locate and install. If, on the other hand, the command succeeded, then you will, besides being availed of version information, find yourself at the Python 3 interpreter. Here you can execute code snippets on the fly, and here is where we will begin this tutorial. You can exit the interpreter at any time by pressing Ctrl+D or by typing:

```
>>> quit()
```

However, your first lines of code really ought to be more positive than that, so let's instead do a cheery:

```
>>> print('Hello world.')
```

Having pressed Enter and greeted our surroundings, we can get down to some proper coding. It's useful to accept and work with user input, which is accomplished as follows:

```
>>> name = input('State your name ')
```

Note the space before the closing quote. The command prints the given prompt and waits for something to be typed, which will appear alongside. This means that when the user starts typing their name (or whatever else they want), it is separated from our curt demand. We could also have used a **print()** line to display this prompt, and then instead used a blank input call as follows:

```
>>> name = input()
```

This means that input is accepted on a new line. Either way, the user's input is stored as a string in a variable called **name**. From the interpreter, we can see the values of any variable just by typing its name – however, we can also use the **print** function:

```
>>> print('Greetings', name, 'enjoy your stay.')
```

The **print()** function can work with as many arguments as you throw at it, each one separated by a comma. Note that we don't need to put spaces around our variable, as we did with the **input** function; by default, **print** separates arguments with a space. You can override the behaviour by specifying a **sep** parameter to the function – for example, the following code uses a dot:

```
>>> print('Greetings', name, 'enjoy your stay.',sep='.')
```

On the other hand, using **sep=''** instead gives no separation at all. Besides separators, **print()** also issues a newline character, represented by the string **'\n'** after the final argument. This can be easily changed, though, by specifying the **end** parameter, which is sometimes desirable, depending on the circumstances.

Besides welcoming people, we can use the interpreter as a calculator. It understands **+** (addition), **-** (subtraction), ***** (multiplication), **/** (division) and ****** (exponentiation), as well as many more advanced maths functions if you use the maths module. For example, to find the square root of 999 times 2015 and store the result in a variable **x**:

```
>>> import math
>>> x = math.sqrt(999 * 2015)
```

One of many things that Python helpfully takes care of for us is data types. Our first variable **name** was a string, while the recently defined **x** is a floating point number (or **float** for short). Unlike typed languages, where we would have to explicitly specify the type of a variable where it is defined, Python is smart enough to figure out that sort of information for itself, saving us the trouble. Also, Python variables do not have fixed types, so we could actually recast **x** above to an integer with:

```
>>> x = int(x)
```

Raspberry Pi-thon

If you're following this tutorial on a Raspberry Pi (1 or 2) and using the Raspbian distribution, there's good news: you don't need to perform any additional steps to get Python working. In fact, you get the pleasure of a ready-to-roll development environment called *IDLE*. You might find this more fun to work with than the command-line interpreter we use throughout this tutorial, and such things certainly become advantageous when working with larger projects.

IDLE enables you to write and edit multiple lines of code as well as providing a REPL (Read Evaluate Print Loop) for immediate feedback. To see this in action, select New Window, hammer out a few lines of code and then run them by pressing F5 or selecting Run Module from the Run menu.

Newer versions of Raspbian come with both versions 2 and 3 of Python, together with two separate *IDLE* shortcuts for starting each version. Previous Raspbian versions came with only the former, but if you've done an **apt-get dist-upgrade** recently, you'll probably find that the new version has been pulled in. But if you don't have Python 3 installed (in other words running **python3** returns **command not found**, then you can get it by opening *LXTerminal* and issuing the following commands:

```
$ sudo apt-get update
$ sudo apt-get install python3
```

Now **x** has been rounded down to the nearest whole number. We could convert it to a string as well, using the **str** type. Another important construct in Python is the list. A list is defined by using square brackets and may contain any and all manner of data – even including other lists. Here's an arbitrary example:

```
>>> firstList = ['aleph', 'beth', 3.14159265]
```

Items in our list are zero-indexed, so we access the first item, for example, with **firstList[0]** and the third one with **firstList[2]**. Some languages are one-indexed, which some people find more intuitive, but Python is not one of them. Strings are similar to lists (in that they are a sequence of characters) and many list methods can be applied to them. Thus we can get the first character of **name** with **name[0]**. We can also get the last character using **name[-1]** rather than having to go through the rigmarole of finding the string's length first, for which you would have to use the **len()** function. Strings are immutable – which means that once they've been defined, they cannot be changed – but this is fine because they can be redefined, copied or reconstructed depending on our purposes.

A particularly useful construct is **list** (or string) slicing. We can get the first two elements of our list by using **firstList[:2]**, or all but the first item with **firstList[1:]**. These are short forms of the more general slicing syntax **[x:y]**, which returns the substring starting at position **x** and ending at position **y**. Omitting **x** defaults to the beginning of the list, and omitting **y** defaults to the end. One can even specify a third parameter, which defines the so-called step of the slice. For example, the slice **[x:y:2]** gives you every second item starting at position **x** and ending at position **y**, or wherever the previous step lands. Again, this can be abbreviated – if you just want every second item from the whole list – to **[::2]**. Negative step sizes are also permitted, in which case our starting point **x** is greater than **y**, and omitting **x** defaults to the end of the list. Thus slicing our list above like so returns the reverse of that list:

```
>>> firstList[::-1]
```

Besides defining lists by explicitly specifying their members, there are a number of constructs available for defining lists that have some sort of pattern to them. For example, we can make the list **[0,1,2,3,4,5]** by using **list(range(6))**. In Python 3, the **range()** function returns an iterator (it doesn't contain any list items but knows how to generate them when given an index), so there is an additional function call to turn it into a list proper. Again, there is potential for 0-based grievances, because 6 is not in that list, and again it's something we simply have to get used to. Similar to slicing, the **range()** function can take optional arguments specifying the start value and step size of the range. This means that we can get all the odd numbers between 5 and 19 (including the former but excluding the latter) by using:

```
>>> twostep = list(range(5,19,2))
```

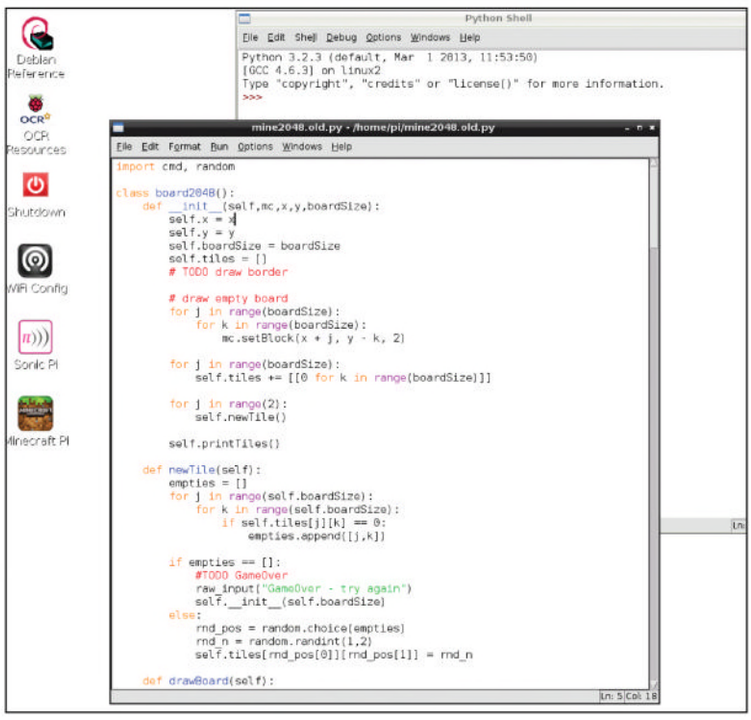
Negative step sizes are also supported, so we could count down from 10 to 1 with **range(10,0,-1)**.

Items can be inserted into lists – or removed from them – by using various methods. Methods in Python are properties of an object, and are called by suffixing said object with a dot, followed by the method name. So we could add **19** to the end of our previous list by using the **append()** method, such as:

```
>>> twostep.append(19)
```

We could also insert the value **3** at the beginning of the list with **twostep.insert(0,3)**. There are many other list methods, which you can peruse using **help(list)**. Help is available for any Python keyword and many other topics, and this can be a very useful resource when you get stuck.

When you start working with multiple lists, you might witness some rather strange behaviour, such as the diagram »



The *IDLE* environment is ideal *[ha, ha – Ed]* for developing larger Python projects. It's included in the Raspbian distribution, too.

» opposite illustrates. This seemingly inconsistent behaviour is all to do with how variables reference objects internally – variables are really just labels, so many variables can point to the same thing.

Very often, the canny coder makes extensive use of loops, wherein a codeblock is iterated over until some condition is met. For our first loop, we're using the **for** construct to do some counting. Note the indentation here; things inside a **for** loop (or indeed any other block definition) must be indented, otherwise you get an error. When you start such a block in the interpreter, the prompt changes from **>>>** to **...**. Python permits any number of spaces to be used for indentation, but consistency is key. We (alongside many others) like to use four spaces.

```
>>> for count in range(5):
...     print('iteration #', count)
```

Enter a blank line after the **print** statement to see the stunning results of each iteration in real time.

The variable **count** takes on the values **0** to **4** from the **range** iterator, which we met earlier. The result is that we issue five **print** statements in only two lines of code. If you want to iterate over different values for **count**, then you can use a different **range** or even a list – you can loop over any objects you like, not just integers.

Another type of loop is the **while** loop. Rather than iterating over a range (or a list), our wily **while** loop keeps going over its code block until some condition ceases to hold. We can easily implement **for** loop functionality with this construct, as shown here:

```
>>> count = 0
>>> while count < 5:
...     print(count)
...     count += 1
```

The **print** statement belongs to the loop, thanks to the indentation. The code still runs if you don't indent this last line, but in that case the **print** call is not part of the loop, so only gets executed at the end, by which time the value of **count** has reached **5**. Other languages delineate code blocks using brackets or braces, but in Python it's all colons and judicious use of white space. That **while** loop was rather trivial, so let's look at a more involved example:

```
>>> year = 0
>>> while year < 1900 or year >= 2015:
...     year = input("Enter your year of birth: " )
...     year = int(year)
```

We met the input statement on the first page, so this code just keeps asking you the same thing until an appropriate value is selected. We use the less than (**<**) and greater than or equal to (**>=**) operators conjoined with an **or** statement to test the input. So long as **year** has an unsuitable value, we keep asking. It is initialised to **0**, which is certainly less than **1900**, so we are guaranteed to enter the loop. You could change **1900** if you feel anyone older than 115 might use your program. Likewise, change **2015** if you want to keep out (honest) youngsters.

Whenever you deal with user input, you must accept the possibility that they will enter something not of the required form. They could, for example, enter their granny's name, or the first line of *The Wasteland,* neither of which can be interpreted as a year. Fortunately, the last line of our example does a good job of sanitising the input – if we try to coerce a string consisting of anything other than digits to an int, then we end up with the value **0**, which guarantees that we continue looping the loop. Note that the **input** function always returns a string, so even good input – for example, **'1979'** – needs some treatment; trying to compare a string and an int results in an error.

This Pythonic creed is worth studying. There are all kinds of bad programming habits, which are best to avoid from day zero.

```
import this
"""The Zen of Python, by Tim Peters. (poster by Joachim Jablon)"""

1  Beautiful is better than ugly.
2  Explicit is better than impl..
3  Simple is better than complex.
4  Complex is better than cOmp1|c@ted.
5  Flat is better than nested.
6  Sparse is better than dense.
7  Readability counts.
8  Special cases aren't special enough to break the rules.
9  Although practicality beats purity.
10 raise PythonicError("Errors should never pass silently.")
11 # Unless explicitly silenced.
12 In the face of ambiguity, refuse the temptation to guess.
13 There should be one-- and preferably only one --obvious way to do it.
14 # Although that way may not be obvious at first unless you're Dutch.
15 Now is better than ...                    never.
16 Although never is often better thanrightnow.
17 If the implementation is hard to explain, it's a bad idea.
18 If the implementation is easy to explain, it may be a good idea.
19 Namespaces are one honking great idea -- let's do more of those!
```

Grown-up coding

It's time for us to level up and move on from the interpreter – even though it's awfully handy for checking code snippets, it is not suitable for working on bigger projects. Instead we'll save our code to a text file with a **.py** extension, which we can either import into the interpreter or run from the command line. You need to find a text editor on your system – you can always use *nano* from the command line, but you may prefer to use a graphical one, such as Gnome's *gedit,* KDE's *kate* or the lightweight *Leafpad,* which comes with Raspbian. You may even wish to use a development environment such as *IDLE* (which comes with Raspbian and is designed especially for the language) but we don't require any of its many features for this introduction, so just a simple text editor will suffice.

Having found such a thing, enter the following listing:

```
#! /usr/bin/env python3
# My first Python 3 code.
# The # symbol denotes comments so anything you put here
doesn't matter

import datetime

def daysOld(birthday):
    today = datetime.date.today()
    ageDays = (today - birthday).days
    return(ageDays)


if __name__ == '__main__':
    weekdays = ['Monday', 'Tuesday', 'Wednesday',
'Thursday', 'Friday', 'Saturday', 'Sunday']

    birthyear = int(input('Enter your year of birth: '))
    birthmonth = int(input('Enter your month of birth: '))
    birthdate = int(input('Enter your date of birth: '))

    bday = datetime.date(birthyear, birthmonth, birthdate)
    dayBorn = weekdays[bday.weekday()]

    print('Hello, you were born on a', dayBorn)
```

```
    print('and are', daysOld(bday),'days old.')
```

Save the file as **~/birthday.py** (Python programs all have the **.py** extension). Before we analyse the program, you can dive straight in and see it in action. Almost, anyway – first we must make it executable by running the following command from the terminal, then we can call it into action:

```
$ chmod +x ~/birthday.py
$ ~/birthday.py
```

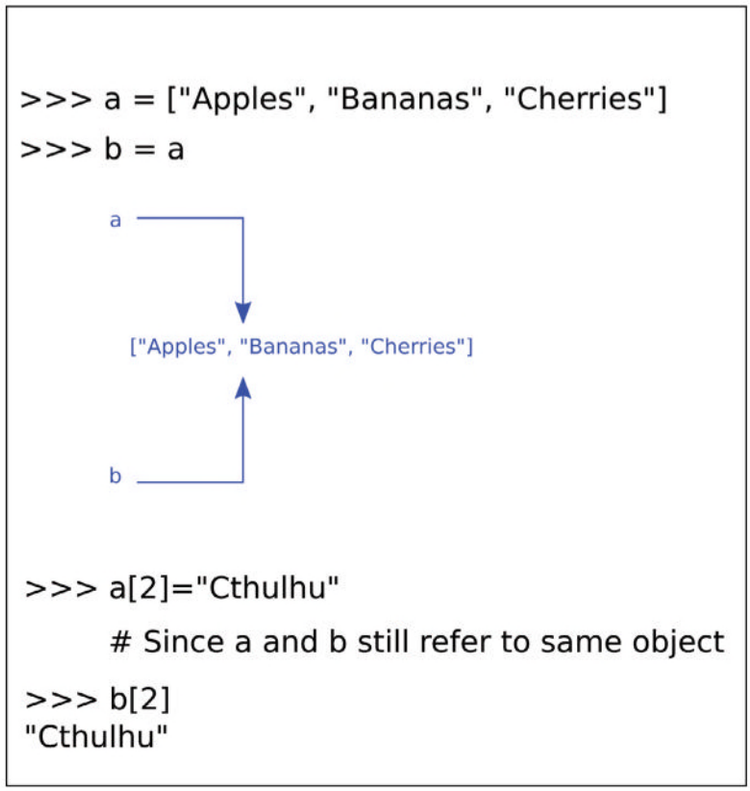
If you entered everything correctly, then the program works out how many days old you are and on which weekday you were born. Due to leap years, this would be some pretty fiddly calendrics if we had to work it out manually (though John Conway's Doomsday algorithm provides a neat trick). As well as having a simple core language structure, Python also has a huge number of modules (code libraries), which also strive for simplicity. This example uses the datetime module, which provides all manner of functions for working with dates and times, oddly enough.

We'll do a quick rundown of the code, because there are a couple of things that we haven't seen before. The first line just tells *Bash* (or whatever shell you use) to execute our script with Python 3. Next, after the comments, we import the datetime module and then begin a new codeblock. The **def** keyword is used to define a function – we've used plenty of functions in this tutorial, but now we're making our own. Our function takes one parameter, **birthday**, and uses the datetime module to do all the hard work, returning the number of days the user has been alive.

The strange **if** statement is one of Python's uglier constructions. The special variable **__name__** takes on the special value **__main__** when the code is run from the command line, as opposed to being **import**-ed into another program or the interpreter. We have written a bona fide Python module here, but when we import it, everything in the **if** block is ignored.

The next four lines are similar to what we've come across before. To keep things simple, we haven't bothered with list comprehension for the weekdays, and we trust the user to provide sensible dates. You are free to remedy either of these using your newfound skills.

The datetime module provides a special data type – **datetime.date** – and our next line (using the input that we have just harvested) sets our variable **bday** to one of this species. Date objects have various methods, including **weekday()**, which we use next. This returns an integer between **0** and **6**, with **0** corresponding to Monday and **6** to Sunday, so using this as an index into our list **weekdays** in the next line gives the required string. The final line calls our function from inside the **print()** function. Besides methods such as **weekday()** and **year()**, **date** objects also allow for general date arithmetic to be done on them. Therefore we can subtract two dates, which returns a **datetime.timedelta** object, which we can then convert to an integer using **days()** or **seconds()**.



You might run into this apparently spooky action at a distance when working with lists. It's not the work of Chthonic demons, however, it's just how Python rolls.

And so concludes the first of our Python coding tutorials. We've actually covered a good deal of the core language constructs, and with a little imagination it's not difficult to develop the ideas we've introduced into a larger and more exciting project. Help is never far away, though, and you can find some great Python tutorials on the web. The official tutorial (**https://docs.python.org/3/tutorial/**) is quite extensive, and you can find others at **http://tutorialspoint.com** and **www.codecademy.com**.

We also have two more excellent tutorials in this very publication. However, learning to code is much more about experimentation and dabbling than following instructions. No tutorial can teach this, but hopefully we've sufficiently piqued your curiosity that you will continue your coding adventures.

List comprehensions

Lists and the various constructs we've just introduced can join forces to form one of Python's most powerful features: list comprehensions. These are a great demonstration of Python's laconic concision. Consider the following example:

```
>>> daysShort = ['Mon', 'Tues', 'Wednes', 'Thurs', 'Fri', 'Satur', 'Sun']
>>> days = [j + 'day' for j in daysShort]
```

Very often, coders use short variable names, commonly **i**, **j** and **k**, for ephemeral variables, such as those which are used in loops or comprehensions. Our iterator variable **j** runs over the beginnings of the names of each day of the week, and then our list comprehension suffixes the string **day** – the addition operator (**+**) concatenates (in other words, joins) strings together. If you want a more arithmetical example, you could try:

```
>>> [j ** 3 for j in range(11)]
```

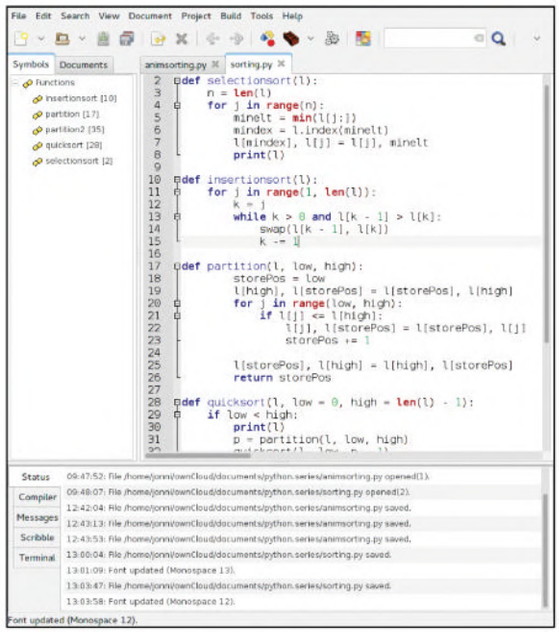
This returns a list comprising the cubes of the numbers from 0 to 10. Comprehensions can use all kinds of other constructs as well as **for** loops. For example, if we have a list of names, we could select only those beginning with 'Z' or those which have 'e' as their second letter by using:

```
>>> names = ["Dave", "Herman", "Xavier", "Zanthor"]
>>> [j for j in names if j[0] == 'Z' or j[1] == 'e']
['Herman', 'Zanthor']
```

It's worth noting at this point that to test for equality, we use the **==** operator. Be careful, because a single equals sign is only used for assignment, so doing something such as **if a = 5:** would result in an error.

Python: Sorting lists faster

We haven't got time for this, it's time to sort out that most fundamental of algorithm classes: sorting (sort of).



Geany **is a feature-packed text editor that's great for editing Python code. Code-folding and highlighting without the bloat.**

Within the human species, sorting things (putting them into some kind of order) is perceived as anywhere between a necessity (things disordered cause the most dire distress) and a dark ritual into which one should not delve. Computers, by comparison, universally prefer things to be sorted, because searching and indexing data is much easier if it is in order. Imagine how much less useful would be an unsorted telephone directory. As such, various sorting algorithms were developed early in the history of computer science. These enable a modern computer to happily sort a several thousand-strong iTunes (err, Rhythmbox) library in a matter of seconds, listing tracks with whatever ordering the user desires, be it lexicographical or chronological. In theory, one could even re-enact Rob Fleming's autobiographical ordering (the order in which he purchased them) from the Nick Hornby book *High Fidelity,* albeit several orders of magnitude faster.

Sorting may not be the most glamorous of programming topics, and indeed Python's built-in sorting methods will prove much faster than anything we program here, but it serves as a great introduction to various programming concepts. Furthermore, it enables us to show some of Python's advantages, including its no-nonsense syntax and the simplicity by which we can use graphics. To make things easier, we're going to work exclusively with lists of integers. All of the concepts apply equally well to any other kind of data, of course, but this way we have a well-defined order (ascending numerical) to aim for.

When challenged with the drudgery-filled task of, say, putting 52 cards into order, first by suit and then within each suit by face value, most people will flounder around with some haphazard piling and re-arranging, and eventually complete the task (or give up and find something better to do). Machines obviously need to be more systematic (and dedicated) in their approach. It's straightforward enough to come up with a couple of sorting algorithms that work, albeit not very quickly. We'll sum up some of these with pseudocode below. Pseudocode is a freeform way of expressing code that lies somewhere in between human language and the programming language. It's a good way to plan your programs, but don't be disheartened when the translation becomes difficult. We've actually overlooked a couple of

Quick tip

If you're using a new version of **Matplotlib** with Gnome, you might find that the animation described in the box displays only a blank window. A workaround is to add **pylab.pause(0.001)** after each **draw()** call.

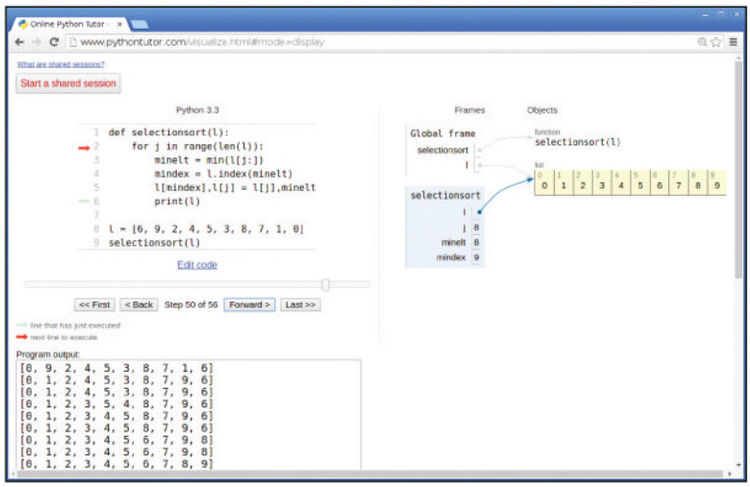
minor details here, but that will all be sorted out when we write our actual code.

```
SelectionSort(list):
    for each position j in list
        find minimum element after position j
        if minimum element is smaller than element at j
            swap them
```

Selection Sort is a fairly simple algorithm, but in simplicity lies beauty. The first loop iteration simply finds the smallest element in our list and places it, rightfully, at the beginning. We then start at the second item and see whether there is a smaller one further down the list, swapping if so. And so it goes on, until the work is done. Note that we swap elements here rather than insert them, this is so that we are only modifying two of our list items; inserting an item would require the whole list to be shunted down between the insertion and deletion points, which in the worst case would mean modifying every item in the list – a costly procedure if said list is long. We met **for** loops in our first tutorial, on page 166, and we can tell that **SelectionSort** will loop over items in the list at least once. But in each loop iteration we must find a minimum element, which for the first few loops (where **j** is small) involves checking most of the list items individually. We can approximate how long an algorithm takes to run by looking at these details. In **Selection Sort's** worst case we will pretty much have two loops – one nested inside the other, each going over our whole list. We use so-called big O notation to express the function's complexity as a function of the input size. In our case, we suppose our list has **n** items and the crude analysis above says that **Selection Sort** runs with complexity O(n^2). This isn't saying that Selection Sorting a list of one item will take one second (actually it takes 0 seconds) or a list of five items will take 25 seconds. What we can say is that the relationship between input size and sorting time, as **n** gets large, will be less than some constant (which depends on the machine) times n^2. This is slightly disappointing news – in layman's terms, **Selection Sort** gets pretty slow pretty quickly.

Here is another naive sorting algorithm, which is called **Insertion Sort**:

```
InsertSort(list):
    for each position j in list
        k = j
        while number at position k is less then that at position
k - 1
            swap numbers at positions k and k - 1
            k = k - 1
```



If you're looking for an excellent resource for learning and debugging your code, visit http://pythontutor.com. Here we can see Selection Sort in action.

This time we are upfront about our two nested loops, so again we can deduce O(n^2) complexity. We swap items that are not in order, and then move backwards down the list continuing to swap out-of-order entries. Once we get back to the beginning, we resume the outer loop, which proceeds towards the end. What is not immediately obvious, but is nonetheless interesting, is that this algorithm (like **Selection Sort**) has sorted the first **i** entries after that many iterations of the outer loop. As it proceeds, an out-of-order element may be swapped in, but by the end of the inner loop it will be in its right place, having trickled down there by a lengthy swap sequence. Time complexity notwithstanding, **InsertionSort**, in many situations, outperforms **Selection Sort**, the most obvious one being where the list is nearly sorted to begin with – in other words, list items are not too far from their rightful places. In this situation, the inner **while** loop terminates quickly, so the algorithm actually behaves with apparently linear, O(n), complexity.

There are a few other naive sorting algorithms that may be worthy of your perusal, including **Bubble Sort** (aka **Sinking Sort**) and its parallel version **Odd-Even Sort**, but we have much to get through so it's time to look at a grown-up method called **Quicksort**. **Quicksort** was developed in 1959 and, thanks to its swift O(n.log n) complexity, was quickly adopted as the default sorting algorithm in Unix and other systems. It is markedly more involved than the algorithms »

Quick tip

Very often, the simple algorithms outperform **Quicksort** for small lists. As such, you can create a hybrid sorting algorithm which uses **Quicksort** initially, but as the list is divided, reverts to **Selection Sort**, for example.

Visualising sorting algorithms

Python has some incredibly powerful modules that can help visualise the sorting algorithms discussed here. We're going to use the **Matplotlib** package, which can do all manner of plotting and graphing. It can be found in the **python3-matplotlib** package in Debian-based distros. There isn't room here to even scratch the surface of what's possible with **Matplotlib**, but we can show that with just a couple of extra lines, our sorting algorithms can be brought to life. Here we've modified our **insertionsort()** function, and added some boilerplate code to set everything up. You are encouraged to modify **selectionsort()** and **quicksort()** too – just add the **line.set_ydata()** and **pylab.draw()** calls after each line that modifies the list.

```
#! /usr/bin/env python3
import pylab
from random import shuffle

def insertionsort_anim(l):
    for j in range(1, len(l)):
        k = j
        while k > 0 and l[k - 1] > l[k]:
            l[k - 1], l[k] = l[k], l[k - 1]
            line.set_ydata(a)
            pylab.draw()
            k -= 1

pylab.ion() # turn on interactive mode
a = list(range(300))
x = list(range(len(a)))
shuffle(a)
line, = pylab.plot(x,a,'m.',markersize=6)
insertionsort_anim(a)
```

Save this file as **~/animsorting.py** and then do **chmod +x ~/animsorting.py** so that it can be run from the comfort of the command line. It's not the fastest way to display graphics (all the data is redrawn after each list update), but it's easy, and much more exciting than the **print** statements we used earlier.

» hitherto explored, but the rewards shall prove worthy of your attention. If **Quicksort** had a motto, it would be "Impera et divide" because at each stage it does some sneaky rearranging and then divides the list into two sublists and acts on them in isolation. (**Mergesort**, another algorithm, does the dividing first and the conquering after.) The list is rearranged and divided according to a pivot value – all items less than the pivot are moved to its left (nearer the beginning of the list), and all items greater than the pivot are moved to its right. This stage is called the partition operation; once it is complete, we work on the two sublists to the left and right of the pivot. And here begins the real fun, because we apply exactly the same methodology to our two smaller lists. Eventually, the division leads to sublists of size 0 or 1, which by definition are sorted.

Quicksort is, then, an example of a recursive algorithm – an algorithm that calls itself. This is a particularly tricky subject for the novice programmer to get their head around, mostly because the tendency is to imagine some kind of infinite chain of function calls, spiralling into the depths of oblivion and memory exhaustion. While it's entirely possible to program such a monster, a decent recursive algorithm ought to be good at how (and when) it calls itself. Typically, our recursive function calls involve smaller arguments (for example, a sublist rather than the whole list) and there are so-called base cases where recursion is not used. For **Quicksort**, the base cases are the aforementioned lists of length 0 or 1. Neglecting the details of the partition operation for a moment, our **Quicksort** pseudocode looks deceptively simple (excepting perhaps the recursion element). It takes, besides an unsorted list, two extra parameters (low and high). These specify the low and high indices between which we should sort. For the initial function call, these are zero and the length of the list, and for subsequent calls they are either from the low element to the pivot element or from the pivot element to the high element.

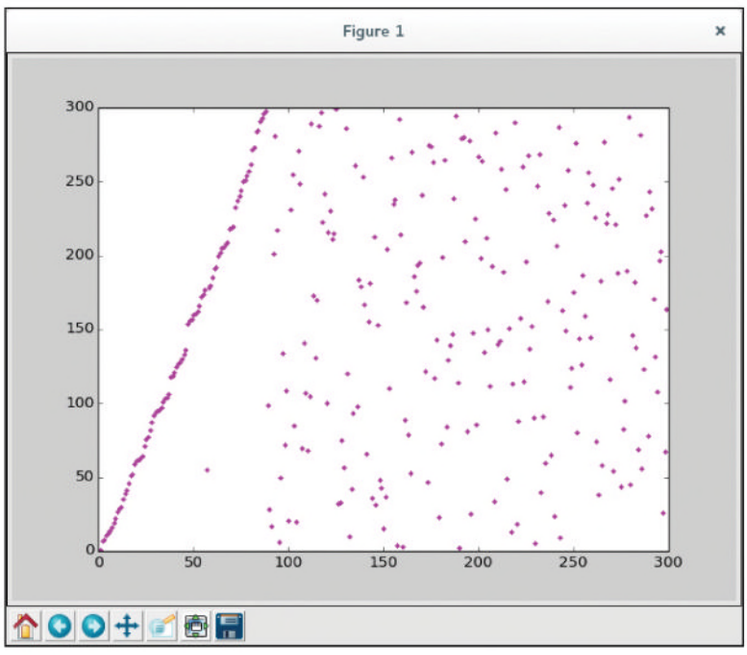
```
Quicksort(list, low, high):
  If low < high:
    p = Partition(list, low, high)
    Quicksort(list, low, p)
    Quicksort(list, p + 1, high)
```

The first thing that the **Partition()** operation must do is choose a pivot. Unfortunately, there is no straightforward way to do this – ideally one would choose a value that is going to end up about halfway in the sorted list, but this cannot be achieved a priori. In practice, you can often get away with choosing a random value here, or if a more systematic approach is preferred, the median of the first, middle and final elements in the list. We'll leave the pivot-choosing as a black box function **choosePivot()**, which returns the index of the pivot. So our partition operation, which also returns the index of the pivot, looks like:

```
Partition(list, low, high):
  pivotPos = choosePivot(list, low, high)
  pivotVal = list[pivotPos]
  # put the pivot at the end of the list
  Swap list[pivotPos] and list[high]
  storePos = low
  For each position j from low to high - 1:
    If list[j] <= pivotVal:
      Swap  list[j] and list[storePos]
      storePos = storePos + 1
  # put the pivot after all the lower entries
  Swap list[storePos] and list[high]
  Return storePos
```

That may seem complicated, but bear in mind all it's doing is arranging the list so that items less than the pivot are to its left (not necessarily in order). We have only one loop here, so the partition operation runs in O(n) time. In an ideal situation, the partition operation divides the list into two roughly equal parts, so at each stage of the recursion we half the list size. This is where the log(n) in the complexity comes from, since halving a list of size n about log-to-the-base-two of n times results in a singleton list, whereupon the recursion ceases. It is possible for **Quicksort** to perform much slower than stated, though – for example, if we start with an already sorted list and always use the last (greatest) element as a pivot, we are reduced to O(n^2) complexity.

› Elements undergoing Insertion Sort trickle one at a time from right to left. Watch it at night-time as an alternative to counting sheep.



Coding it up

It's time to translate our three algorithms into Python. We saw briefly how to define functions in the prequel, start the block with a **def** keyword, and then indent all the code that belongs to that function. In our pseudocode we referred to our list as, rather unimaginatively, **list**. Unfortunately, that's one of not terribly many reserved words in Python, so we'll instead use **l** (for llama). **Selection sort** then looks like:

```
def selectionsort(l):
  for j in range(len(l)):
    minelt = min(l[j :])
    mindex = l.index(minelt)
    if minelt < l[j]:
      l[mindex] = l[j]
      l[j] = minelt
```

For simplicity, we're doing an inplace sort here – we've modified the original list so we don't need to return anything. The Python code is quite different from the pseudocode, but much of this is cosmetic. We use slicing to find the minimum element on or after position **j** and then use the **index()** method to find that minimum's position in the list. The **if** statement is actually superfluous here (we've left it in to better match the pseudocode), because if the condition is false (which would mean the minimum element was at position **j**), then it would harmlessly swap a list item with itself. There's no explicit **swap** operation, so we have to do that manually in the last two lines. Incidentally, these can be further condensed to a single

Optional arguments and speed considerations

Because our **Quicksort()** function needs the **low** and **high** parameters, the initial call looks like **quicksort(l, 0, len(a) - 1)**. Python does allow default values for arguments to be specified – you simply suffix the argument with an **=**. However, we can't use, for example, **high = len(l)** because we're not allowed to use internal variables. The workaround would be to make the first part of the function look like:

```
def quicksort(l, low = 0, high = None):
    if high is None:
        high = len(l)
```

You need some reasonably-sized lists to see **Quicksort's** benefits – we discovered that it could sort 10,000 items in about 0.3 seconds, whereas the others took much longer, particularly **Insertion Sort**, which took about 14 seconds. Exact timing depends on the particular list, of course, but it's also worth noting that (much) faster implementations of these algorithms are possible. Lists in Python have their own **.sort()** method, which happily sort about 50,000 items instantly.

We've tried to use code here that's at once simple and as close to the given pseudocode as possible. In general, Python isn't a language known for its speed, unfortunately, particularly when working with lists, though many of its other functions and modules are implemented through its C API.

```
l[mindex],l[j] = l[j],l[mindex]
```

This means that the whole algorithm is done in four lines. Swapping variables without this neat Pythonic trick would involve setting up a temporary variable to hold one of the values. Save this file as **~/sorting.py** or similar, then start the Python 3 interpreter in your home directory. You should be able to import your module like so:

```
>>> import('sorting')
```

Assuming you didn't see any error messages or make any typos, you can then apply your algorithm to a shuffled list. We'll use the **random** module to shuffle a list from 0 to 9 and test this:

```
>>> import random
>>> l = list(range(10))
>>> random.shuffle(l)
>>> sorting.selectionsort(l)
>>> l
```

Voilà – a freshly sorted list. Of course, it's nice to watch the algorithm progress. So add a **print(l)** statement at the end of the loop, with the same indentation so it is still within said loop. To reload the module and apply the changes, we need to use the importlib module's **reload()** function, the standard **import** statement won't notice changes on already loaded modules. Or you can just exit (Ctrl+D) and restart the interpreter. The screenshot on page 171 shows the output for the list [6, 9, 2, 4, 5, 3, 8, 7, 1, 0]. We can see the list becoming sorted from left to right, and that the last stage of the loop doesn't do anything.

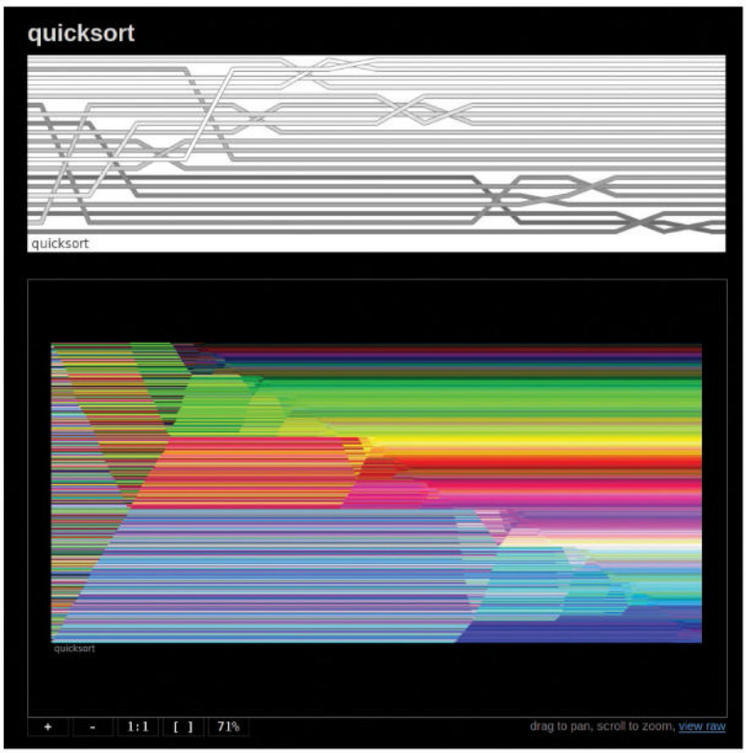
Moving on to **Insertion Sort**, we translate the pseudocode into Python code, which does not look drastically different. Add this to the **sorting.py** file.

```
def insertionsort(l):
    for j in range(1, len(l)):
        k = j
        while k > 0 and l[k - 1] > l[k]:
            l[k - 1], l[k] = l[k], l[k - 1]
            k -= 1
```

We start the outer loop at **1** so that the **l[k-1]** comparison in the **while** statement doesn't break things. We also add **k > 0** to the **while** condition because the loop should stop before **k** gets to zero, rather than try (and fail) to retreat past the beginning of the list. We've used our swapping shortcut from before and in the final line use the **-=** notation, which is short for **k = k - 1**.

Again, add **print** statements to either the **for** or **while** loops to see the algorithms progress. Smaller elements are swapped to the beginning of the list one place at a time until the list is sorted.

The **Quicksort** implementation is a bit more involved. For the partition operation, we're going to take the bold step of just using the first element for the pivot, so we don't need to worry about the first few lines of pseudocode.



Besides animations, sorting algorithms can be visualised as wave diagrams. Find out more at http://sortvis.org.

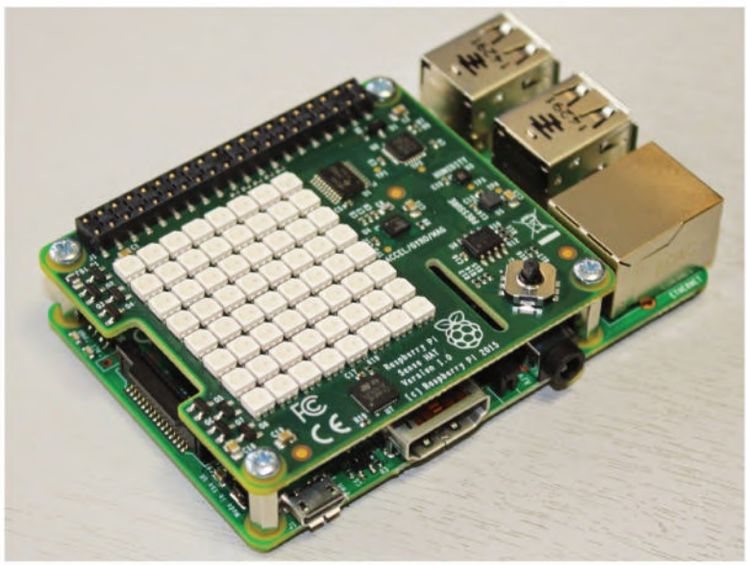
```
def partition(l, low, high):
    storePos = low
    l[high], l[storePos] = l[storePos], l[high]
    for j in range(low, high):
        if l[j] <= l[high]:
            l[j], l[storePos] = l[storePos], l[j]
            storePos += 1
    l[storePos], l[high] = l[high], l[storePos]
    return storePos

def quicksort(l, low, high):
    if low < high:
        p = partition(l, low, high)
        quicksort(l, low, p - 1)
        quicksort(l, p + 1, high)
```

Try visualising this using the guide in the box on page 171 – the **pylab.draw()** function should be called after the final swap in the **partition()** function, so that the pivot element is plotted in the correct place.

There are many more sorting algorithms, and research is ongoing to find more, but hopefully this introduction has shown you something of sorting as well as increased your Python skills. Turn the page for a more recreational lesson.

Python: Astro Pi & Minecraft

We conclude our Python exposition on a recreational note, combining Minecraft and space exploration.



Astro Pi sits snugly atop Terrestrial Pi, connecting via the GPIO pins.

We have published a fair few guides to the Python API for *Minecraft:Pi* edition for the Raspberry Pi before, see page 72. They're not critical for understanding this tutorial, but are certainly worth checking out if the idea of programming a voxel-based world appeals. Also worthy of attention, and also featuring excellent *Minecraft* tutorials, is Martin O'Hanlon's excellent website **http://stuffaboutcode.com**, whence comes the code used in this tutorial. The API, besides being a lot of fun, further edifies the Pi's status as an educational tool, because many young learners find programming much more intuitive when the results can be visualised three-dimensionally from the point of view of Steve, the game's intrepid hero.

One of the most exciting Raspberry Pi projects right now is the Astro Pi. This is a Hardware Attached on Top (HAT) expansion board, which features all manner of cool instrumentation: gyroscope, accelerometer, magnetometer, various environmental sensors, a multicoloured LED matrix and much more. It is available for free for school pupils and educators from the website **http://astro-pi.org**, if you can come up with a good coding idea, and will very soon be available to the general public. The most exciting thing about the project is that in November 2015, a Raspberry Pi, attired with an Astro Pi, will join ESA astronaut Tim Peake for a six-month stay on board the ISS. A competition is currently underway across UK schools in which entrants must devise Python code utilising the Astro Pi. The lucky winners will have their programs join Tim on his voyage to the ISS, where they will be run in zero-G. Owing to power requirements aboard the space station, the Pi cannot be connected to a proper display – its only means of visual feedback is the 64-LED array. This is multicoloured, though, so can still provide a fair amount of information.

The Astro Pi ships with a Raspbian SD card preloaded with all the required drivers and configs, but you can also download an installer for these. Open a terminal and type:

```
$ wget -O - http://www.raspberrypi.org/files/astro-pi/astro-pi-install.sh --no-check-certificate | bash
```

It takes a while to run on the original, single-core Pi, and you'll need to reboot for everything to work properly. While we're setting things up, make sure you've got *Minecraft* and the Python 3 API set up too. These are installed by default on recent Raspbian versions (there should be an icon on your desktop).If you don't have it, install it from the repositories:

```
$ sudo apt-get update
$ sudo apt-get install minecraft-pi
```

If you don't have an Astro Pi board, you can still have a lot of fun playing with the *Minecraft* Python API. Just start *Minecraft,* enter a new world, then Alt+Tab back to the

The mission

On Thursday 19 November 2015, two Raspberry Pis (one with the infrared filtered Pi-Noir camera module, and one with the normal Pi-Cam) will join astronaut Major Tim Peake aboard a Soyuz rocket, blasting off from Baikonur cosmodrome in (glorious nation of) Kazakhstan.

A nationwide coding competition was launched for primary and secondary school pupils, with the winners having Tim run their code when he arrives at the International Space Station. Space, industry and education sectors came together to make this exciting outreach project possible. A number of themes have been identified to inspire junior coders: Space Measurements, Spacecraft Sensors, Satellite Imaging and Remote Sensing, Space Radiation and Data Fusion.

Getting the Raspberry Pi and Astro Pi approved for cargo is a complex process – anything that is going to be connected to the ISS power supply has to be thoroughly tested. There are all manner of considerations, including electromagnetic interference, sharp edges assessment, and thermal testing – in zero-G conditions, air warmed by the Pi's CPU won't dissipate (as happens naturally by convection here on Earth), instead it will just hover around being hazardous. The flight case has been precision-engineered to comply with rigorous vibration and impact tests, as well as to provide much needed airflow. It also acts as a giant heatsink for the Pi's CPU – surfaces on the ISS are not allowed to exceed 45°C.

desktop and open a terminal window. From here, start Python 3 (**$ python3**) and enter the following:

```
>>> from mcpi import minecraft
>>> mc = minecraft.Minecraft.create()
>>> mc.postToChat("Hello world.“)
```

The API is capable of much more than this – we can change the player and camera position, get the ground level at a given set of co-ordinates, and get and set block information there, too. The following draws a rough and ready representation of a Raspberry Pi (it could be a Pi 2 or a Model B+), component by component:

```
>>> mc.setBlocks(-6, -3, -9, 7, -3, 11, 35, 13)
>>> mc.setBlocks(7, -2, -8, 7, -1, 5, 35, 15)
>>> mc.setBlocks(4, -2, 8, 6, 0, 11, 42)
>>> mc.setBlocks(0, -2, 8, 2, 0, 11, 42 )
>>> mc.setBlocks(-5, -2, 8, -2, 0, 11, 42)
>>> mc.setBlocks(-5, -2, 1, -2, -2, 1, 35, 15)
>>> mc.setBlocks(2, -2, -9, -1, -2, -9, 35, 15)
>>> mc.setBlocks(-6, -2, -7, -6, -2, -6, 42)
>>> mc.setBlocks(-6, -2, -2, -6, -2, 0, 42)
>>> mc.setBlock(-6, -2, 3, 35, 15)
>>> mcsetBlocks(0, -2, -2, 2, -2, -4, 35, 15)
```

The **setBlocks()** function fills the cuboid given by the first six numbers (the first three are co-ordinates of one corner and the next three are co-ordinates of the other corner). The next number decides the block type (42 stands for iron, while 35 stands for wool). Wool is a special block that comes in 16 colours; we don't need to specify a colour, but if we wish to do so, we can through an optional eighth parameter, which in general is referred to as block data. We've used the **setBlock()** function above – this just sets up a single block, so only requires one set of co-ordinates, plus the block type and optional block data arguments. You can find a thorough guide to the API at **www.stuffaboutcode.com/p/minecraft-api-reference.html**, but we'll describe what everything does as we use it.

For now, we'll turn our attention to coding on the Astro Pi. This connects to the Pi via the General Purpose Input/Output (GPIO) pins, but thanks to the astro_pi Python module, we don't need to worry about coding at the level of individual pins. The module provides some simple and powerful functions for querying sensors and manipulating the LED array. For example, we can display a suitably space-themed message in an extraterrestrial shade of green using the **show_message** function:

```
from astro_pi import AstroPi
ap = AstroPi()
ap.show_message(“E.T. phone home...”, text_colour = [0
,255,0])
```

The **text_colour** parameter here specifies the RGB components of the colour. As well as text, we can also work with the individual LEDs, using the **set_pixels()** function. This probably isn't the type of thing you want to do too much of by hand, though, because each LED requires its own colour triple. You can use this trick as a slight shortcut if you're only going to be working with a few colours, though:

```
X = [255, 0, 0]  # Red
O = [255, 255, 255]  # White

question_mark = [
O, O, O, X, X, O, O, O,
O, O, X, O, O, X, O, O,
O, O, O, O, O, X, O, O,
O, O, O, O, X, O, O, O,
O, O, O, X, O, O, O, O,
O, O, O, X, O, O, O, O,
O, O, O, O, O, O, O, O,
O, O, O, X, O, O, O, O
]

ap.set_pixels(question_mark)
```

It is probably more desirable to type this code into a text editor and then import it into Python, rather than work in the interpreter. So you need to put the first two lines from our E.T. phone home snippet at the top of the program, so that our **ap** object is correctly set up. You can then import it into the interpreter or run it from the command line if you prefer. As well as dealing with individual pixels, we can also load images »

Quick tip

We're assuming you use Python 3 throughout this series. This project still works with Python 2, but you may need to install the Pillow imaging library with **sudo pip install Pillow**.

Space case. This carefully machined and very strong case (it's made of 6063 grade aluminium, don't ya know) will house the Pi on its space odyssey.

directly on to the array by using the **load_image()** function as follows:

```
ap.load_image("~/astro-pi-hat/examples/space_invader.png")
```

Bear in mind the low resolution here – images with lots of detail don't work very well, but it's excellent for displaying pixel art and the like. Also, the LEDs are only capable of displaying 15-bit colour depth (five bits for red, six for blue, and five for green), so colours are dithered accordingly.

We can also query the Astro Pi's many inputs using the self-explanatory functions **get_humidity()**, **get_temperature()** and **get_pressure()**. The gyroscope, accelerometer and magnetometer are all integrated into a so-called Inertial Measurement Unit (IMU). We won't be doing any inertial measuring, but you can find all the relevant API calls in the documentation at **https://github.com/astro-pi/astro-pi-hat/blob/master/docs/index.md**. You'll also find some great examples in the **~/astro-pi/examples** directory, which show off everything from joystick input to displaying a rainbow pattern. Run these with, for example:

```
$ cd ~/astro-pi-hat
$ sudo python3 rainbow.py
```

The virtual interactive Astro Pi

The project we're going to undertake is actually the brainchild of student Hannah Belshaw, who submitted the idea to the Astro Pi competition. The (extensive) coding was done by Martin O'Hanlon, to whom we're ever so grateful. The project draws a Raspberry Pi (like we did earlier), but this one is equipped with the Astro Pi HAT.

Rather than have you copy out the code line by line, we'll download it straight from GitHub, and explain select parts of it. Don't expect to understand everything in the code right away – there's a lot of stuff we haven't covered – but once you get a couple of footholds, it'll serve as an excellent base for adding your own ideas. To download the code to your home directory, just open up a terminal and issue:

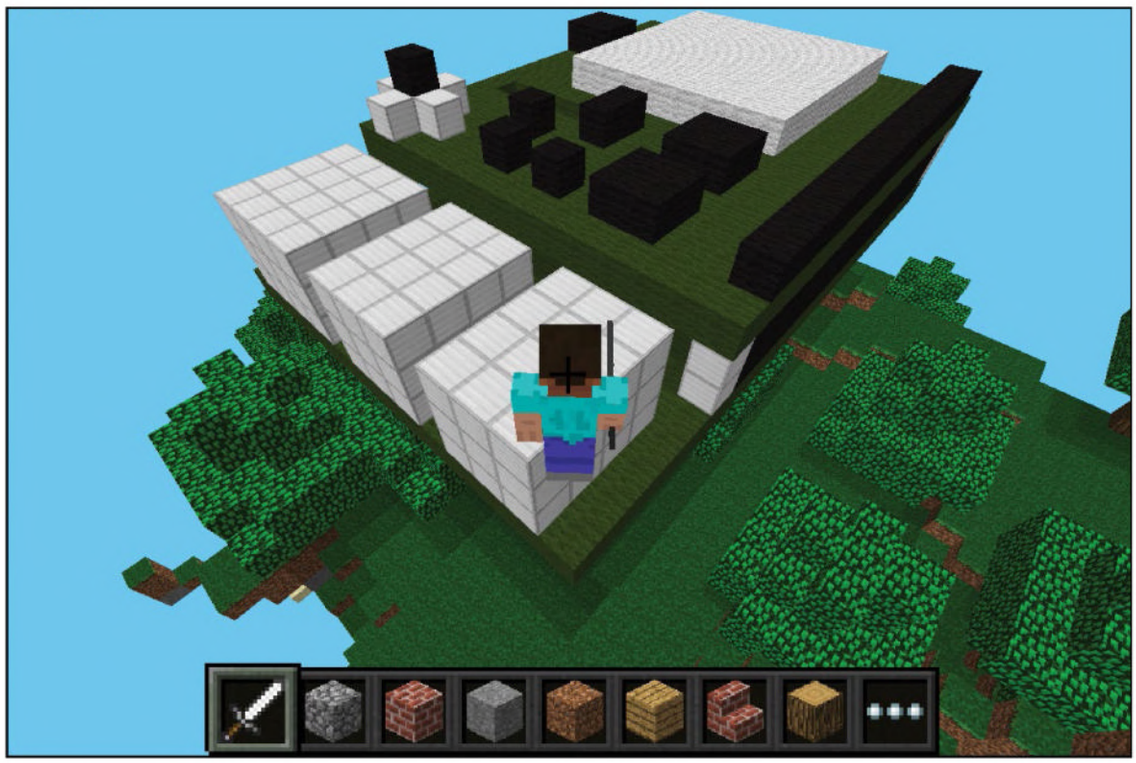
```
$ cd ~
$ git clone https://github.com/martinohanlon/MinecraftInteractiveAstroPi.git
```

This downloads all the required project files into the **~/MinecraftInteractiveAstroPi/** directory. Before we run it, make sure *Minecraft* is started. Also, if you've still got a Python session open from the beginning of the tutorial, then exit it now (Ctrl+D) so that things don't get confused. Now we're good to go. Because the Astro Pi libraries need GPIO access, and this in turn requires root privileges, we need to run it through **sudo**:

```
$ cd ~/MinecraftInteractiveAstroPi
$ sudo python mcinteractiveastropi.py
```

Everything in *Minecraft* is located by co-ordinates – triples of numbers which identify tiles by their x, y and z positions. The y co-ordinate measures height, x and z are harder to define, but they are nonetheless at right angles to each other and go in directions parallel to the ground plane. Normally, the player will start at position 0,0,0 (you can see your co-ordinates in the upper-left corner), in which case you might notice that it suddenly got dark in *Minecraft*-world. If you look up, you will see that this partial eclipse is caused by the appearance of a blocky contraption that our code has created. *Minecraft* allows you to fly by tapping and then holding Space. So fly up to this and you'll see an introduction message telling you to hit the virtual Astro Pi by right-clicking it with your sword. Left-clicking results in Steve hitting stuff properly, which would destroy the blocks that constitute our virtual device, so don't do that.

You can explore all the components of the Pi and the Astro Pi. Hitting the sensors, the latter avails you of the local pressure, temperature and humidity. You can also find out the orientation of the Astro Pi, which is given in terms of the pitch, roll and yaw angles. Bear in mind that you need to have calibrated the magnetometer for these readings to make sense. Instructions for this are at **www.raspberrypi.org/forums/viewtopic.php?t=109064**. You can also use the virtual joystick to move the virtual assemblage in three



Sword-wielding Steve stands ominously close to a USB port. Something about knives and toasters...

dimensions: the button toggles whether it moves in the horizontal or vertical plane.

Studying the code

If you open up the main project file, **mcinteractiveastropi.py**, in a text editor (or *IDLE*, if you prefer), you can catch a glimpse of how things work. We won't go into the specifics of object-oriented programming and classes; all you need to understand is that the board we draw in *Minecraft* is an instance of the object whose description begins with the **class** at line 19. This object has its own variables and functions, and can in fact be instantiated as many times as you like. The particular instance created by the code is called **mcap** and is defined way down on line 272.

Quite early on is defined a list called **LED_WOOL_COL**, which reconciles the 16 colours of wool available with something that the LED array can understand. When we hit a virtual LED block, it changes colour and the corresponding LED on the array does likewise. This is done in the **interact()** function, which in fact deals with all the components that can be hit. Rather than using the **setBlock()** functions, the virtual Raspberry Pi is constructed of objects from the custom **shapeBlock** class (which you can explore in the **minecraftstuff.py** file). These have a property called **tag**, which is a human-readable string describing the blocks in question. When a block is hit, this property is checked using a series of **if** and **elif** (short for **else-if** – in other words, if the previous condition didn't hold, try this one) statements and the appropriate action is taken. Some of the components just display an informational message using **mc.postToChat()**; some, such as the environmental and orientation sensors, query the Astro Pi hardware and display current readings; the joystick even moves the Pi around.

Because the sensor data often has lots of decimal places that we may not be interested in, we take advantage of Python's string formatting capabilities to round things appropriately. We can use curly braces as placeholders for variables, then use the **.format()** method to determine how they are displayed. For example, the following string (from line 157) formats the readings for the combined humidity and temperature sensor rounded to two decimal places:

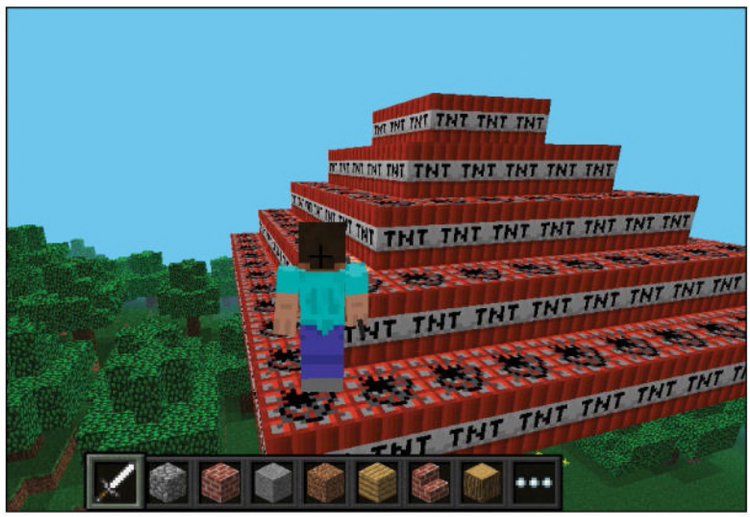
```
" humidity = {}, temperature = {}".format(round(humidity,2), round(temp, 2))
```

We've already met the slightly odd **if __name__ == "__main__":** construct in our first tutorial on page 166. It's used to discern whether the program has been executed from the command line or just imported. So that the player can keep exploring, this block of code contains a main loop which constantly polls for new events (the player hitting blocks):

```
    while(True):
        #each time a block is hit pass it to the interactive astro pi
        for blockHit in mc.events.pollBlockHits():
            mcap.interact(blockHit.pos)
        #keep reading the astro pi orientation data otherwise it goes out of sync
        ap.get_orientation()
        #sleep for a bit
        sleep(0.1)
```

This loop runs until it is forcefully interrupted, either by pressing Ctrl+C or by killing the *Minecraft* program. We pass the position of each block-hitting event to the **interact()** function, which figures out what has happened and what to do. The main loop is wrapped in **try** block, which is a construct usually used in conjunction with an **except:** for catching errors and exceptions. In this case, it is followed with a **finally:**, which is executed (whether or not an exception arises) when the **try:** completes. In our case, this ensures that we kill our running code with Ctrl+C, the virtual Pi is dematerialised, and the Astro Pi is reset.

This is a really great project to explore and extend, so go forth and do these things. Also great work, Hannah, for coming up the idea in the first place, and mad props to Martin for allowing us to use his code.



Bring TNT to life with the magic of the *Minecraft* Python API. This probably isn't a safe place to stand.

Minecraft: Pi Edition API, teleporting and TNT

Even if you don't have an Astro Pi, you can still have a lot of fun with the *Minecraft* API. Once you've imported the module, you need to set up the **mc** object (which handles all communication between Python and *Minecraft),* then you're good to go. We have already seen the **postToChat()** and **setBlocks()** functions, but there are a few more. For instance, we can move the player to the coordinates of our choosing with **player.setPos()**. This means that we can write and call a simple teleport function from within the interpreter:

```
def teleport(x=0, y=0, z=0):
    mc.player.setPos(x, y, z)
```

We've given some defaults for the arguments here, so that calling **teleport()** with no arguments takes Steve back to the origin (where the Astro Pi is drawn).

If you've had a look at the various blocks available, you'll no doubt have discovered TNT (block type 46). Sadly, there isn't a way to blow it up in-game. But fret not – mature adults can get their explosions by using some additional block data. Here's a function that'll figure out Steve's position and then assemble a substantial pyramid of live TNT above him:

```
def tnt_pyramid():
    x, y, z = mc.player.getPos()
    for j in range(5):
        xedge_lo = x - 5 + j
        xedge_hi = xedge_lo + 10 - 2 * j
        zedge_lo = z - 5  + j
        zedge_hi = zedge_lo + 10 - 2 * j
        mc.setBlocks(xedge_lo, y+ j + 2, zedge_lo, xedge_hi, y + j + 2, zedge_hi, 46, 1)
```

The magic is all in that final [code] 1 [/code]. This sets the TNT to be live, so that if you left-click it with your sword (or anything), then it begins to flash and pulsate ominously. At this point, you ought to migrate to a safe distance. Also, for the older, single-core Pis, the resulting chain reaction will be very taxing.