NEW!

THE ULTIMATE Raspberry Pi HANDBOOK

The *only* guide you need to get more from the amazing mini PC

180 PACKED PAGES

- Make a media centre
- Secure your home
- Build a robot

and more!

Covers all models of Raspberry Pi!

Future

THZ13 2016

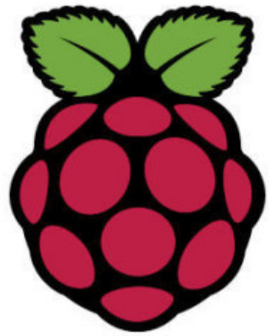
YOUR PERSONAL GUIDE TO THE UNIVERSE

OUT NOW!

DELIVERED DIRECT TO YOUR DOOR

Order online at **www.myfavouritemagazines.co.uk**
or find us in your nearest supermarket, newsagent or bookstore!

THE ULTIMATE Raspberry Pi HANDBOOK



THE ULTIMATE Raspberry Pi HANDBOOK

EDITORIAL TEAM

ART EDITOR
Fraser McDermott

IMAGES
Chris Hedley, Thinkstock, Future Photo Studio

EDITOR
Alex Cox

EDITOR-IN-CHIEF
Graham Barlow

CONTRIBUTORS
Les Pounder, Mayank Sharma, Jonni Bidwell, Neil Mohr, Neil Bothwick, Efrain Hernandez Mendoza

MANAGEMENT

EDITORIAL DIRECTOR
Paul Newman

GROUP ART DIRECTOR
Steve Gotobed

MARKETING

MARKETING MANAGER
Richard Stephens

CIRCULATION

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

PRINT & PRODUCTION

PRODUCTION MANAGER
Mark Constance

PRODUCTION CONTROLLER
Viv Calvert

LICENSING

SENIOR LICENSING & SYNDICATION MANAGER
Matt Ellis
matt.ellis@futurenet.com
Phone +44(0)1225 442244

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

Welcome!

…to the only guide you need to get the most out of any model of the ultimate mini PC.

It's amazing, really. The Raspberry Pi has been around for a scant few years, and it's gone from being a single low-powered board to an entire ecosystem of mini-computers with varying capabilities. It was the success of that amazing launch unit – one which now looks a little pathetic compared to its siblings – that set the course of the Pi family, and the most recent addition is perhaps the most impressive of all. The dinky baby Pi Zero – as big as a pack of chewing gum and, at £4, costing not a lot more – offers up so much potential thanks to its size and value that the only real question is 'what's next'?

Those with half an eye on certain corners of the internet may have picked up on a few rumoured board designs apparently leaked from the Pi Foundation which suggest a major power upgrade for the Raspberry Pi 3, but we're not putting any stock in them until anything is announced officially. And we don't need to: the HAT system, Hardware Attached on Top, means we can add all the capabilities we like to our Raspberry Pi units, expanding the potential way beyond simple coding.

We've gone in-depth in this title to ensure you get to see the best of the HATs available, as well as packing in a selection of the best tutorials and features from the expert minds behind Linux Format magazine. If you want to stay on the cutting edge of Linux and the Raspberry Pi, I strongly encourage you to subscribe – you'll find out how on p176. Cheers!

Alex Cox, Editor

The ULTIMATE HANDBOOK Manifesto

Ultimate Handbooks are designed to give you a complete guide to a device or piece of software you own. We aim to help you get more from the products you love and we guarantee you'll get the following from each book…

- A reference guide you can keep on your desk or next to your computer, and consult time and time again.
- New skills you can take with you through your life and apply at home or even in the workplace, whenever you need them.
- Expert advice to help you do more with your hardware and software – from solving new problems to discovering new things to try, we'll show you the best ways to do everything you might want.
- Clear recommendations for other products, accessories and services you can use with your device or software to get the best possible results.
- Advice you can take everywhere with you – thanks to the free digital edition of this book which you can download and read on your tablet, smartphone or laptop. See page 178 for more details.

How are we doing? Email **techbookseditor@futurenet.com** and let us know if we've lived up to our promises!

THE ULTIMATE Raspberry Pi HANDBOOK

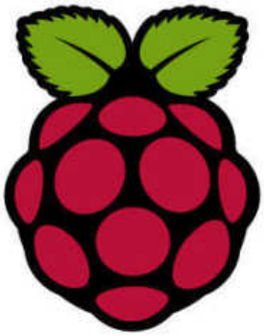
It's time you got more from your amazing Raspberry Pi, we're here to guide you into a world of fun projects and engaging learning.

The basics

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

Hardware

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



Top projects

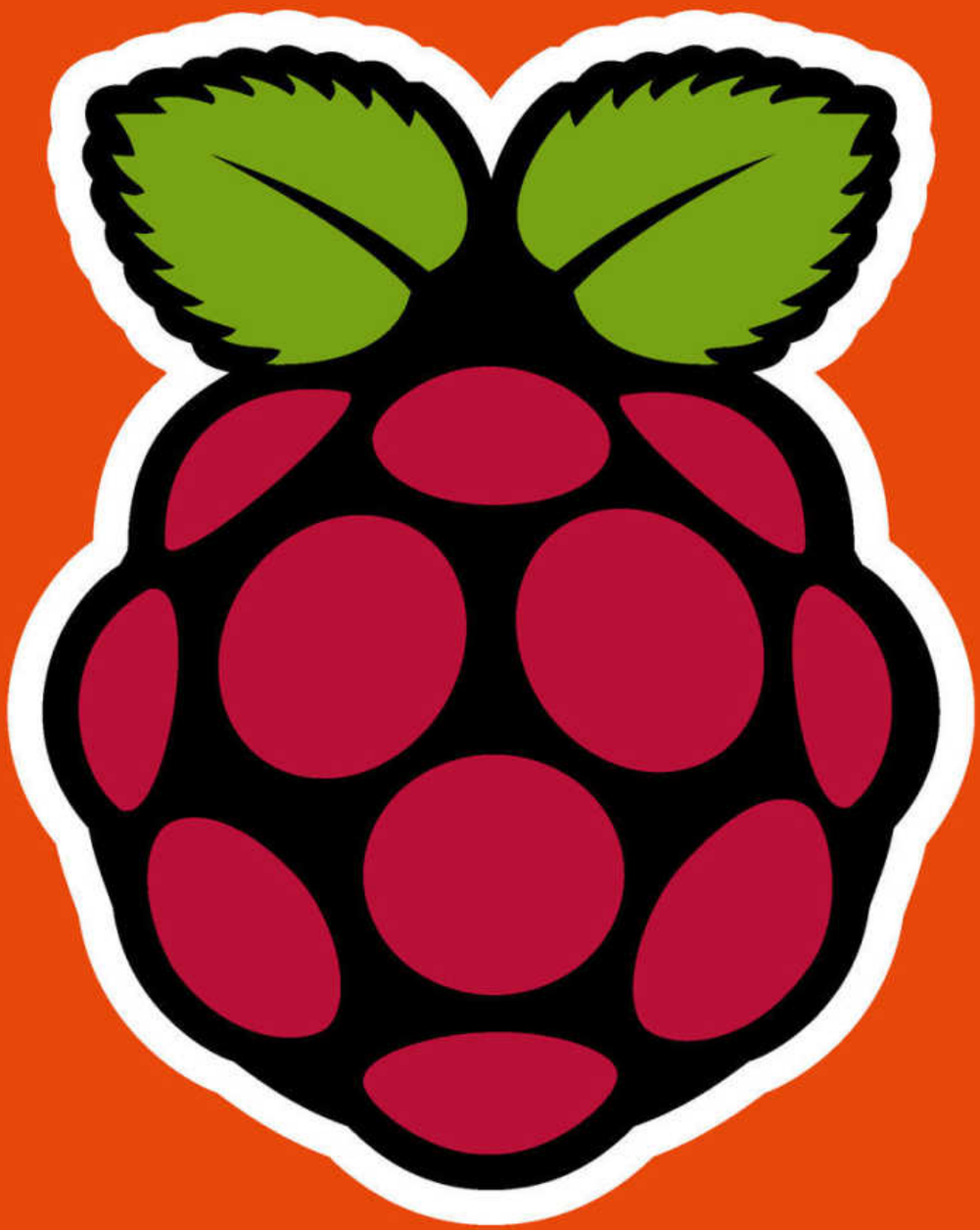
Let's dive straight into our Pi projects! There's a ton of fun to be had here.

Linux skills

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

Coding

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



The basics

Just getting started?
Here's what you need to know.

Raspbian: Easy

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

Step-by-step: Windows

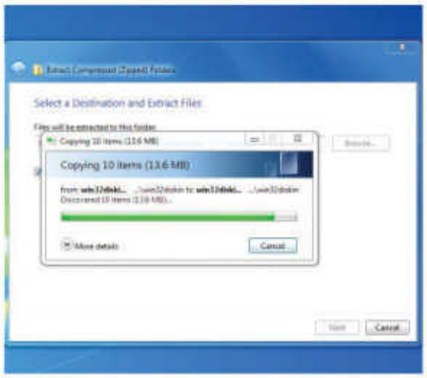
1 Download the tools

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

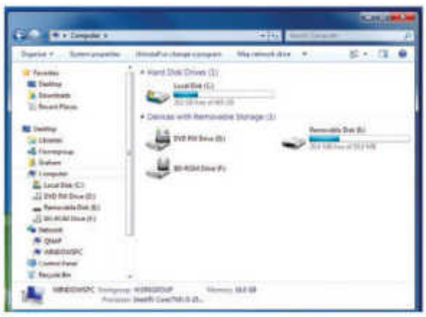
It doesn't need to be installed, because the download is a ZIP file which, when uncompressed with a right-click > Extract All, contains everything needed to run the application directly. It should also go without saying that you'll need a copy of the Raspbian image file for writing to your SD card. But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

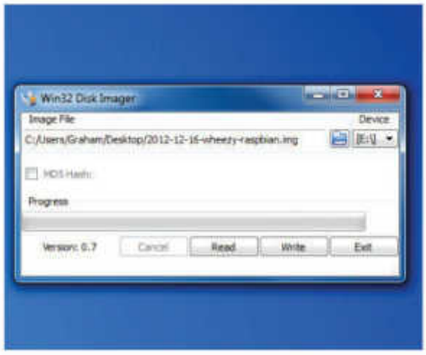


Windows displays the size of the mounted partition and not the entire storage device.

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

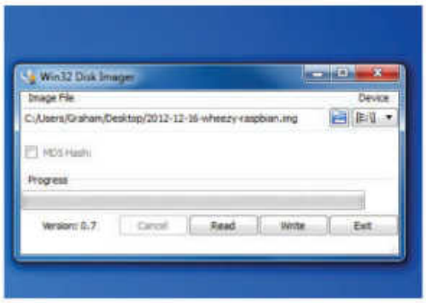
4 Write the data

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

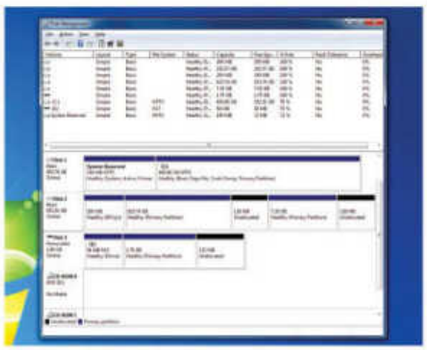
Before putting your freshly made SD card into your Raspberry Pi, it's worth having a quick check to make sure the write process worked as expected. It's also a good way of learning a little about how the SD card has been formatted. You could look again at your computer's device overview to check that 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.

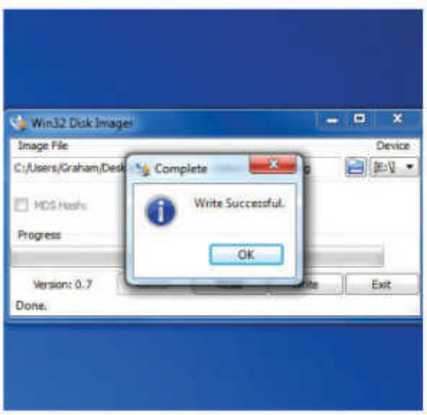
Selecting the top result will open the *Disk Management* application, and within its main window you can see details of all the storage connected to your machine. Locate your SD card volume – ours was E – and look at its partitions in the table below. You should see three – one for booting, which is 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.

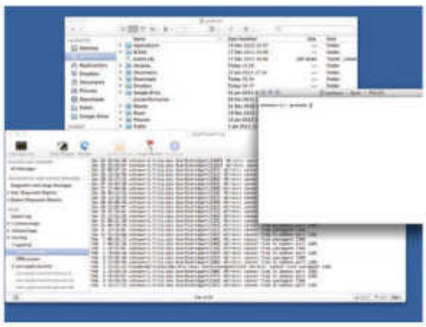
Step-by-step: Apple

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

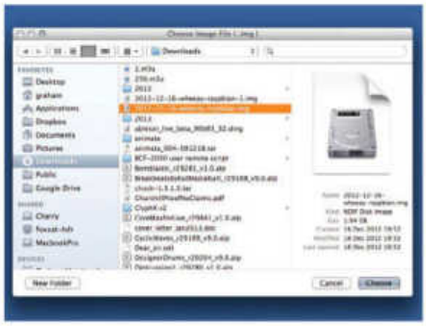
From OS X, like Linux, the standard way of installing the Raspbian operating system on to your SD card is from the command line. If you want to try this method, follow the instructions for Linux but replace the device name with those from OS X. But there's a safer, easier option, and that's using a graphical 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.



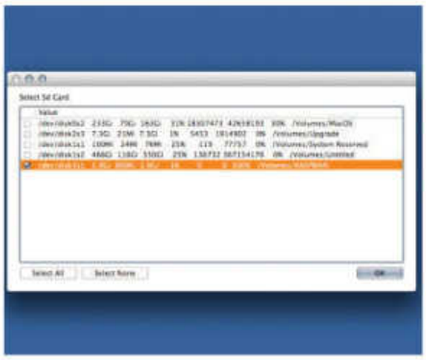
Use the file requester to locate the Raspbian image file and click on 'Choose'.

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.

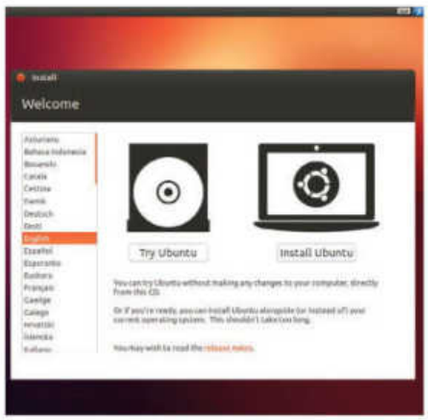
Raspbian: Easy

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.

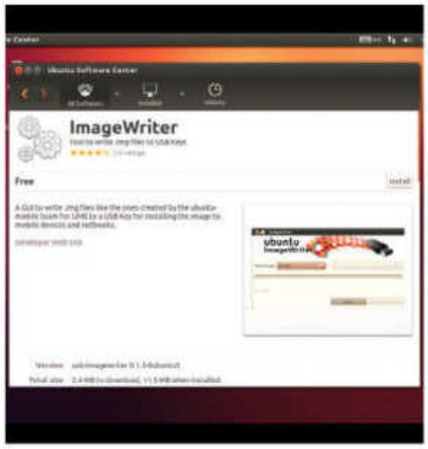
2 Be prepared

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.

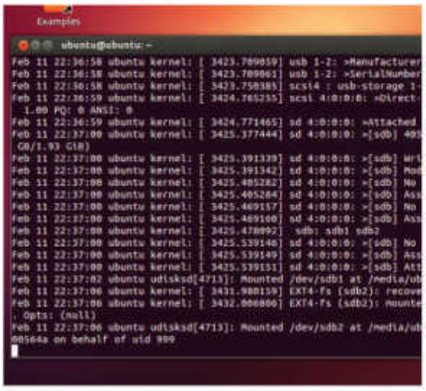
5 Test the SD card

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.

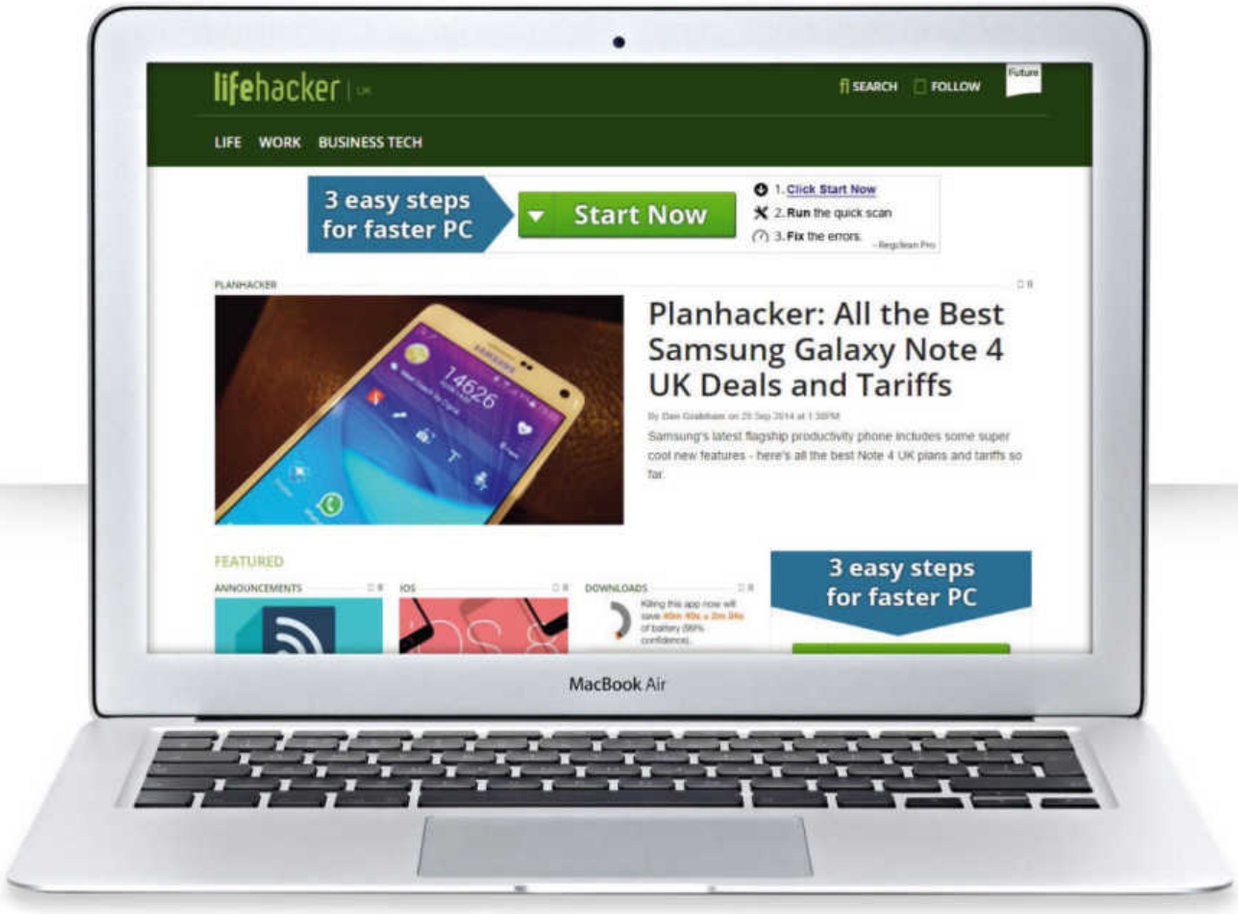


› You can cancel the dd command mid-way through by pressing [Ctrl] and [C] together.

Future

lifehacker | UK

Helping you live better & work smarter



LIFEHACKER UK IS THE EXPERT GUIDE FOR ANYONE LOOKING TO GET THINGS DONE

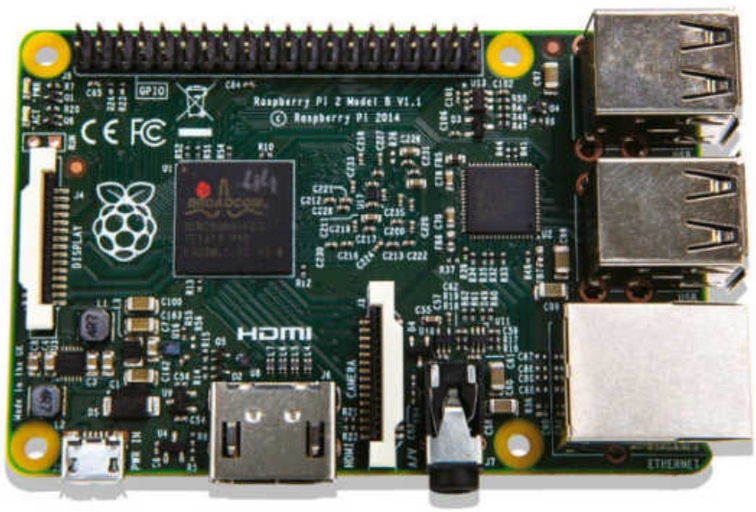
- Thousands of tips to improve your home & workplace
- Get more from your smartphone, tablet & computer
- Be more efficient and increase your productivity

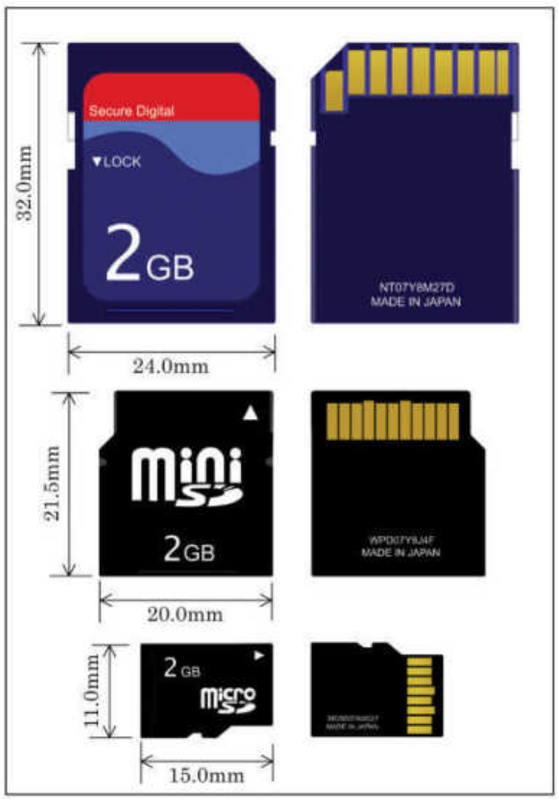
www.lifehacker.co.uk

twitter.com/lifehackeruk facebook.com/lifehackeruk

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 4.3GB (gigabytes) to install the current Rasbpian Jessie 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.

So you'll need to pick up at least an 8GB card to start with, and if you need more storage as you go on, 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 an 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, and look for the highest class number possible, which represents the speed that the card can read and write data. The site to check is **http://elinux.org/RPi_SD_cards**. This lists all working and faulty cards – 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 sidesteps 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 store-bought cards, so it's worth looking for a good deal or seeing if you've got any spare from an old camera.

Power

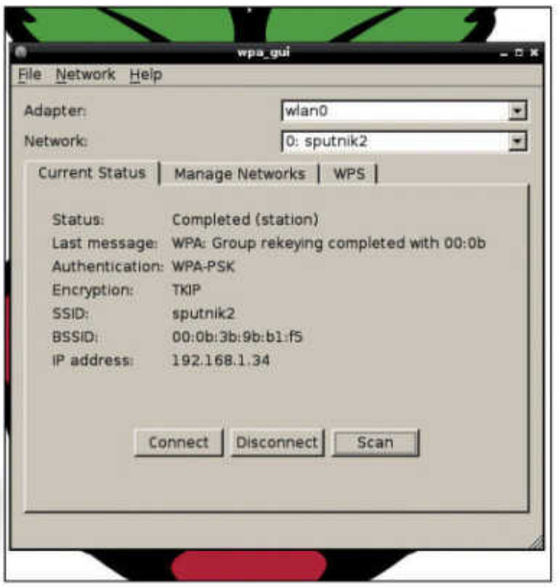
You might not think power should be that much of a consideration for your Pi, but it is. The problem is that if you don't get the power provision correct for your configuration, you can't easily tell whether any problems you experience are a result of the power supply or user error with the software. 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 giving them enough juice. You also need to make sure the power supply 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 outputting 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 most Raspberry Pi models with a standard B plug to standard A USB cable; with a Pi Zero you'll need a USB to microUSB converter. 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. All Raspberry Pi models (bar the Zero and some Model A versions, which have no built-in Ethernet port) connect in exactly the same way, and as long as your home

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.

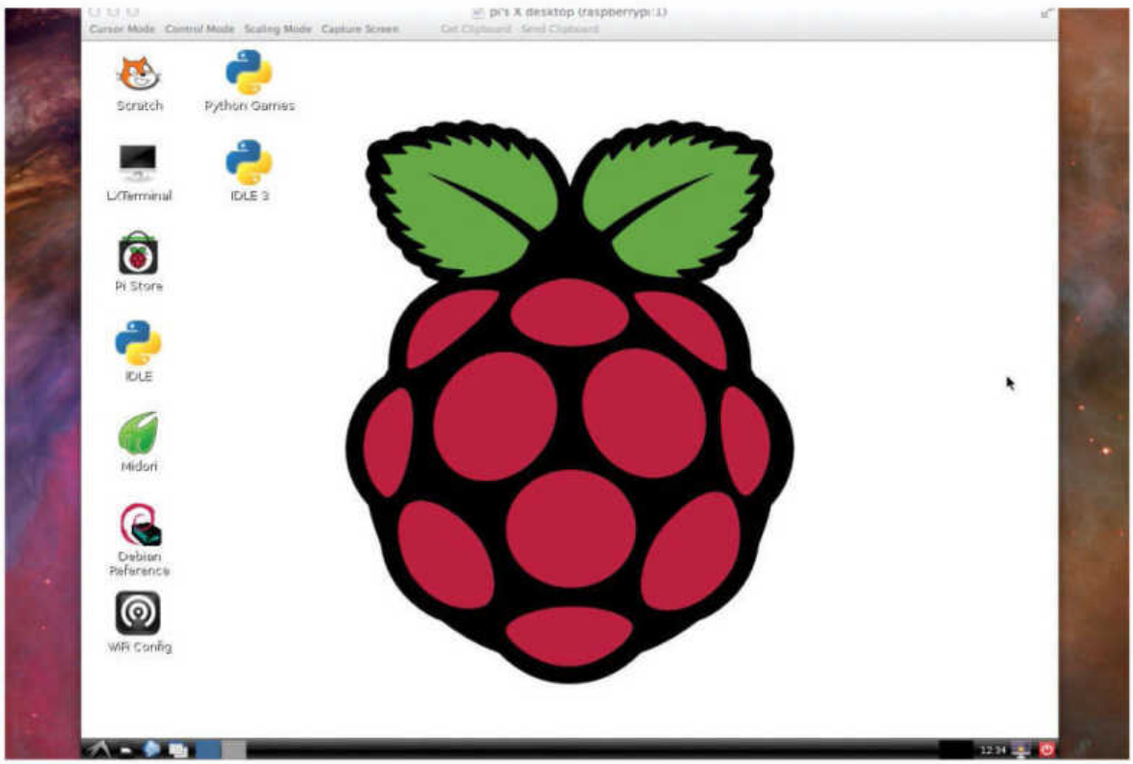
hub is working, simply connecting an Ethernet cable between the two is all you need to do. If you can't access the hub, another option is to use a powerline Ethernet adaptor. One connects to a spare Ethernet port on your hub and into a spare power socket, while the other connects to a power socket closer to your Raspberry Pi, with an Ethernet cable connecting the Pi to your network.

If you've got a model A or Zero 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 »



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

configuration page listing the IP addresses of any connected devices). The password is 'raspberry' for the default 'pi' user 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 Raspberry Pi 2 that output video. The 3.5mm jack socket can output composite video, so 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 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.

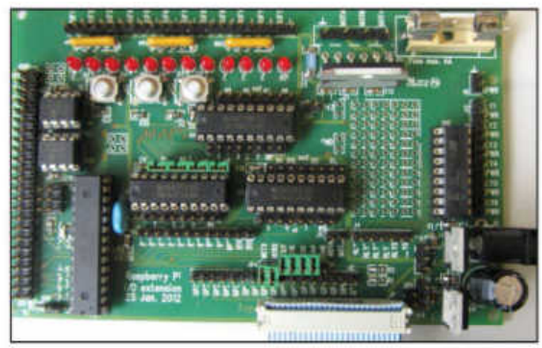
Another strength of the HDMI connector is that it can also carry the digital sound data from the Raspberry Pi. Your TV or amplifier will need to be compatible with this feature to work, and you will need to play around with the Raspberry Pi configuration, but it's a neat solution if you want to turn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

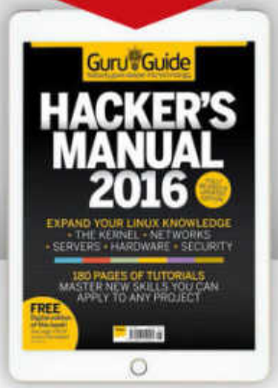
There are 40 pins on the Raspberry Pi 2 board, 26 of which can are used for GPIO purposes, with the others providing power, ground, and EEPROM access. Their function differs between Pi models, so you need to know which revision you have if you plan to buy a daughterboard that plugs directly on to the GPIO pins. One of 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. It's a great way to interface more directly with those pins – but not, by any means, the only way. You don't necessarily need an additional board to get started, because the pins can be used for many projects without modification. Or you can browse the market; many innovations are springing up, sometimes manufactured in very small numbers, so if an expansion board – or 'hat' – crops up that takes your fancy, grab it.

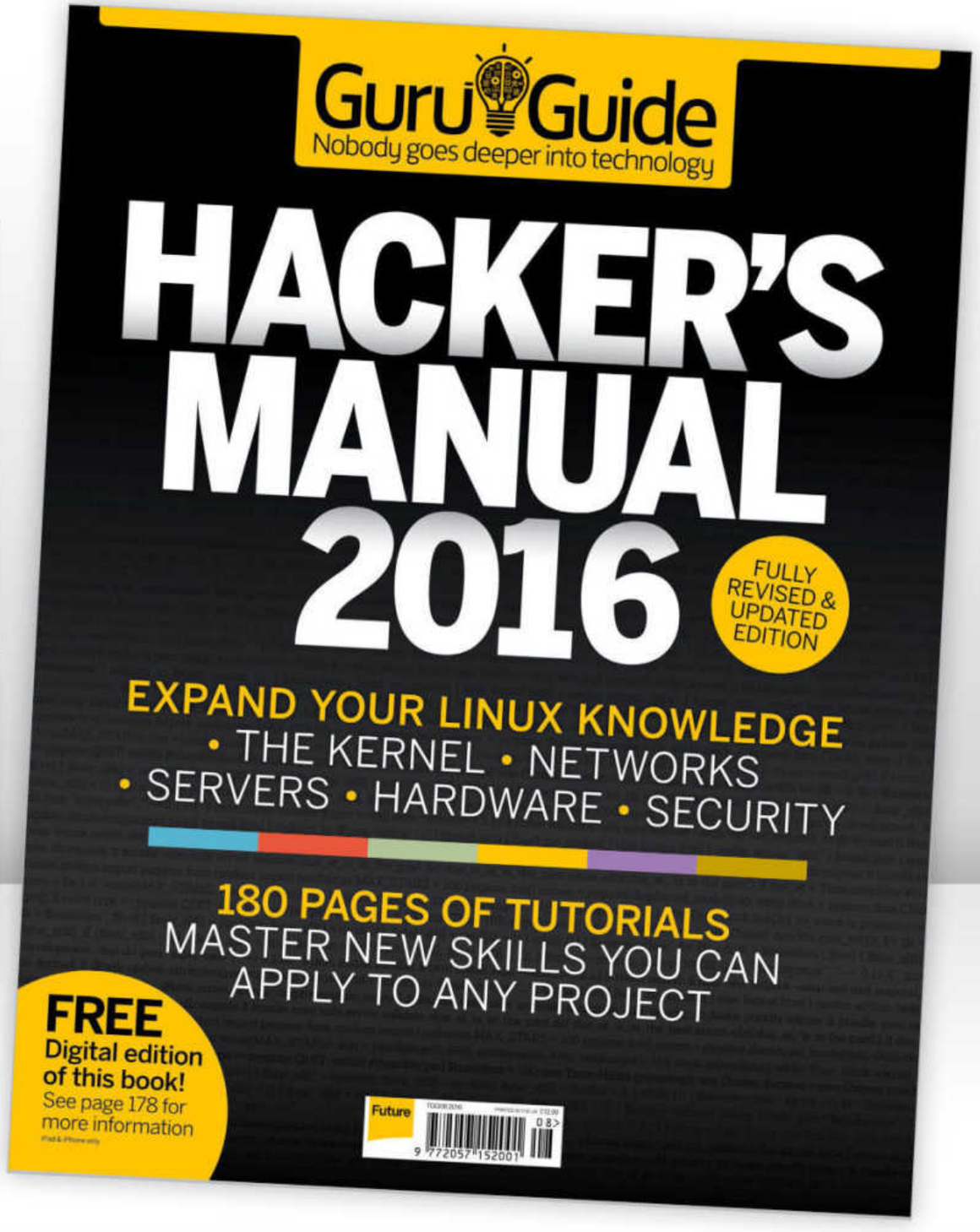


Boards that connect to the GPIO pins on your Raspberry Pi, such as the Gertboard, can be used.

POWER UP YOUR LINUX SKILLS!

OUT NOW!
WITH FREE DIGITAL EDITION



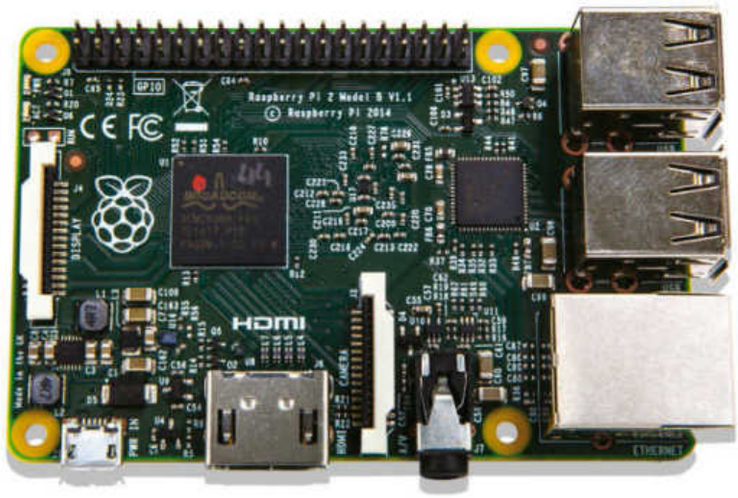


DELIVERED DIRECT TO YOUR DOOR

Order online at **www.myfavouritemagazines.co.uk**
or find us in your nearest supermarket, newsagent or bookstore!

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 a Raspberry Pi and a desktop 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 boot proceds, 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 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 and the latest Pi 2 have two, and the Zero has just one. The number, however, isn't really important as only one explains boot issues. First lets see which versions have what LEDs:

All models of Raspberry Pi

- **LED1:** Green, labelled ACT: SD card access

Raspberry Pi 2 and Model A+

- **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 (if you have one)should light. This indicates the device is getting the correct amount of power, and this LED should remain lit while 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.

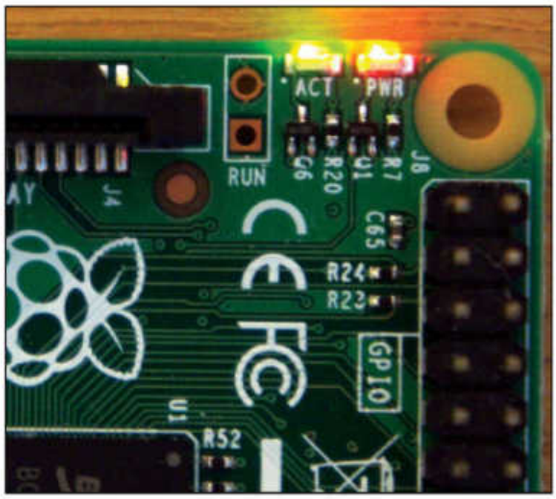
It's also possible to decode which part of the boot process the Pi is stalling at. Here's a list of what the various flashing modes from the ACT/OK LED mean – although these rely on a firmware from at least the middle of 2012, and we've taken this data from the Raspberry Pi forums. In use, we've found 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.

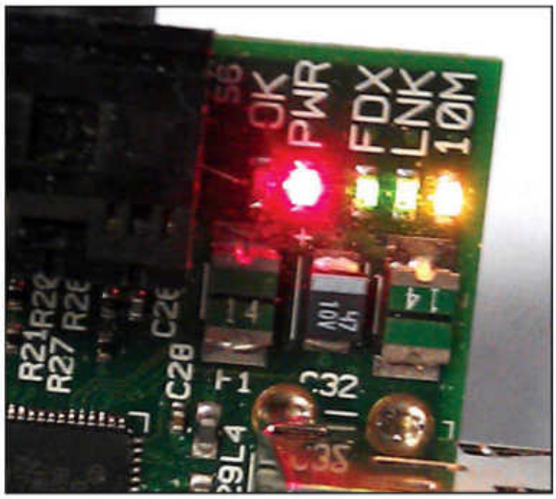
- **3 flashes:** start.elf not found
- **4 flashes:** start.elf not launched
- **7 flashes:** kernel.img not found
- **8 flashes:** SDRAM not recognized. You need newer bootcode.bin/start.elf firmware, or your SDRAM is damaged

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

The boot





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

It is possible to troubleshoot these problems by looking for the files involved and making sure they're correct. Take a 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. The installation guide in this issue 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-pin 3.5mm A/V jack, depending which is available. 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. 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.

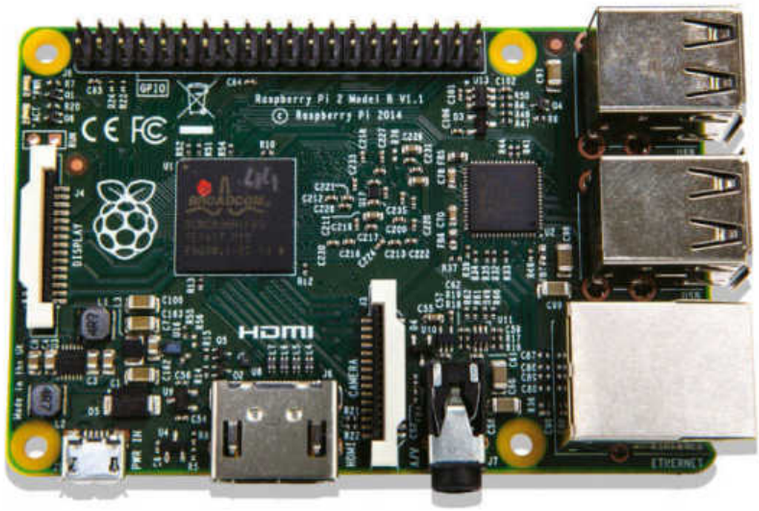
The green LNK LED is the networking equivalent to the ACT LED for SD card access, because it's this that flickers on and off as data is received and transmitted across the network. If the LED does light up, you can assume, at least at a low level, your hardware has been able to negotiate a connection 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. Note that the Pi Zero has no networking built in.

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.

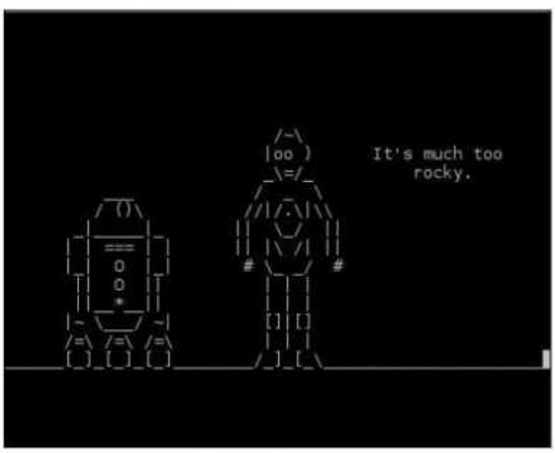
How will I know?

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.



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).

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

```
find / -mtime 1
```

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

Power up

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

First, we'll create a new directory, move into it and create a few empty files (we use the command **touch**, which creates an empty file with the name of each argument). Hint – you can use tab completion (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:

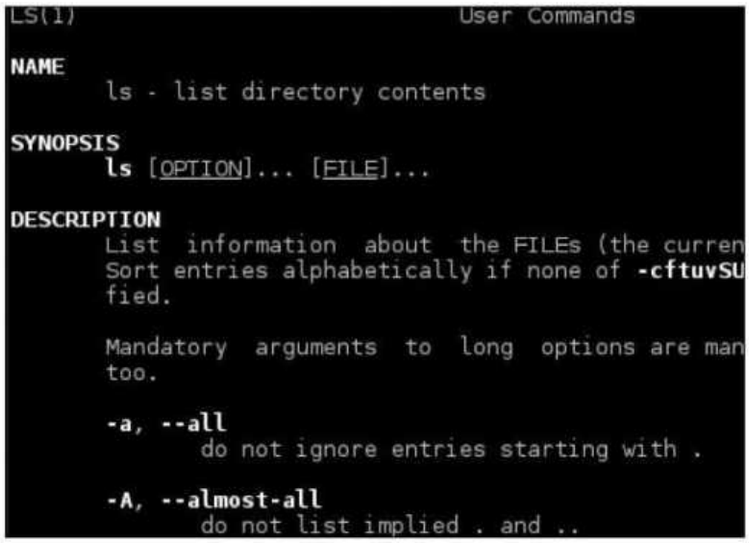
```
ls > files
cat files
```

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.

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

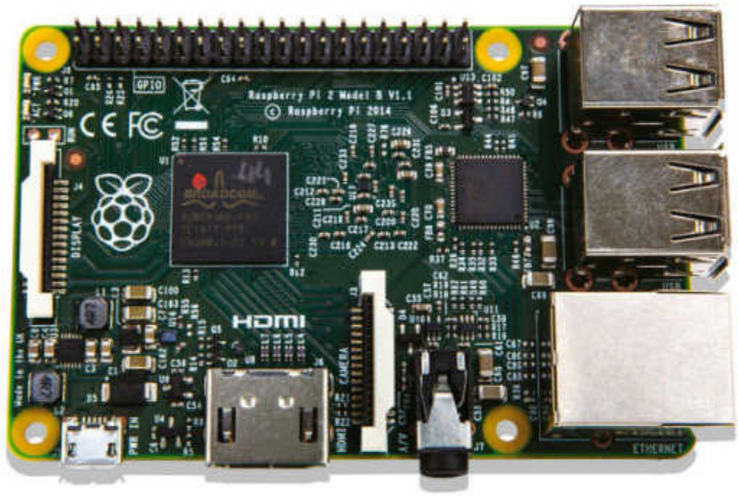
» **[Ctrl]+[C]** Kills whatever is running in the terminal.
» **[Ctrl]+[D]** Sends the end-of-file character to whatever program is running in the terminal.
» **[Ctrl]+[Shift]+[C]** Copies selected text to the clipboard.
» **[Ctrl]+[Shift]+[V]** Pastes text from the clipboard.

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.

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 first model of the Raspberry Pi

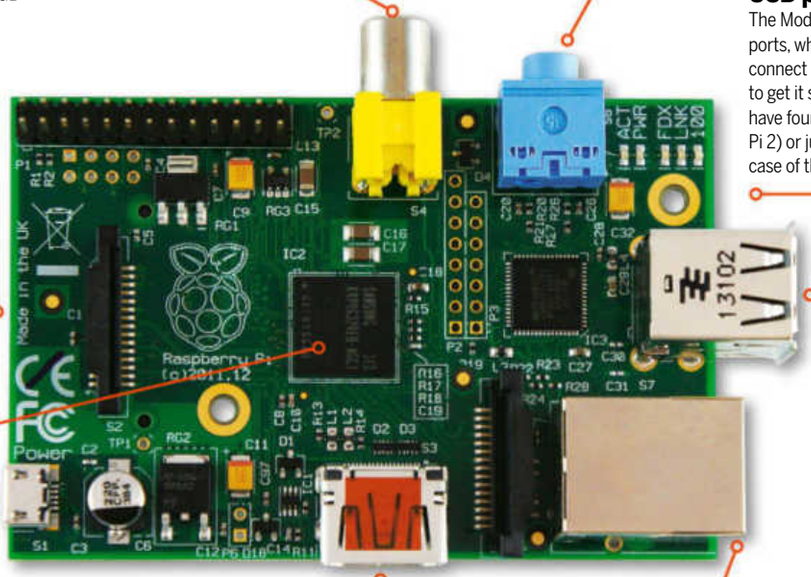
SD card slot
The Raspberry Pi doesn't come with any storage, so you'll have to procure your own SD card to store the operating system and your files. The beauty of this is that you only pay for what you're going to use – you can pay £8 for a 4GB card, £200 for a 128GB card or anything in between.

RCA video out
For older TVs, there's an RCA video connector. This has now been removed, replaced by either the same interface delivered through the 3.5mm jack or via on-board pins on the Pi Zero.

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. Later revisions have four ports (B+, Raspberry Pi 2) or just one port in the case of the Pi Zero and A+.

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 256MB, while the Model B comes with 512MB.



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.

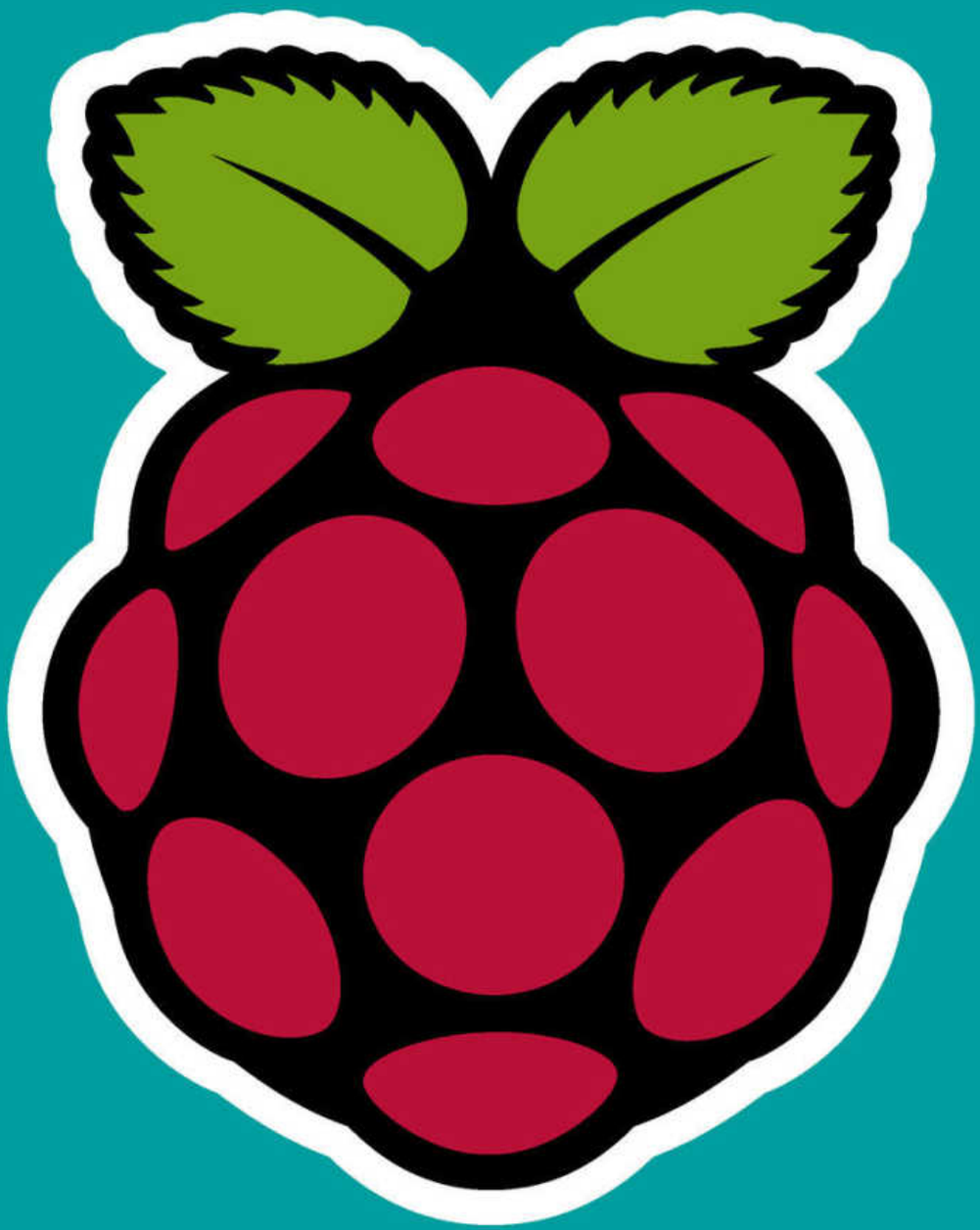
HDMI out
The design team assumed that most people would be plugging into either a television or a screen, so there's an HDMI out for connecting to modern television sets and computer screens.

Ethernet
Plug straight into the internet – unless you have a Model A Raspberry Pi or Pi Zero, which don'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, Pi 2, Zero. 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+ and Zero have 512MB of RAM, the Model A+ only has 256MB and the Pi 2 has 1GB. The Model A+ and Zero have 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 A+ and Zero lack. The other difference is the price, with the Pi 2 costing $35 rather than as little as £5.

If we were to to choose, the new Pi 2 is the one to go for as it offers so much more power. The Model A+ and Zero are 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.

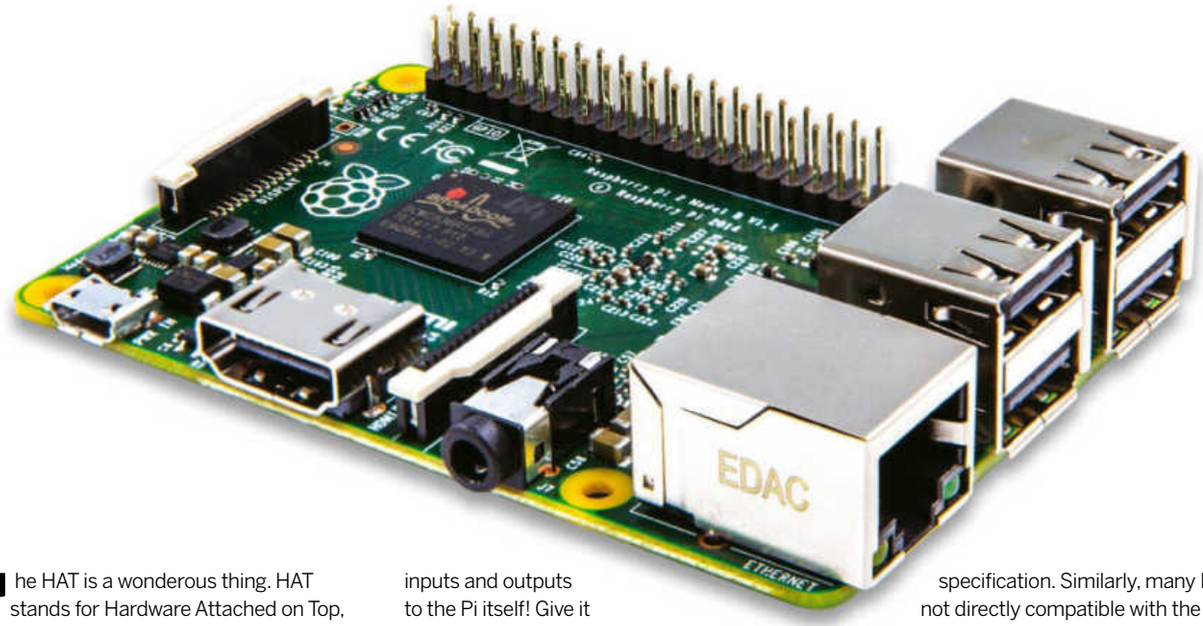


Hardware

The Raspberry Pi itself is just the beginning.

HATs on

That Raspberry Pi down there looks a bit naked, doesn't it?



The HAT is a wonderous thing. HAT stands for Hardware Attached on Top, and it is exactly that: additional hardware that you attach to the top of your Raspberry Pi, making use of its handy GPIO pins to add additional capabilities and sensors.

If you've never put a HAT on top of your Pi, you're missing out on one of its core features. Those GPIO pins are the perfect way to get started with hardware programming, and drag the Pi away from the loving embrace of a monitor and keyboard and into the dark unknown of an independent existence. Add inputs and outputs to the Pi itself! Give it an eye on the world! Teach it to love! Alright, maybe not that one.

Wherever I lay my HAT

The HAT system is an invention of the Raspberry Pi foundation, which released a full specification around the time of the launch of the Raspberry Pi B+, which expanded the number of GPIO pins available to the Pi from 26 to 40. For this reason, the majority of HATs are not compatible with the original hardware specification. Similarly, many HATs are not directly compatible with the Pi Zero, as it does not include header pins and does not conform to the same footprint as the B+ / Pi 2. That said, it's a trivial matter to solder on your own header pins, and there's nothing directly about the hardware which would preclude you from using any of these with a little work.

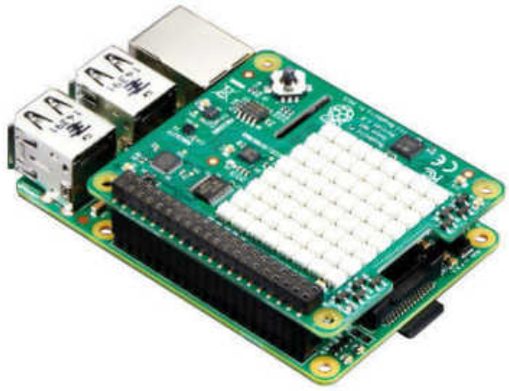
So, it's time to look at our pick of the best HATs out there. Take your Pi out of its case, pick one of these up, plug it in, and fire up Python – it's time to make something awesome.

Pi Sense Hat

We sense a disturbance in your wallet...

If it's good enough to be blasted into space with astronaut Tim Peake, it's good enough for your project, and the Pi Sense Hat – the centre of the Astro Pi project – is good for a lot. The 8x8 RGB LED matrix is probably a decent first indication of its abilities, as it can be used to display data gleaned from each of the on-board sensors by way of colours, shapes, graphs and more. You can monitor everything from temperature and humidity to barometric air pressure as your Pi makes its way into the atmosphere (or, more than likely, stays a little closer to terra firma), but it's the intertial measurement unit on board that's likely to be the real deal sweetner: screw this onto the top of your Raspberry Pi and you'll have access to a combined accelerometer, magnetometer and gyroscope for use in your projects. This is likely to be a robotics and aviation staple for some time to come, but its sheer range of abilities and solid codebase should make the Sense Hat a part of many more varied and interesting projects going forwards. You owe it to yourself to have this in your collection. **For more, see p116.**

£25, pimoroni.com



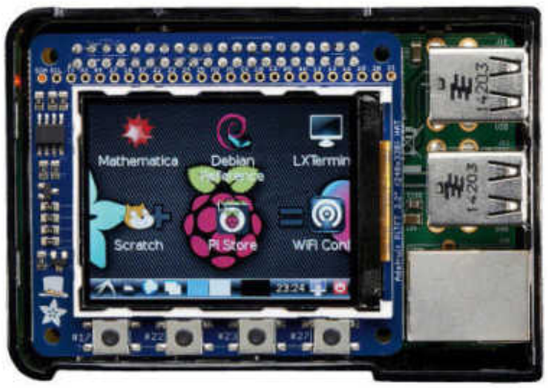
The Sense HAT's 8x8 matrix begs for a little visual experimentation.

Adafruit PiTFT 2.2” HAT Kit

See what’s what with this HAT.

Running without any additional soldering required on the A+, B+ and Raspberry Pi 2 (and with minimal effort required to get it going on the model A and B) the PiTFT is the cutest little 320x240 pixel display. Perfectly sized to fit on top of the footprint of the B+ and Raspberry Pi 2, you could use this to output console information (how about a quick-glance view of top while you’re running?) or any kind of graphics; we’ve seen it running the likes of Quake, for example. Making use of the hardware SPIO pins (SCK, MOSI, MISO, CE0, CE1) and just one GPIO pin, this leaves every other GPIO pin free. They’re all dealt with by a breakout cable; if you’re itching to put together a handheld device, for example, this could be the perfect option as you’ll have ample interfacing space for buttons. Speaking of which, Adafruit has been kind enough to put four microswitches below the tiny TFT panel, so you can make up for the lack of a touchscreen by fashioning your own rudimentary input system with, again, no soldering required.

$24.95, adafruit.com



Raspbian is totally useable on a 2.2” display. Honest. You’ll just have to trust us.

SparqEE GSM Cellv1.0

“Hello? HELLO? I’m on my Raspberry Pi!”

It’s time to bin that fancy iPhone and throw that Android handest in the river, because your Pi just got cellular communications capabilities! Alright. Maybe not. But imagine the applications: a phone home – figurative or literal – function on a roaming robot; your own portable 3G router; or fallback for your home network should your wired connection go down, to name just three possibilities. You could, we’re sure, cobble together an actual communications device with this and a screen and some kind of input, as it takes a regular mini SIM – contract, pay as you go, or a SparkEE SIM – but it’s probably more suited to Internet Of Things type designs. Connecting to WCDMA/HSDPA (2100/1900/900 MHz) or GSM/GPRS/EDGE (850/900/1800 /1900 MHz) means it’s pretty network-agnostic, although it lacks the blistering speeds of 4G and you’ll need a Sparq Shield (£9.80 - £11.46) to actually hook it up to your Pi’s GPIO pins, as this unit is designed to work on multiple platforms – it’s also Arduino-compatible. Useful!

£69.90, uk.rs-online.com

Part cellphone interface, part terrifying robot centipede.

TrafficHAT

Be sure to stop on red...

Let's be straight, here: you're not going to make something incredible with this. Really, you're not going to make anything at all. It's a set of three LEDs in traffic light order, a hardware button, and a buzzer. That's it. But for the meagre price, it's just about the most perfect way of getting started with GPIO programming, whatever language you're using, as it doesn't require any special libraries or techniques. If your codebase supports GPIO, it supports this. What's more, the developer has created a special addon for beginners' language Scratch, so literally anyone capable of poking around on a Pi can get something out of this. As a Raspberry Pi owner, you're duty bound to either foster your own skills in hardware control, or pass down a love of coding to the next generation, which means this apparently useless little HAT is actually completely essential – and since it comes either in kit form or fully assembled, you could even teach yourself a bit of soldering in the same breath. Also available: SnowPi, the Festive GPIO Snowman. Yes, really..

£7, ryanteck.uk

Stop on GPIO 24, go on GPIO 22, and make sure you honk GPIO 5 relentlessly.

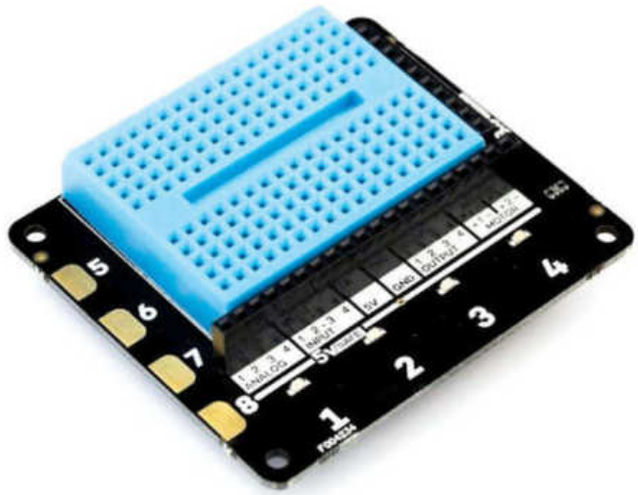
Explorer HAT Pro

The pith helmet of Pi HATs

There are many Pi HATs which do specific things. You'll find a large number of them on these pages. And they're perfect if you have a single goal that you know you want to achieve. But one of the key drivers of home electronics is experimentation, which (outside of the Pi and Arduino world) would usually be done on a breadboard, a copper-linked board with through-holes into which you can shove all the resistors, LEDs, capacitors and other components you could possibly want to use. This isn't precisely that, but it's close. The Explorer HAT offers a mini breadboard, but also breaks out a whole bunch of useful things: 5v tolerant inputs, 5v outputs, capacitive inputs along the edge of the board, coloured LEDs for output, analogue inputs for all those tricky sensors and pots, and a pair of H-bridge motor control drivers. All this kit is backed up by a comprehensive Python library with full documentation and examples, so it's relatively simple to plug in and get started very quickly indeed. Perfect for all experimenters.

For more, see p112.

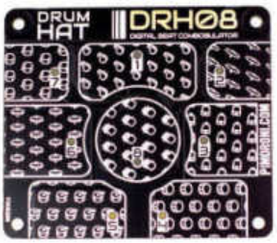
£18, ryanteck.uk



If your breadboard is blue and full of holes, don't cut bread on it.

PianoHAT & DrumHAT

Whether you plinky-plonk or rat-a-tat.



Fat-fingered pianists need not apply...

The applications for these two are obvious, right? Right. The PianoHAT offers up an octave of tiny keys so you can turn your Pi into a mini keyboard, and the DrumHAT offers up eight dinky drum pads for the maximum in miniature rhythmic fun. But think outside the box for a second. Don't just think of the DrumHAT as a drum machine interface. Consider the fact that it adds eight individual touch sensitive buttons, each with corresponding and individually addressable LEDs to your Raspberry Pi. And the PianoHAT, with its super-useful octave and instrument buttons, includes a full 16 inputs and LEDs applicable to any project. But then it's important not to forget how awesome these are for controlling the likes of Sunvox or Yoshimi, turning your Pi into a true portable mini-synth or, in conjunction with a USB-MIDI adapter, a controller for any hardware synths you might have lying around. Why not code the LEDs to follow along with a MIDI file and teach you how to play your favourite tunes? With the full Python library included, there's no excuse for you not making the top 40.

£15 (PianoHAT) / £12 (DrumHAT), pimoroni.com

Display-O-Tron HAT

Perhaps someone should make a light cycle game...

Here's a challenge for you: make this display the Tron script.

Bright and shiny VGA displays are all well and good, and they certainly have a lot of practical applications, but there's a lot to be said for LCD displays like this. They can often be more of a programming challenge, for a start; you'll need to learn how to interface with each of the characters on this 16x3 ASCII screen, or how to code one of its six possible custom characters. It also includes an LED bar graph and a six-zone RGB backlight, all of which is individually addressable. Its practical applications should be pretty clear. We'd probably install this as a network monitor of some kind, with the coloured LEDs set to give an at-a-glance view of current traffic, the bar graph LEDs flickering to represent packets or individual nodes, and the text on the screen giving more detailed information. It could also be useful as a more specific system monitor, with the output of top piped to the screen; why not split the LEDs on the display to show two different colours, representing remaining memory and CPU cycles? So many possibilities, so little time!

For more, see p55.

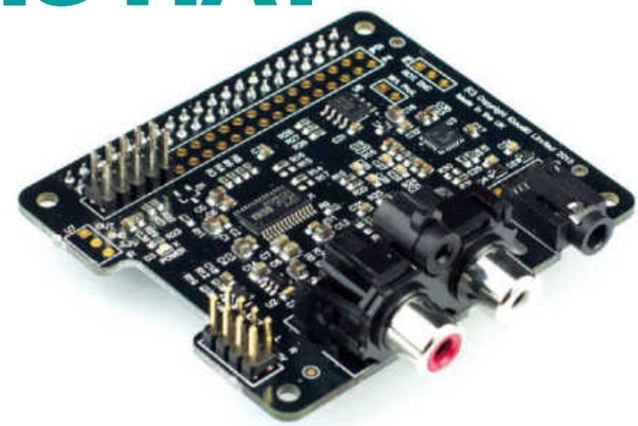
£22, pimoroni.com

IQAudio RaspberryPi-DAC+ Audiophile HAT

You can hear clearly now.

DACs – or digital to analogue converters – are not cheap. And the Raspberry Pi isn't exactly short of ways to output audio. But this HAT is both reasonably priced and a brilliant way of getting high-quality audio output from your Pi without sacrificing too many CPU cycles. It plugs straight in to the Pi's I2S audio signals, bypassing the need for a processor-costly USB solution, so if you're looking to tease true HD audio from your mini-computer, this could in fact be your only plausible solution. Plug your incoming audio into the 3.5mm jack of your Pi, process and EQ it to your liking, then send it back out of the left and right phono plugs or through the integrated headphone amp. Your sound will emerge smoothed and with added clarity thanks to the on-board DAC, which has its own EEPROM for configuration. There are handy cutouts for the Pi screen and camera modules, too, as well as access to most of the Pi's I/O pins and an onward connection, should you need it, to the 20w Pi-AMP+ (£45, iqaudio.co.uk).

£35, thepihut.com



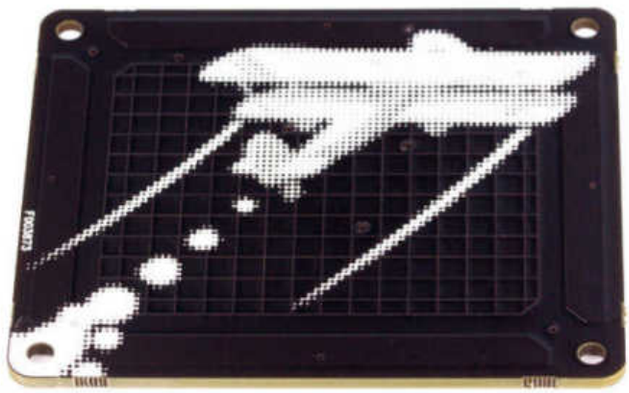
This little thing will make your tunes sound brilliant. Building a headless Spotify box? Get one.

Skywriter HAT

Write like you mean business.

Tom Cruise has a lot to answer for. Not just that incident where he jumped up and down on Oprah's sofa. Not just Eyes Wide Shut. Not just those suspicious teeth. We're talking Minority Report, and his penchant for waving his hands about like a cyber-doofus in order to control a computer of the future. It's not super-likely to become the control scheme of the next decade, is it? Well perhaps you'd like to develop some software to change that? With the Skywriter HAT, you'll get 5cm of 3D positional electrical sensing. Wave a digit or elbow over this board – you don't have to touch it – and it'll feed back data to your Pi. You can use it to set up gesture control, a virtual pointer, tapping and more. And because it senses an electrical field rather than requiring direct contact, you can hide that slightly questionable silkscreen plane picture behind, for example, some non-conductive acrylic in your final build. OK, it's not quite Tom Cruise's avant-garde arm-flailing, but it's a start, right?.

£16, pimoroni.com



Alternatively, try the Hover (p47).

Picade

A classic look for your classic games.

The Raspberry Pi is the heart of many arcade systems. Indeed, we've heard tales of many owners of proper cabinets who've ripped out the JAMMA guts and wired up a Pi and a fully-loaded SD card (filled, naturally, with entirely legally-acquired arcade games) in order to create a multi-arcade system. With Retropie installed, you can even create a multi-system machine with a glorious user interface. But what if you don't want a gigantic arcade cabinet looming over your living room, yet desire some delicious tactile input?

That's where the Picade comes in. It's a heck of a lot more simple than whipping out the jigsaw and woodworking up your own mini cabinet, and it comes with literally everything you'll need to get a micro-cabinet put together, bar a couple of screwdrivers and the Raspberry Pi itself. You get twelve top-quality microswitches, a proper arcade joystick, artwork, powder-coated cabinet components, wires, speakers, and the Picade circuit board and an 8" LCD display with the appropriate driver board.

› **Buttons! Boards! Cables! The lot!**

Yes, it's probably slightly more expensive than cobbling the kit together yourself. And yes, you probably don't actually need it. But since when has needing something been a barrier to having it? Relive those classic games in style, and in a package you can tuck away when you're finished with it. Or, get out the Dremel to tweak the layout, or use the components to cobble together a system of your own devising. That's what it's all about, isn't it?

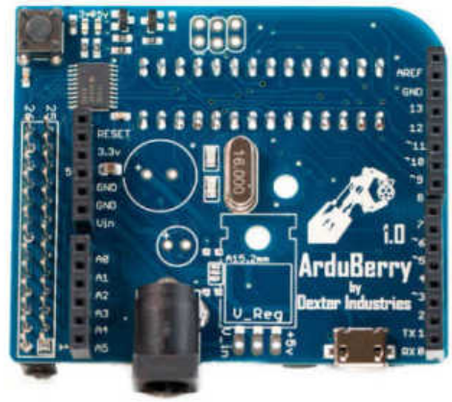
£180, pimoroni.com

Arduberry

Is it an Arduino? Is it a Pi?

The Arduino is arguably the platform that, more than any others, spawned the Raspberry Pi. A mini marvel perfect for embedded systems, it lacks the raw power of the Pi but makes up for it in sheer numbers of accessories and low-wattage applications. So why, if you've graduated to the Raspberry Pi, should you have to abandon your collection of Arduino shields? You shouldn't. So don't: the Arduberry is the solution. Install this HAT and you can plug in and stack your Arduino shields (or even consider new ones, as the platform is by no means dead) with little to no configuration required. Given the Arduino's wide spread and somewhat heady vintage – it's been around since 2005 – there's a vast array of sensor packages, communications devices and much more available. Combine these with the vastly increased power of the Raspberry Pi, add in a little interaction with full-featured applications and a little coding flexibility, and you've got yourself a device far in advance of its original potential. It's a pricey way of going about it, though: you can pick up a complete Ardunio kit with all the gear you'll need for less than half the price of this...

$30, dexterindustries.com



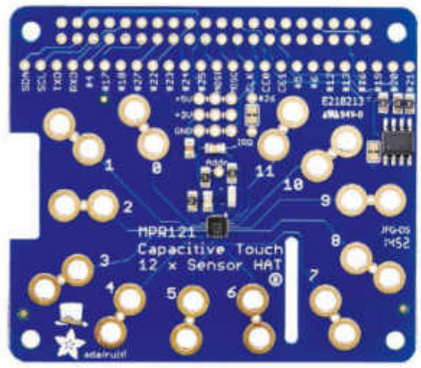
Arduino sheilds might be the old HAT, but they're certainly not old hat.

Adafruit Capacitive Touch HAT

You've got the touch...

Grab your alligator clips, because it's time for some fun. Adafruit's own demonstration on its website shows this HAT hooked up to an array of fruits and vegetables, responding to each of them being touched. Indeed, we show you how to make a banana instrument with it over on p66. And that's certainly one thing you could do with it, although we'd argue it's not necessarily the most practical use of the Capacitive Touch HAT. It adds twelve individual contact points that detect, yes, capacitive touch, so you can wire them up to anything conductive – a thin strip of metal, for example – or water filled, and detect when the touch of a human body drops the capacitance of the object. It's a version of the same tech used on modern touch-screens, albeit briken up into individual points rather than representing a grid. In its most base form, you'll be able to read on-and-off touches, but dive into the raw data from its sensors and we suspect you'll be able to tweak sensitivity and read exact capacitance results as well. As for potential projects, why not have it keep tabs on every time your fridge is touched, or make yourself a spy device to find out who's touching your potatoes? We won't judge...

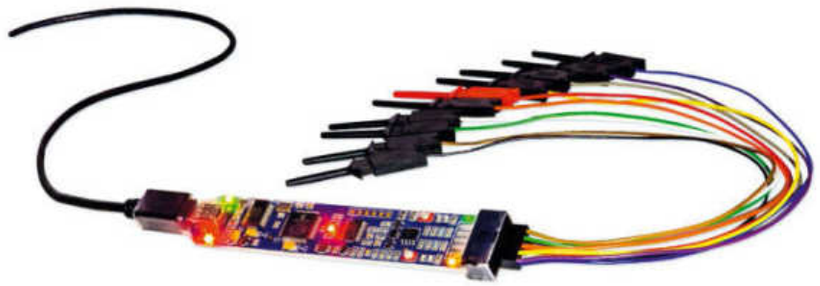
$15, adafruit.com



Twelve sensors, twelve things you can sense. Beats the usual five senses.

Bitscope Micro

Scope out the situation.



There's no reason the Raspberry Pi has to be the centre of your electronics project. It could just as easily be the outside influence that helps your project come to life, and with the Bitscope Micro you can transform your Pi into a suite of tools that every electronics buff requires. It's a 20MHz digital oscilloscope, allowing you to keep track of those pesky signals! Its an 8-channel logic analyser that can also deal with serial logic and protocols! It's a real-time spectrum analyzer, meaning you can split up those signals and see exactly what's going on! It's a waveform generator, a clock generator, and a multi-channel data recorder! While it's not technically a HAT – it's USB powered – we've included it here because, frankly, you need one of these in your toolbox just as much as you need a soldering iron and copious amounts of hot glue. And if you're pulling off electronics projects on the Pi, you can always hook this up to a spare PC or Mac and probe around your project from there. Grab the BNC Port Adapter if you want to add standard oscilloscope probes. **For more, see p49.**

£60, uk.farnell.com

Looks like a terrifying cyber-octopus, but really it's a useful electronics device.

4Tronix PlayHAT

Play is a serious business.

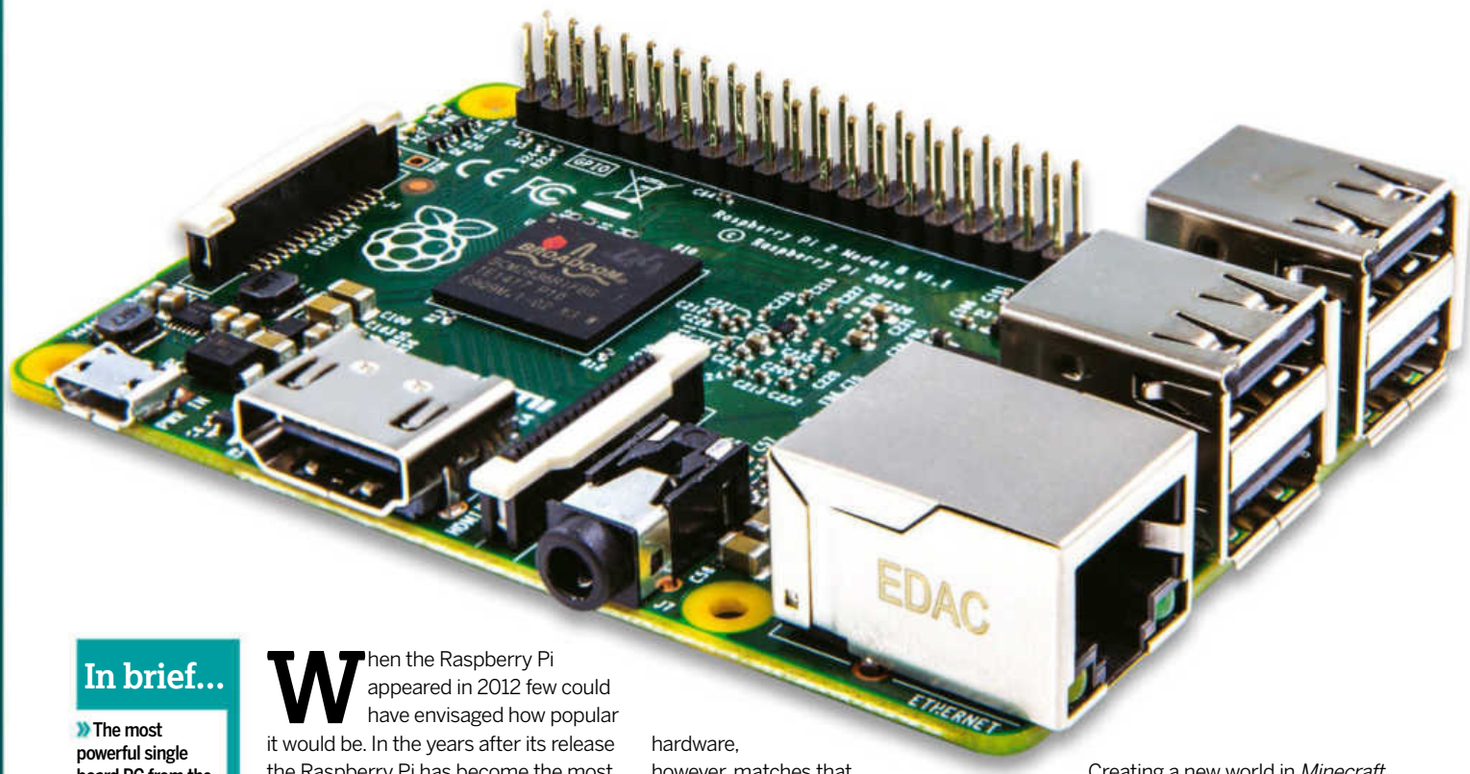
We'll close this out with a bit of a challenge for you. It's all very well having new fangled displays, sensors, inputs and more. But there's a lot to be said for old-school tech and what you do with it. So pick up one of these reasonably priced, lightly equipped boards, and do something special with it. What crazy way could you use the 9x9 neopixel display? How can you embrace the four coloured buttons and buzzer? And how can you use them together? You might think of a dice roller, a Simon-type game, a code-breaking excercise. But we think there's more to it, and we'd like to challenge you to develop something more advanced. Knock together a full old-school game, perhaps, or a demo making full use of the LED matrix. Combine this with other hats. Do something awesome. Then share it with us and the clever folks at Linux Format magazine – who knows, you could make it in to print. Email the team at lxf.letters@futurenet.com and we'll let you know what we think!

£9.50, uk.farnell.com

What does it do? Why, it does whatver you want it to do...

Raspberry Pi 2

We salivate at the prospect of an extra-delicious Pi and promptly break our teeth on the definitely non-edible treat.



In brief...

- The most powerful single board PC from the Raspberry Pi Foundation comes with the spec boost that we were all hoping for. The Pi 2 is the strongest in a line of products from the Foundation and can run a number of 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.

The Raspberry Pi 2 sees the original ARM unit (which also appears in the Pi Zero) replaced with an ARM 7 CPU running at an improved 800MHz, easily overclocked to a full 1GHz. But rather than stick with a single core, this version 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 big-boy Pi actually is when directly compared to the B+.

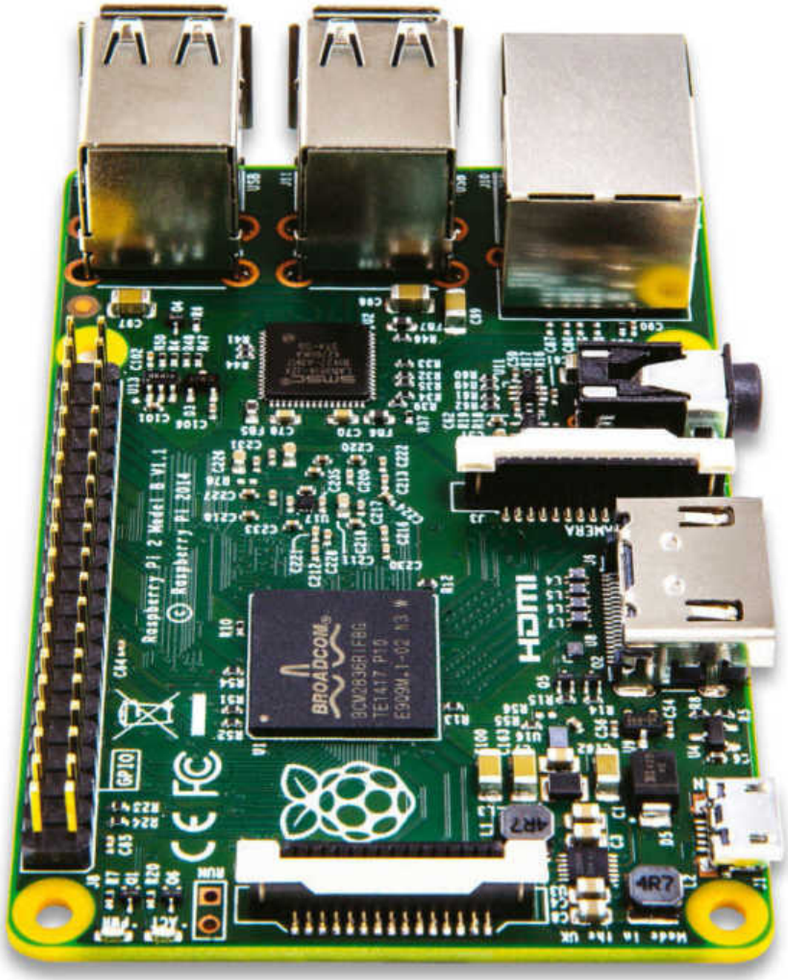
> **"Booting from cold: The B+ managed it in 33 vs 17 secs for the Pi 2."**

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 the online SunSpider benchmark 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



The form factor may be the same as the B+, but the Pi 2 packs a punch.

an average of 3.8MB/s and on the Pi 2 we saw 4.6MB/s, which is an 0.8MB speed increase.

The Raspberry Pi Foundation has 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.

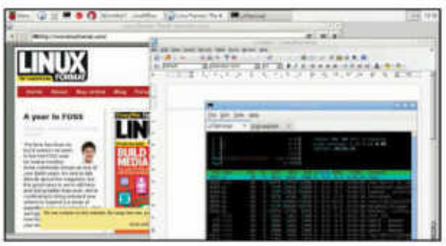
SunSpider Benchmarks

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 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

The Raspbian desktop runs well on the B+, but running on the boosted 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

THE EASY WAY TO LEARN WINDOWS

AVAILABLE IN STORE AND ONLINE

www.myfavouritemagazines.co.uk

Raspberry Pi B+

How does the cheaper, less-powerful B+ compare with its even punier 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.

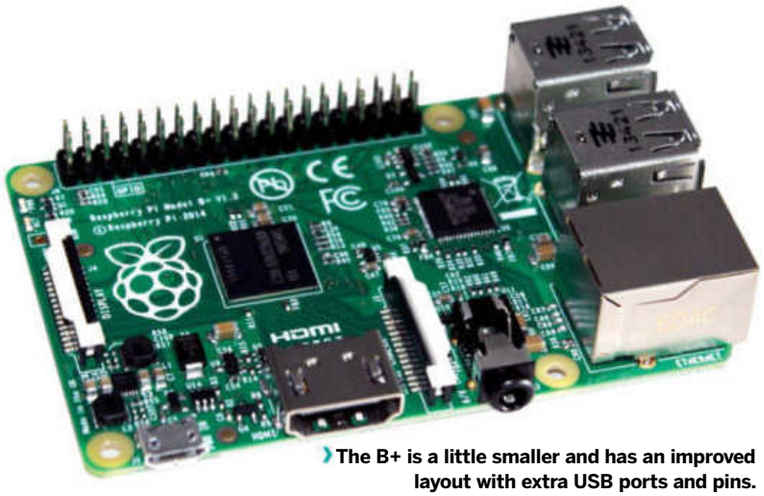
Specs

- 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

Shortly after releasing the Raspberry Pi Compute Module the Raspberry Pi Foundation snuck another single board computer into its catalogue. The Raspberry Pi Model B+ represents the final version in the Model B series, although it's obviously been superceded by the model 2 at this point. But it holds up!

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 few expansion boards, if any, use the full 40 pins of the new GPIO.

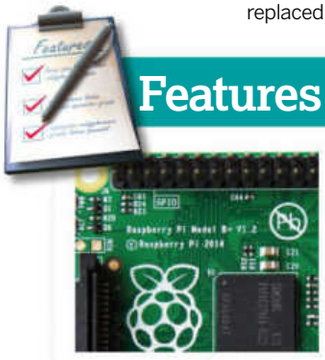
The extra pins breakout more of the SoC, giving you more pins for bigger projects and there are two new GPIO pins – pins 27 and 28 – which enable future add-on boards to use an EEPROM chip, and will automatically configure the add-on board on boot.

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

Raspberry Pi Zero

Time to delve into another helping of Raspberry Pi. This time it has zero calories but does it still taste as sweet?

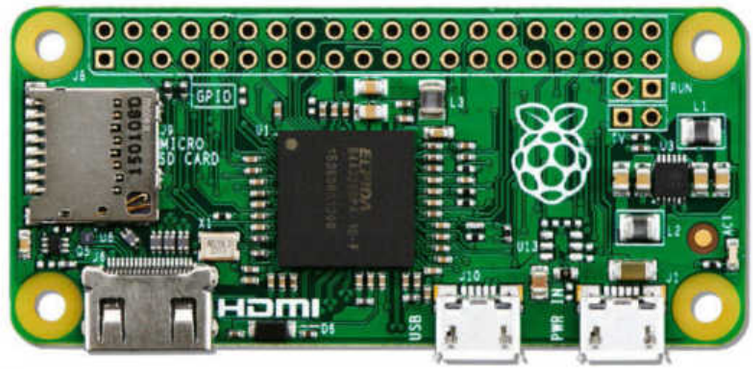
Specs

- **CPU:** Broadcom 1GHz BCM2835
- **RAM:** 512MB
- **Storage:** Micro SD card slot
- **Ports:** Mini HDMI (1080p 60), Micro USB for data and power
- **Other features:** Unpopulated 40 pin GPIO header, unpopulated composite video header
- **Size:** 65mm x 30mm x 5mm

This is the second Raspberry Pi released in 2015 and this time the Foundation shift their focus from power to price. Previous Raspberry Pi models have sold for around the $30 mark and improved the specifications with each release. But after a meeting with Google's Eric Schmidt, Eben Upton changed the focus of the next Raspberry Pi to being cheap rather than all powerful.

The Raspberry Pi Zero is a $5 computer. That's not a typo, we can now buy a computer for the same price as lunch. The Pi Zero is closer in specification to the original Raspberry Pi using the original BCM2835 System on a Chip (SoC) with an ARM11 CPU clocked at 1GHz, and it offers 40% more power than the Pi 1. Micro SD card storage is present on the Pi Zero, but the usual push-click locking mechanism has been removed. Ports around the board are sparse with only micro USB for power and peripherals and a mini HDMI port for audio/video.

The micro USB peripheral port requires the use of an adaptor as does the mini HDMI port, but as always, a number of retailers have already filled that gap. On Pi Zero you'll find no DSI or CSI connectors, which means no compatibility with the official Pi touchscreen or camera. These connectors were taken off to reduce the cost of the Zero. Pi Zero features the, now standard, 40-pin GPIO (General Purpose Input Output) but the header pins aren't present, offering an opportunity to try out your soldering skills. We tested the GPIO with Python 3 and Scratch and can report that it worked exactly as expected. We also tested a typical add-on board, in this case the Unicorn HAT from Pimoroni, and that too worked after installation. So the Pi Zero is compatible with a large number of the add-on boards.



The Raspberry Pi Zero is a small board but retains the 40-pin GPIO header for compatibility with the massive number of Raspberry Pi add-ons.

An IoT thang

We tested the Pi Zero with the latest version of Raspbian Jessie, updated just before the Pi Zero was released, and boot times were slower clocking in at 52 seconds from power on to desktop – this is comparable to the original Pi.

So who are the target market for Pi Zero? The makers are one group who will benefit from a low-cost platform with an expansive user base. The Pi Zero is an embeddable platform that will fit well into an IoT (Internet of Things) project or any other permanent installation. While the Pi Zero doesn't come with any Wi-Fi connectivity, it can be added relatively easily. In fact, there is already a hack to add a Wi-Fi dongle to the unused USB headers under the Zero. Another group to benefit from the Pi Zero are those who cannot afford a computer. With Pi Zero we reduce the cost to the bare minimum and offer a low point of entry for families to enjoy learning together.

So why should you buy the Raspberry Pi Zero? If you love robotics, weather projects and hardware hacking then the Pi Zero is an ideal platform for low cost experimentation. Embedding the Zero into a project is now just as cost effective as using boards, such as the ESP8266 and many of the Arduino clone boards. By removing some of the components and leaving a distilled Pi experience, we have a cheap, embeddable platform that can easily integrate into the Raspberry Pi add-on ecosystem. Being compatible with add-on boards and using the same OS also enables access to the vast library of Raspberry Pi centric resources.

The Raspberry Pi Zero now joins the family of boards and offers an exceptionally cost-effective first step into the world of computing, coding and electronics.

Benchmarks

Test	Pi 2	B+	Zero
SunSpider (ms)	2,476	9,477	10,507
3D	499	1,657	1,672
Access	190	482	1,258
Crypto	194	647	837
Math	141	431	872
String	930	4,281	2,968

Sysbench	Pi 2	B+	Zero
Prime avg (ms)	29	50	35
Prime min (ms)	29	50	35
Prime max (ms)	54	85	103

Verdict

Raspberry Pi Zero

Developer: Raspberry Pi Foundation
Web: www.raspberrypi.org
Price: £4/$5

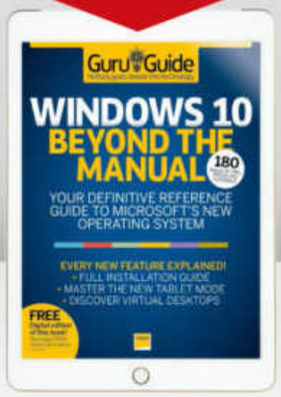
Features	7/10
Performance	5/10
Ease of use	8/10
Value	10/10

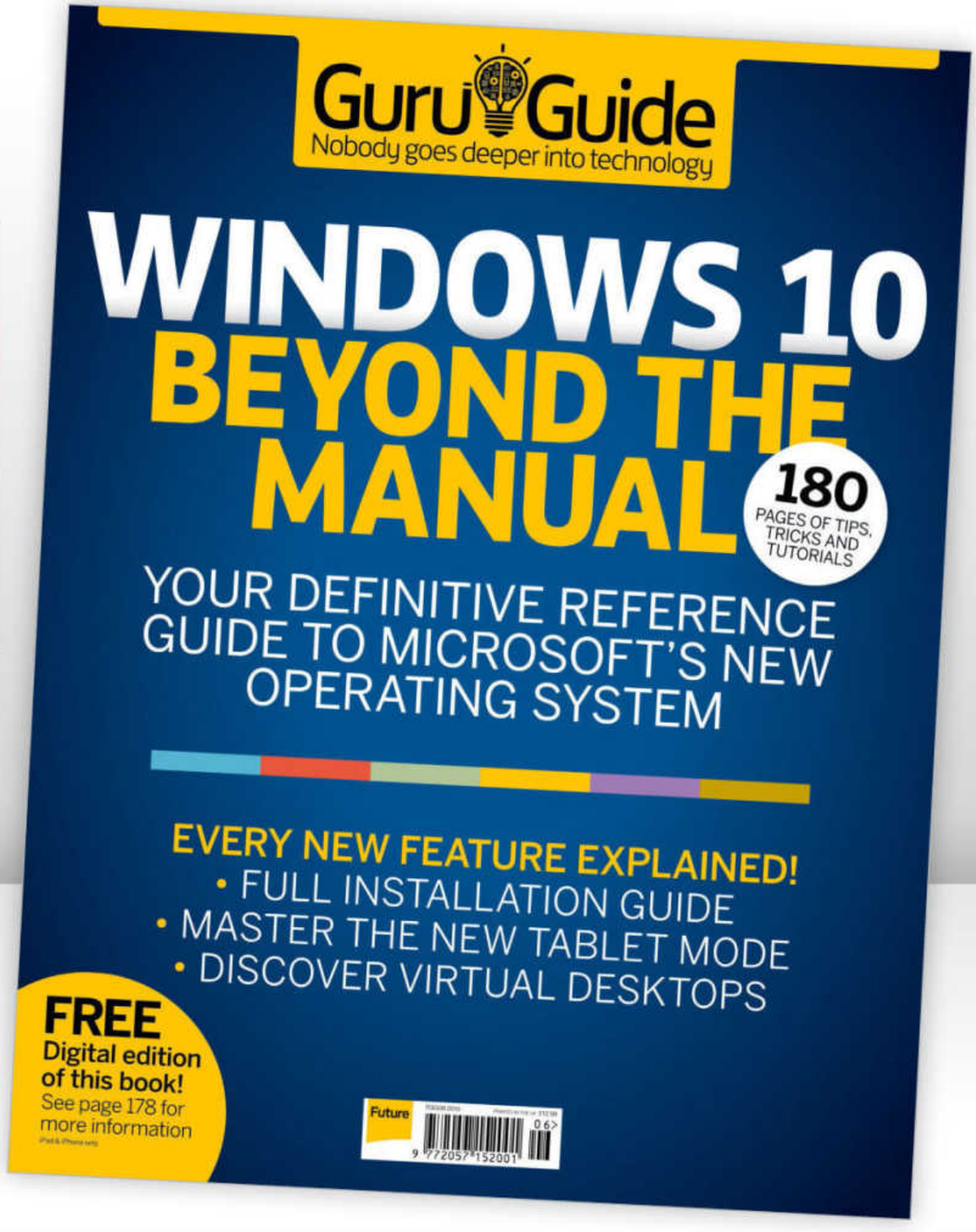
The Raspberry Pi Foundation has once again released a platform that will excite and invigorate a generation of coders. Thanks largely to a low price and massive community interest.

Rating 9/10

ALL YOU NEED TO KNOW ABOUT WINDOWS 10

OUT NOW!
WITH FREE DIGITAL EDITION





DELIVERED DIRECT TO YOUR DOOR

Order online at **www.myfavouritemagazines.co.uk**
or find us in your nearest supermarket, newsagent or bookstore!

Pipsta

A micro printer for the Raspberry Pi. But what can it do? We investigate.

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.

» **The Pipsta is a small unit, but it packs a punch when it comes to producing lovely print outs.**

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.

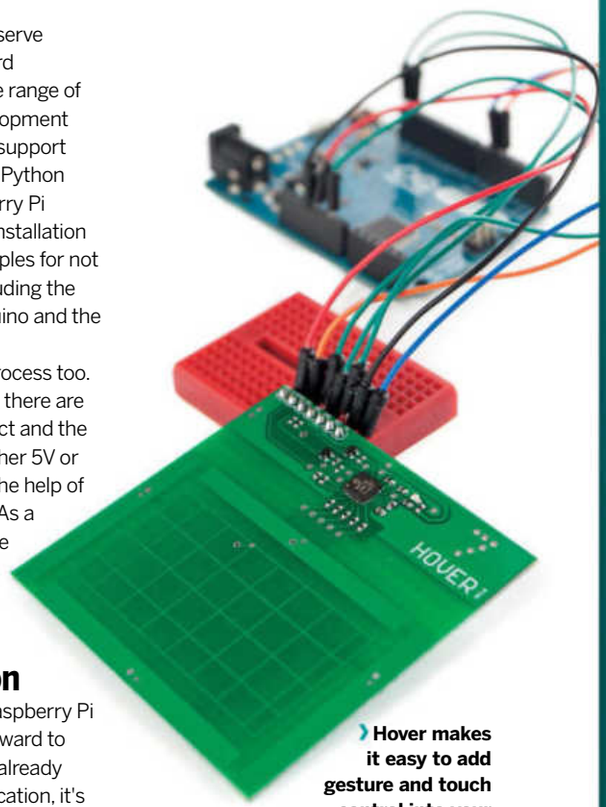
Installation is a simple process too. Besides its I2C connectivity, there are just two GPIO pins to connect and the board is compatible with either 5V or 3.3V microcontrollers with the help of onboard logic level shifting. As a rare and welcome bonus, the board's breadboard-compatible header comes pre-soldered too.

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.



» **Hover makes it easy to add gesture and touch control into your hardware project.**

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

The Kano we're looking at here 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.

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.

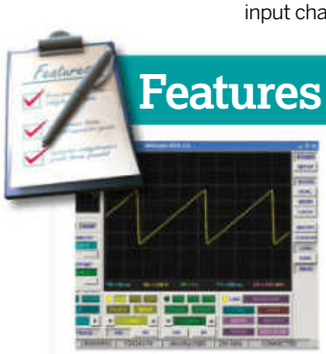
While the software provided by BitScope is not much to look at, it is certainly comprehensive, and the company has clearly put a great deal of effort into making it run efficiently, particularly on the Pi. The main application, *BitScope DSO*, has all the controls you'd find on a bench oscilloscope, 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

Agobo 2

We channel Victor Frankenstein and dabble with creating something beautiful in our lab, albeit with more metal than squiggy bits.

In brief...

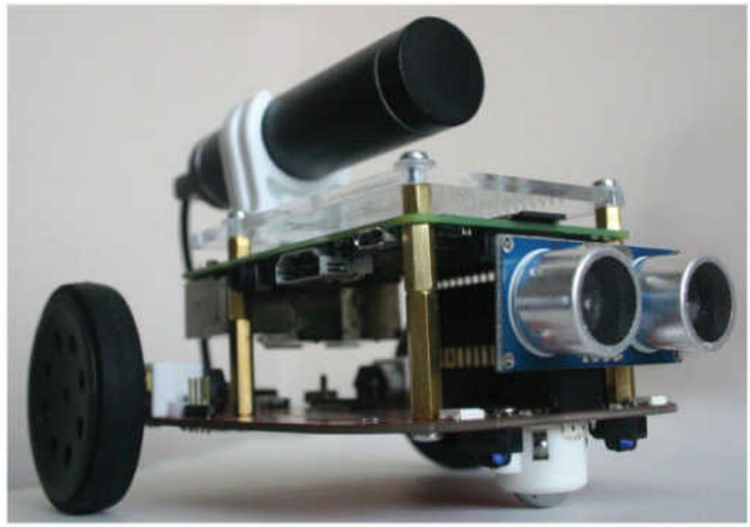
A simple but also expansive robotics platform designed to work with the A+, B+ and Raspberry Pi 2 and aimed at beginners and intermediate users. It features a strong PCB-focused design and an easy to assemble kit of parts. Control is via a well put together Python library enabling users to get started with robotics with very little friction.

Amateur robotics projects can be very personal. We build the chassis that form the robot's body, add sensors that become its eyes and ears, and we code the robot's intelligence. But getting started in robotics is fraught with pitfalls so companies such as 4tronix have put together kits to help.

The original Agobo came out in late 2014 and it provided a PCB chassis and pre-soldered components onto which a user could fit a Raspberry Pi A+ board. One criticism levelled at the original Agobo was the use of the A+, which often necessitated buying an A+ Pi just for this project. The company, 4tronix has taken that on board and has produced Agobo 2, which works with A+, B+ and Raspberry Pi 2.

This orientates the GPIO connection 90 degrees anti-clockwise so that the USB ports are at the rear of the robot and raises the larger B+/Pi 2-shaped board over the rear axles of the robot via a simple GPIO extension kit. The company is well known for using a PCB (Printed Circuit Board) as the chassis and by doing this it provides a strong platform for the robot while cramming in the electronics into one package.

Agobo 2 uses micro-gear metal motors to drive the robot along and thanks to a built-in motor controller it has precise bidirectional control of the motors. Sensor input is handled via an HC-SR04 ultrasonic sensor – similar to a parking sensor – and two line-following sensors that can detect a path drawn in front of the robot. As robotics is generally a project that doesn't necessitate a screen for operation you are generally forced to SSH into a project via Wi-Fi or Ethernet. With Agobo 2 we have a dedicated serial interface for use with USB to TTL cables, which enables a serial connection from your laptop directly to the robot. Agobo 2 also has a small momentary switch, which can be used to bring the project to life via the Python library (more on that later). Agobo 2 is powered by a small mobile phone battery, which is connected to the Agobo 2 PCB to provide a regulated power supply to the components and Raspberry Pi.



Agobo 2 uses a strong PCB chassis design that groups components into one place and creates a compact, strong and well-designed robot.

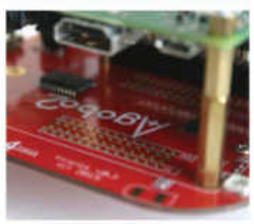
Clever robot

It's not all about the hardware, though. A good robot needs the right software and 4tronix has put together a great Python library that covers the functionality of the robot. Ultrasonic sensors have their own function called getDistance() that handles the complex calculations for computing the distance from an object using ultrasound. Line-following sensors also have their own function, which returns the current state of each sensor. Precise forward and backward motor control is possible, because of a series of functions that handle the motors together. This enables a robot to spin on the spot and turn in a graceful arc. The functions are accessible by learners of all ages and abilities, and by using 14 lines of code we were able to create a robot programmed to avoid anything within 50cm of the ultrasonic sensors.

Agobo 2 refines the original platform while retaining the same ease of use of the original, both in software and hardware terms. The kit requires only a screwdriver to build and the Python library even enables even a novice to simply get started with robotics. There are other robot kits out there but the Agobo 2 is a cost effective solution.

Features at a glance

PCB chassis
The Agobo 2 integrates the housing of the Raspberry Pi, batteries and motors into a strong single PCB.



Serial connection
Robots are mobile so it makes sense to include a serial connection via a USB to TTL lead.

Verdict

Agobo 2

Developer: 4tronix
Web: www.4tronix.co.uk/store
Price: £32.45

Features	9/10
Performance	10/10
Ease of use	8/10
Value	9/10

Its strong chassis and easy-to-build and code platform enables anyone to dip their toe into the world of robotics.

Rating 9/10

Display-O-Tron HAT

An LCD screen with lots of LEDs and a capacitive touch interface. Get ready to hack your name in lights.

In brief...

» An LCD screen featuring a fully controllable series of multi-colour LEDs and a capacitive touch interface. Some break out of the GPIO pins from the Raspberry Pi to the board enables possible integration into many different projects. A robust Python 2 library simplifies the use of the board for any level of user.

Pimoroni, the Sheffield based company of makers, has enjoyed great success with its various Raspberry Pi-related products and its latest board, Display-O-Tron HAT, looks to continue that trend.

Display-O-Tron HAT is, as its name suggests, a HAT-compliant board that fits neatly on top of all 40 pins on the A+, B+ and Raspberry Pi 2. Measuring 65mm in width and 56mm in height, the Display-O-Tron HAT matches the screw holes present on the Pi perfectly and enables the boards to be secured for projects. The main focus of the Display-O-Tron HAT is an large 16 character by three lines LCD screen that's exceptionally clear to read. Underneath the LCD screen there are six RGB LEDs, capable of creating any colour which is then diffused under the LCD screen to produce a fluid cloud of light which illuminates the LCD. There are an additional six LEDs to the right of the LCD arranged into a bar graph. Also present on the Display-O-Tron HAT are a series of capacitive touch buttons, which detect touch input which are exceptionally sensitive even through 3mm of acrylic and enable the Display-O-Tron HAT to be mounted inside of a case. Just above the LCD screen there are a series of GPIO (General Purpose Input Output) pins that are broken out from the Raspberry Pi and require a header to be soldered for general use. There's access to power and ground as well as I2C (Inter-Intergrated Circuit), UART (Universal Asynchronous Receiver/Transmitter) and SPI (Serial Peripheral Interface). There's also access to five standard GPIO pins, which means that extra components can be added to a Display-O-Tron HAT-powered project easily. This is a refreshing feature for a HAT-based board as typically they prevent access to the GPIO, which reduces the number of projects that they can be used in.

› Sporting a sleek black PCB and gold lettering, the Display-O-Tron HAT fits neatly over your Raspberry Pi.

Library upgrade

Software for the Display-O-Tron is a Python 2 library based upon the original Display-O-Tron 3000 board. Installation is relatively easy thanks to an install script that can be run in a terminal directly from Pimoroni's website. The Python library has functions to control all aspects of the HAT and we were able to able to dive in quickly and write the customary "Hello World" with a lovely colour-changing sequence courtesy of the backlight functions and a for loop.

So is the Display-O-Tron HAT a novelty item? No, in fact it's quite a versatile device. Due to its GPIO access we can connect external components, such as LEDs, buzzers and even motor controllers, which enables projects such as robots to be controlled and provide output via the HAT.

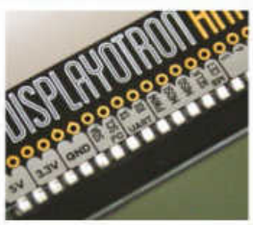
The clearance of the board to the Pi underneath is also sufficient for adding the official Raspberry Pi camera, which opens up the prospect of using the board as a timelapse controller for a nature or science project.

The only downside of this board is the Python library. Python 2 is still relevant but it would be great to see the library being updated with Python 3 support, which has been done with previous Pimoroni boards, such as the Explorer HAT Pro.

The Display-O-Tron HAT is a great board to hack with. By providing a solid board and Python library you'll be able to add an LCD display to a project without a mass of wires or add a touch interface rather than a traditional micro switch/pushbutton mechanism. Hacking should always be fun and with this HAT we have a fun platform for hackers of all abilities.

Features at a glance

LCD screen
A large 16x3 LCD screen dominates the board. It's clear and crisp display is legible in direct sunlight.



GPIO access
Being HAT-compliant the Display-O-Tron covers all 40 GPIO pins, but also breaks out some pins.

Verdict

Display-O-Tron HAT

Developer: Pimoroni
Web: http://bit.ly/DisplayOTronHAT
Price: £22

Features	8/10
Performance	9/10
Ease of use	9/10
Value	9/10

» *A great board for your next project. Well built and well supported by an easy to use Python 2 library.*

Rating 9/10

Raspberry Pi display

We get hands on with the long-awaited official Raspberry Pi touchscreen display. Careful with those jam-smeared fingers.

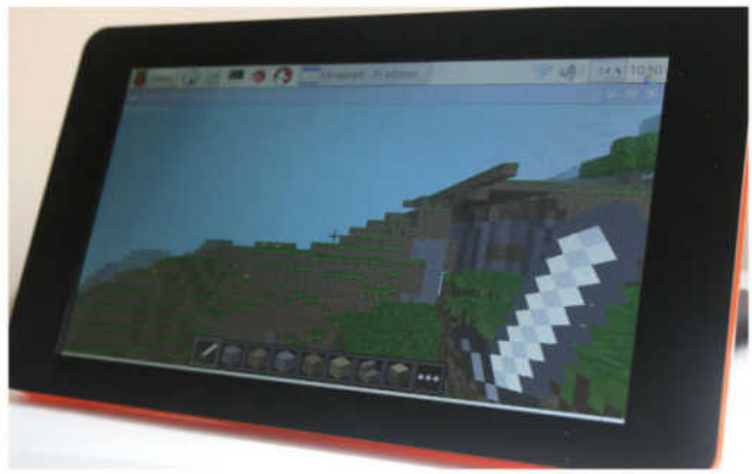
In brief...

A touchscreen developed by the Raspberry Pi Foundation that supplies a 7-inch touch interface for projects. It's designed to be portable and easy to use for maker embedded projects or classroom work.

Despite selling six million units, the Raspberry Pi has one port that has never been used: the display. Located on the reverse of the Micro SD slot, the display port was designed specifically for the newly released official Raspberry Pi display.

The official Raspberry Pi display features a seven-inch screen with a resolution of 800x480 pixels and provides a capacitive touch interface. The display is surrounded by a black bezel which gives the appearance of a tablet, which you need to be careful with as it's rather thin and can be easily bent. The display doesn't come with a case or stand as the Raspberry Pi Foundation is keen for the community to fabricate their own and companies, such as Pimoroni, to meet the need.

On the back of the screen are two thin connectors for the video and for the touch interface and these connect to a driver board. This board has three ribbon connections: two from the display and a third which connects to the Raspberry Pi Display port. Your Pi can sit on top of the driver board and is secured in place by four screws. Power to the display is provided by a Micro USB port which is best powered by a 5V 2A power supply. The Pi can then be powered in two ways: By connecting to the USB port via a USB to Micro USB lead, or by connecting two jumper cables from the 5V and Ground pins on the driver board to the Raspberry Pi, the latter option will prevent the use of add-on boards.



The official Raspberry Pi 7-inch display provides a portable and neat touchscreen for use in classroom learning and hackspaces.

If the latest official Raspbian image is used, the Pi will automatically detect the correct resolution for the display

Software installation is exceptionally easy and requires nothing more than a simple `sudo apt-get update` and `sudo apt-get upgrade` to install the touchscreen drivers. This is a refreshing change to other displays, such as Adafruit's 5-inch touchscreen which uses the AR1100 touch controller and requires calibration via a Windows PC.

Portable touch

The touch interface on the Raspberry Pi display reacts to input very quickly. We tested it with a Pi 2 and found no stuttering or slowdown. The interface can detect ten points of touch at once, enabling multitouch possibilities for your projects. For general desktop use, the touch works well, but there's currently no right-click functionality, which is a limitation of X not really being designed for a touchscreen.

The display is bright and has a comfortable viewing angle and while the resolution is smaller than we are now accustomed to there's still plenty of space for hacking in your favourite application. The display can be used with an HDMI screen, opening up the prospect of a dual display system, but that requires a few configuration changes for applications to use the correct screen.

At this time the display is sold as a kit, which requires a small amount of assembly, but it's envisaged that future revisions will come pre-built.

The official display will not replace your main monitor but it does fulfil two needs. First, the need for a portable screen for use in hackspaces and classrooms. The size of the display and its price-point lend it well to this. Second, the display can be easily embedded into a project enabling the Pi to power a number of touchscreen controlled solutions.

The official Raspberry Pi 7-inch display is a lovely piece of kit. As it just works as a touchscreen, we will see this display powering a slew of great projects in the coming months.

Features at a glance

Driver board
Handles the display, touch interface and provides power and an I2C interface to your Raspberry Pi.

Easy assembly
The displays come as a kit but to assemble one doesn't require any specialist knowledge.

Verdict

Raspberry Pi Display

Developer: Raspberry Pi Foundation
Web: www.raspberrypi.org
Price: £48

Features	7/10
Performance	7/10
Ease of use	8/10
Value	9/10

Doesn't come with a stand or case and isn't hi-res, but does offer cost effective embedded touch for projects.

Rating 8/10

Raspbian Jessie

Let's take a detailed look at the latest Raspbian release from the Raspberry Pi Foundation.

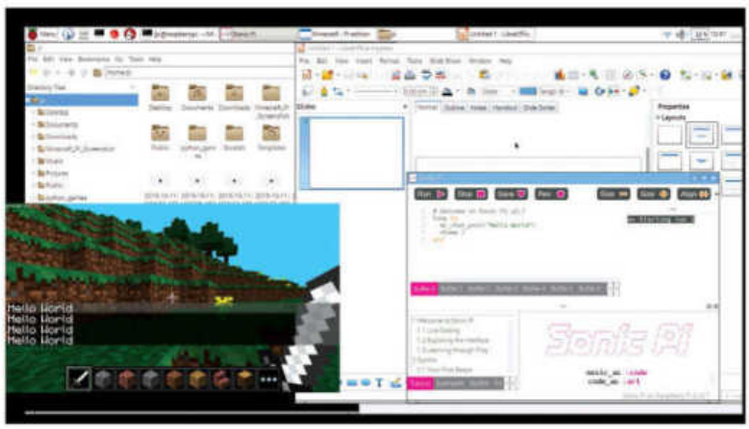
In brief...

The latest release of the official distro supported by the Raspberry Pi Foundation. The distro is based upon Debian Jessie and comes with a raft of improvements to hardware and software, which sees the distro mature into an all-encompassing platform for hackers and makers as well as a cost-effective second computer.

Since its début in 2012, the Raspberry Pi has used Raspbian as the official operating system. Looking back at the early versions of Raspbian, we see a rather basic OS that was still in its infancy, but with the latest release of Raspbian we have a whole new beast.

The latest Raspbian is based on Debian Jessie and ships with kernel 4.1.7 by default, and refines the desktop re-design spearheaded by Simon Long in 2014 by shifting to the newer *GTK 3* toolkit. When you first boot up you will see a major difference to the boot process; it now boots to the Raspbian desktop by default, but this can easily be changed. Raspbian Jessie also ships with a new way to make system configuration changes and this is the *Raspberry Pi Configuration* application, a GUI for the stalwart *raspi-config*, which is used to overclock your Pi among other things. Adding applications to the main menu is now made easier thanks to the *Main Menu Editor*, similar to *Alacarte*, which has been written in Python. A nice touch is the inclusion of *scrot*, an application used to take screenshots, something we use a lot when distro hopping.

To further cement the idea that the Raspberry Pi can be used as a typical desktop computer, the Raspbian Jessie comes with the *LibreOffice* suite and the *Claws Mail* email client. There are also two new Java IDE: BlueJ and Greenfoot in the Programming menu taking advantage of the Pi 2's hardware.



The Raspbian desktop environment, thanks to the work of Simon Long, is now a slick experience with a very refined user interface.

Perhaps the biggest change is under the hood. Typically, only the root user or a user with sudo access is able to use the GPIO pins. The workaround is using IDLE, the Python editor, where you'd need to open a terminal and run IDLE with `sudo`. This isn't an issue anymore, as now any user can access the GPIO pins via IDLE in the Programming menu. This is a major change and will enable a much easier transition for those learning to code via the GPIO.

Powerful platform

Talking of Python, Raspbian Jessie has a new version of the popular Pygame library called Pygame Zero. This is commonly used to make games with Python but it has a rather steep learning curve. It's a simpler version of Pygame with a focus on helping educators wanting to enrich coding lessons. Any code written for Pygame Zero uses a text editor or IDLE, but to run it you will need to open a terminal and run `pygzrun` along with the name of your project.

With this latest release we see why Raspbian is considered to be the de facto distribution (distro). Its mix of thoughtful refinements to established applications, *raspi-config* and sudo-less GPIO access, and new software, including *LibreOffice*, which enables the Pi to meet the needs of different user groups and not just coders.

Raspbian Jessie does come at a cost, the install size is well over 4GB, leaving approximately 3GB of available space on an 8GB SD card. This isn't too high a cost as Raspberry Pi comes with 8GB micro SD cards. But for those of you with older Pi running on a 4GB SD card or the Raspberry Pi Compute, you will need to wait a little longer for a planned 'lite' version.

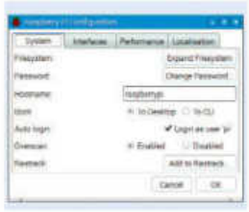
We tested the new release on a Raspberry Pi B+ with 512MB of RAM and can report that it was perfectly usable even with less RAM. However, with 256MB of RAM that's present on the A and A+ the Pi was slower but this is to be expected.

The Raspberry Pi continues to dominate the single board computer community and the latest Raspbian release will further ensure its dominance, despite a growing threat from the Ubuntu Mate distro.

Features at a glance

GPIO access

The first release to feature GPIO access for all users. Previously only root or sudo could use the pins.



Settings

The new suite enables anyone to tinker with their setup, from keyboards to overclocking their Pi.

Verdict

Raspbian Jessie

Developer: Raspberry Pi Foundation
Web: www.raspberrypi.org
Licence: Various

Features	9/10
Performance	9/10
Ease of use	7/10
Documentation	7/10

With improved user experience, thanks to refining existing tools, we see a powerful platform emerging.

Rating 9/10

Get the UK's best-selling Linux magazine

DELIVERED DIRECT TO YOUR DOOR

Order online at **www.myfavouritemagazines.co.uk**

or find us in your nearest supermarket, newsagent or bookstore!

Pi-Top laptop kit

We get hands-on with an education-focused Raspberry Pi laptop kit that promises to make the class *green* with envy...

In brief...

A Raspberry Pi laptop solution offering an impressive level of portability and features that are aligned with the UK GCSE curriculum. But these features come with a hefty price tag and a number of rough edges to a relatively new software base. That said, this is a powerful package for those who can work around the issues.

Right now the Raspberry Pi sports a plethora of portable options. September 2015 saw the release of the new touch-screen, and other companies such as Kano are working on their own portable setups. Pi-Top started life as a crowd-funded project and combines both hardware and software. The hardware is a sturdy plastic laptop shell (supplied as a kit) and includes a 13.3-inch HD (1,366x768) LCD screen with eDP interface, an 8GB SD card, a battery with a claimed life of 10 hours, and a Wi-Fi dongle. It comes with or without a Pi 2. The kit is fairly simple to assemble but may require adult supervision for a few fiddly bits, such as attaching the LCD screen to the driver board.

The driver board handles connecting the Pi to the built-in battery, recharging the battery via an included external power supply, and sending HDMI video input to the LCD screen. Connecting the GPIO of your Pi to the driver board enables battery management, but will cover all 40 GPIO pins; it can be removed, enabling use of the GPIO. The driver board and the Pi (located to the right of the laptop) are covered by a slide cover, for quick access to the Pi. Access to the Pi's USB and Ethernet ports is tricky but possible. The keyboard and trackpad are fine for daily use but the keyboard can feel a little spongy and imprecise at times.

On the software side, you get the pi-topOS, built upon Raspbian Wheezy 7.8. Pi-topOS acts as a layer on top of the Raspbian OS. On first boot you are prompted to set up your Pi-Top, which includes creating an online account. This is used to save your learning progress to the Pi-Top cloud-based learning system, which is aligned with the UK Computing curriculum for 13 to 15 year olds.

The Pi-Top comes with a bundle of applications similar to Raspbian. These include Scratch, *Libre Writer, Minecraft Pi, Sonic Pi* and the Python editor IDLE. We tested IDLE 3, the Python 3 editor, with the RPi.GPIO library, the most popular library for hardware hackers and makers. Being based on Wheezy, IDLE3 was unable to access the GPIO using the default user – it required opening a terminal and running the command via sudo – but we successfully built and tested a simple LED project. This is a step backward for those used to the latest Raspbian Jessie image, but we'd expect this to be fixed in a future release. Of course, since the system is based on Raspbian, you are free to install your favourite applications via the package manager.

Pi-Top isn't quite your typical laptop, and under the lid it's all Raspberry Pi powered.

Learning experience

During our tests there were a couple of issues. Logging in as an incorrect user prevented us from re-attempting a login with the correct details. A reboot solved this issue, but it did take time. Also, an update bug prevented pi-topOS from connecting to the update server despite constant reminders and using an Ethernet cable to connect to the router.

Pi-Top also has its own software, a game called *CEED Universe*, which teaches coding and making concepts via an interactive retro game – a great idea that keeps children engaged while learning key skills. It's reminiscent of Kano OS, which itself uses gamification to teach core concepts. *CEED Universe* is a lot of fun and provides a great level of interaction for children wanting to learn more. It's a great idea and provides a portable Pi solution, but the cost of $300 (around £200) is high, with just a 30-day warranty, for the education market.

Features at a glance

Practical package
The Pi-Top laptop case provides easy access to the GPIO via a sliding panel above the keyboard.

Applications
Standard Raspbian apps are included, such as *LibreOffice* and the *Chromium* web browser.

Verdict

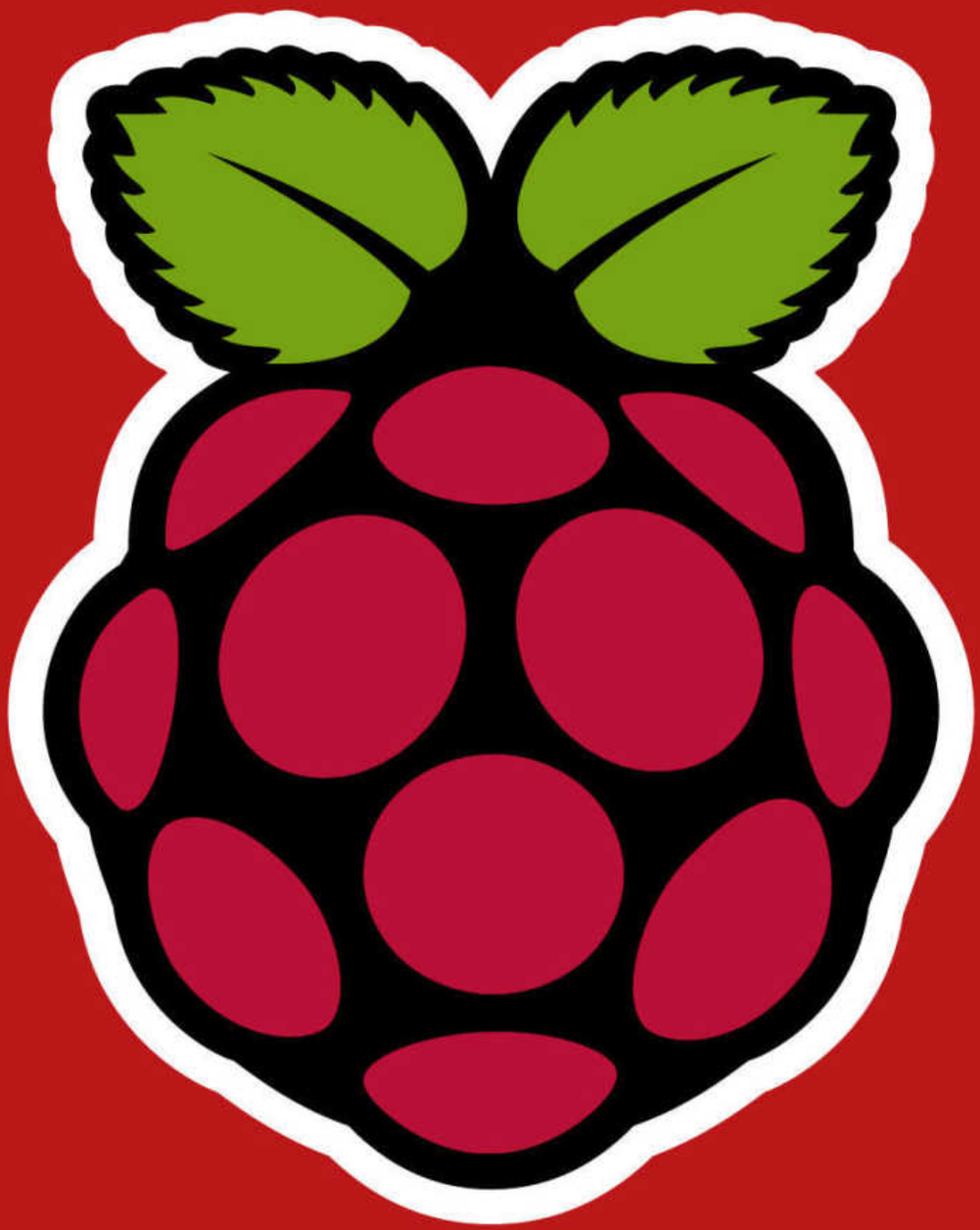
Pi-Top

Developer: CEED Ltd
Web: www.pi-top.com
Price: $300 with a Pi 2, $270 without

Features	7/10
Performance	8/10
Ease of use	7/10
Value	5/10

A great idea but not without issues. The software bugs will doubtless be tackled, but the cost is the main barrier.

Rating 6/10



Top projects

Put that Pi to work.

RASPBERRY Pi

The best new projects

We locked our tame expert in his garden shed with a Raspberry Pi 2 – he's emerged, blinking into the light with nine great projects for all levels of user.

The Raspberry Pi 2 is more powerful than we could have hoped for. To celebrate this, and total Pi sales sailing past the 5 million mark, we're diving into the best Raspberry Pi projects. If you're either a beginner or an old hand, or just someone itching to do more with your Raspberry Pi 2, there are projects here that will push your Pi and your brain to the limits.

Most of the projects will work on the original Raspberry Pi Model B, but we've indicated compatibility between boards and what you'll need for each one.

We've got Pi projects that embrace software, hardware and coding. Projects include building a cutting-edge *Ghost* blog to taking your first steps into the world of robotics. If you have a Raspberry Pi 2 to test out then why not try to get a full desktop version of Ubuntu up and running? It's impressive to behold. The Raspberry Pi has truly become a worldwide phenomena, it's already the best-selling UK home computer ever made and it's only going to go from strength to strength as it has suitably forward-looking leadership from Eben Upton and the Raspberry Pi Foundation. So if you want to get onboard there's no better way than throwing yourself into these projects and picking up some vital Linux and Pi knowledge. And that's a key point: you're not locked into working with the Pi only, much of this is FOSS/Linux knowledge, so you'll be able to take it with you and work on desktop and server systems.

> **"Throw yourself into these projects and pick up some vital Linux and Pi knowledge."**

Build a blog

Get that rant off your chest – install a lightweight but elegant Ghost blog using Node.js, Nginx and a Raspberry Pi 2.

Blogs don't need heavyweight hardware and thanks to the Raspberry Pi 2, we can create a slick, responsive blog using three great open source projects: *Node.js*, *Nginx* and the *Ghost* blogging platform.

We start with a fresh Raspbian install that has been configured to run an SSH server. The easiest way to do this is via the *raspi-config* Advanced menu. We'll use an SSH client to log in to the Raspberry Pi remotely, but first we'll need the Pi's IP address, which you get by running **ifconfig** from *LXTerminal* (access it via the icon on Raspbian's desktop).

If you're connected via Ethernet look for **eth0** and if you're using Wi-Fi it's **wlan0**. You're looking for the **inet addr** and the number after this is the internal IP address of your Pi. Using an SSH client on your computer (this comes as standard with Linux distros) log in to your Raspberry Pi:

```
ssh pi@IP_ADDRESS_OF_PI
```

Once connected, you'll need to download *Node.js* and change directory to extract and install it:

```
sudo wget http://nodejs.org/dist/v0.10.28/node-v0.10.28-linux-arm-pi.tar.gz
cd /usr/local
sudo tar xvzf ~/node-v0.10.5-linux-arm-pi.tar.gz --strip=1
```

Now download and install *Ghost* into a directory in our **home** directory:

```
sudo mkdir ghost
cd ghost
wget https://ghost.org/zip/ghost-0.5.8.zip
unzip https://ghost.org/zip/ghost-0.5.8.zip
```

Node.js uses *npm* a packaging tool, which we'll use to install *Ghost* as follows:

```
sudo npm install --production
```

This will take some time to complete. Now run:

```
sudo npm start
```

This command will start the *Node.js* service and run *Ghost* in development mode. At this time we're simply testing that everything has been done correctly. To stop the server press Control+C. Now lets install *Nginx* to act as a proxy:

```
sudo apt-get install nginx
```

We now need to configure *Nginx* to work with *Ghost*. In *LXTerminal* change the directory to where *Nginx*'s config files reside and delete the default configuration file:

```
cd /etc/nginx/
sudo rm sites-enabled/default
```

Next, we need to change directory to **sites-available** and create a new file called **Ghost** using the *nano* editor.

```
cd sites-available
sudo nano ghost
```

This file needs to contain the configuration to connect *Ghost* to *Nginx*, enabling users to access the blog:

```
server {
    listen 0.0.0.0:80;
    server_name ghostblog.com;
    access_log /var/log/nginx/*your-domain-name*.log;
    #root /home/pi/ghost;

    location / {
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header HOST $http_host;
        proxy_set_header X-NginX-Proxy true;

        proxy_pass http://127.0.0.1:2368;
        proxy_redirect off;
    }
}
```

Save the file (CTRL+O) and exit *nano* (CTRL+X), and change directory and create a symbolic link:

```
cd ..
ln -s sites-available/ghost sites-enabled/ghost
```

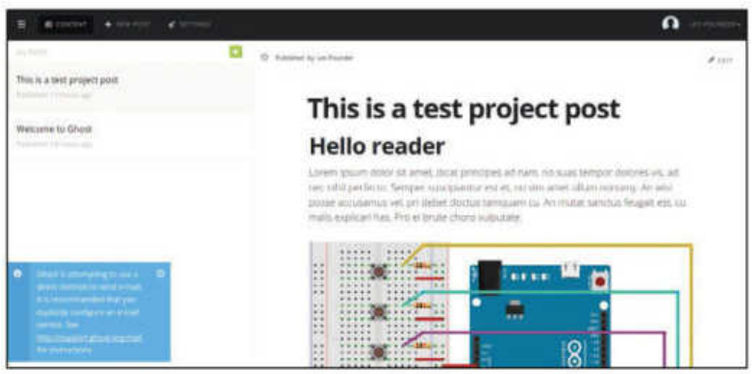
Nginx will listen for traffic on port 80 and will redirect it to *Ghost*. Last, we need to start the *Nginx* server and *Ghost*.

```
cd /home/pi/ghost
sudo service nginx restart
sudo npm start
```

On another device use a browser to navigate to the Pi's IP address, where you'll see the *Ghost* login screen. To learn more about Ghost head to **www.ghostforbeginners.com**.

For this project you will need

- A Raspberry Pi 2 model B, Zero, or a Raspberry Pi 1 model B or B+
- Raspbian OS
- Wi-Fi or Ethernet connection
- SSH Server running on your Raspberry Pi



***Ghost* is an elegant and simplistic blogging platform that's scalable for all types of devices thanks to its responsive design.**

What's LAMP?

In the past setting up a blog was quite a resource-intensive process. For starters you'd need to install a LAMP stack, which stands for Linux (the OS component), *Apache* (the web server component), *MySQL* (the database framework for storing content) and PHP (the scripting language that enables webs page to pass information to *MySQL* and create HTML content on the fly). All of these are used today for full-blown sites, but there's a growing trend for new technologies to replace this bloated process.

The new(ish) kids on the block are *Node.js* and *Nginx*. *Node.js* is a JavaScript framework that runs on the server and not in the user's web browser. This is used to power *Ghost*; from its user interface to its database capabilities.

Nginx is used to handle serving the web pages to users in a scalable way. This is a high-performance lightweight HTTP server and reverse proxy that's gaining a significant hold, where *Apache* was once dominant.

Remote control your Pi

Use your Raspberry Pi from across the room or even from across the world using the magic of SSH and VNC.

For this project you will need

- A Raspberry Pi 2 model B, Zero or a Raspberry Pi 1 model B or B+
- Wi-Fi or Ethernet connection.
- Keyboard, mouse and monitor for set up
- Raspbian OS
- Power supply

Using your Raspberry Pi is normally accomplished via a keyboard, mouse and monitor but at times this might be awkward, eg when controlling a robot or time-lapse photography rig. Remotely controlling a computer is nothing new but it's a handy solution to accessing your hard to reach Raspberry Pi project. In this project we'll install an SSH server, which will enable us to remotely control the Raspberry Pi using the terminal from another machine. We'll also install a VNC server which will enable us to use the Raspberry Pi desktop over a network.

Connect up your Raspberry Pi and boot Raspbian to the desktop, you may need to log in using your username and password. If your Pi doesn't boot straight to the desktop type **startx** in *LXTerminal* and press Enter. If you're using a Wi-Fi dongle, make sure that it's configured and has an IP address. If you're using an Ethernet connection, open LXTerminal and type **ifconfig** to find your IP address.

Now we must configure the software that will run on the Pi and give us the access that we need. Open *LXTerminal* and type the following to setup an SSH server:

```
sudo raspi-config
```

Navigate to the Advanced Options Menu and look for SSH server, enable it and exit the application. If you're asked to reboot, do so and return to the desktop. With the SSH server installed we can now test that it works using another computer. We're using a laptop running Linux Mint and SSH'd into the Raspberry Pi using the terminal:

```
ssh pi@IP OF YOUR PI
```

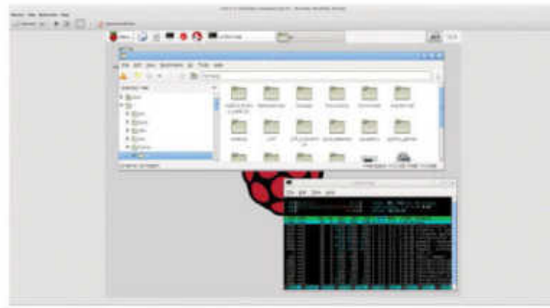
The first time that you connect to the Raspberry Pi, SSH will ask you to confirm that the Pi is what you think it is via a unique fingerprint, and for this project we can accept that it's correct. After a few moments you will be asked for your Raspberry Pi password and need to successfully log into your Pi. Any command issued from this point onwards will be run on the Raspberry Pi.

SSH comes as standard with Linux and Mac, whereas Windows users need to download *PUTTY* which is a free application.

```
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.0.8' (ECDSA) to the list of known hosts
pi@192.168.0.8's password:
Linux raspberrypi 3.12.35+ #730 PREEMPT Fri Dec 19 18:31:24 GMT 2014 armv6l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue Feb  3 19:17:48 2015 from 192.168.0.3
pi@raspberrypi ~ $ uname -a
Linux raspberrypi 3.12.35+ #730 PREEMPT Fri Dec 19 18:31:24 GMT 2014 armv6l
pi@raspberrypi ~ $
```



We used *Vinagre*, a remote desktop viewer built into Linux Mint and Ubuntu to view the Raspberry Pi desktop.

With the SSH server successfully installed, our attention shifts to installing the *VNC* server. In the SSH session run the following command:

```
sudo apt-get install tightvncserver
```

This installs the VNC server and to run the server type:

```
tightvncserver
```

You'll be asked for a password – use a strong one with a maximum of eight characters. You will also be asked for a view-only password that isn't needed at this time.

In the terminal you'll see the output of the **tightvncserver** command, and it will advise you that a VNC session has been started and that you can connect using the IP address or hostname followed by :1.

Now we need a *VNC* client on our computer to connect to the server running on the Pi. We've used *Vinagre* which comes installed on Linux Mint and Ubuntu. Open *Vinagre* and click on Connect, in the host box enter the IP address of your Raspberry Pi, followed by :1 and click Connect. You will be asked for your VNC password. With the correct password entered you can now use your computer's mouse and keyboard to control your Raspberry Pi and the Raspbian desktop is visible on your desktop.

For those of you looking to play *Minecraft* in this manner, we're sorry to say that, at this time, it's not possible due to the way *Minecraft* renders itself on the Raspberry Pi. Generally though, you're now able to use your Pi remotely from another computer in your home. In fact, you can have more than one *VNC* connection running, which will enable two people to share a Raspberry Pi.

What's SSH and VNC?

SSH is used by system administrators to remotely work with servers across the world, for instance, a VPS or server running a website or web application. Because there's no physical access to the device a user has to remote in using SSH either via a terminal, or using a web interface provided by a host.

SSH connections are encrypted and it's highly unlikely that anyone can see what's being done, unlike VNC which should not be used over the internet. VNC sends its information via an unencrypted connection and any VNC ports that are open can be scanned by malicious parties. In fact, there's a website that lists websites with open VNC ports and enables you to connect just like *Chat Roulette*. However, tunnelling a VNC connection via SSH, grants you a level of security.

There are SSH applications for all OSes and on Linux it comes by default, and VNC applications are installed with Ubuntu and Linux Mint in the form of *Vinagre*.

Build a SmartTV

Build a media centre and make your TV smart using OpenELEC and a little bit of Pi tinkering.

COMPATIBILITY
All Pis but best on Pi 2

We have many ways to consume media these days but our main method is still through our TV (even it might be via streaming technology). The Raspberry Pi is well suited to displaying media and since the day of its release users around the world have been creating media centres based on the Pi. In this project, we'll show you how to do it for yourself.

We start this project by copying the operating system to the SD card. For this we'll need to extract the contents of the file downloaded from the OpenELEC website. This will extract an image file, which we need to copy to the SD card. To copy the image to our SD card follow the guidance at **http://bit.ly/LXF_OpenELEC**. Note: For whatever version of Raspberry Pi that you use, ensure that you have the correct image file as Pi 1 and 2 images are currently not compatible.

With the image copied to the SD card, insert it into your Raspberry Pi along with your wireless keyboard and mouse and other connections, such as Ethernet and HDMI to your TV. The last connection to make is the power which will boot your Raspberry Pi.

On your first boot up OpenELEC will lead you through a series of screens that will set up your media centre. Once complete you'll be returned to the main menu which is split into sections to catalogue your pictures, videos and music. With any of these categories you can easily import content from a variety of sources and the following steps are repeatable for all types of media that you wish to import. If you haven't already inserted a USB hard drive, with movies copied on it, into your Raspberry Pi and then navigate to the Videos menu and then to Files. From here a new dialog box will appear and you'll be able to browse to the USB drive and

Album art is automatically downloaded thanks to a scraping tool that uses popular online music websites.

OpenELEC uses the Kodi, formerly XBMC, user interface, which has matured along with the project, and supplies a slick and seamless method of input.

the movies contained therein. Your movies will appear in the Videos library and you can select and play them from the list. Content can also be stored on network drives, such as a NAS, and OpenELEC can connect via many standards such as SSH, NFS and Samba, enabling content to be centralised in your home and available to multiple media devices.

OpenELEC also comes with a series of add-ons that can enhance the experience and you can find a selection of these in the Add-on menu for each category, eg a great video add-on is the *Revision3* channel that broadcasts lots of maker and hacker-based shows. You can easily install the add-on by finding it in the list and clicking Install, and OpenELEC will handle everything from there on. For those of us in the UK, you can also install BBC iPlayer functionality thanks to Kodi (see **http://kodi.wiki/view/Add-on:IPlayer add-on**).

Controlling OpenELEC using a wireless keyboard and mouse is the default method, but there are other more stylish and consumer-friendly ways of doing things. First, there's FLIRC, (available from the PiHut store **http://bit.ly/PiHutFLIRC**) which is an infrared receiver that can be programmed to use your existing remote to control OpenELEC. Alternatively, if you have a spare Android tablet knocking about there's a free app called *Yatse* (**http://bit.ly/LXF_Media_Remote**) that turns your tablet into a multimedia controller. This app can show your media catalogue on the tablet screen, and enable you to browse and select media which is then played on your television.

For this project you will need

- A Raspberry Pi 2 model B (for best results) or a Zero or Raspberry Pi 1 model B or B+
- OpenELEC
- Blank SD/Micro SD card
- Ethernet connection
- USB hard drive or flash drive
- Wireless mouse and keyboard

What's OpenELEC?

Media centres were one of the first projects to emerge for the Raspberry Pi on launch and they have gone on to become increasingly popular. In fact, they are so popular that the Raspberry Pi Foundation has invested in their development. Speaking to Eben Upton recently, he said that he's a keen user of OpenELEC and is extremely happy with the performance of the latest version on the Raspberry Pi 2 with it. While the Raspberry Pi 1 and 2 both share the same Videocore IV GPU, which means that they both can easily work with large 1080p video files. The OpenELEC user interface is also quite a CPU-intensive process and it was common for users of the original Raspberry Pi to overclock their Pis to coax out every last ounce of performance. For Raspberry Pi 2 this isn't the case, because of the immense improvements made to the CPU and RAM, and means that Raspberry Pi 2 is clearly the stable base that you need to build your media centre upon.

Install Ubuntu

Installing Ubuntu on the Raspberry Pi 2 is now possible thanks to a great community project based on Snappy Ubuntu.

For this project you will need

- A Raspberry Pi 2 model B
- 4GB SD card
- Peripherals to use with your Raspberry Pi 2, such as mouse and keyboard

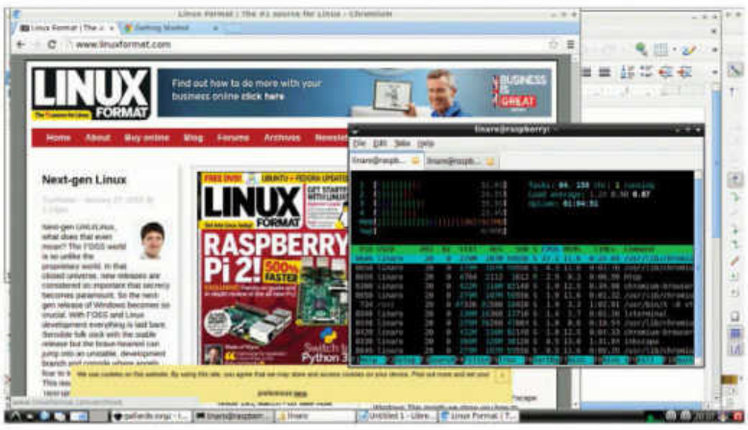
When the Raspberry Pi was first announced in late 2011 there were murmurs of Ubuntu support, but alas it didn't materialise because of the choice of CPU powering the original Raspberry Pi. This situation continued for three years until the release of the Raspberry Pi 2 and its Arm7 CPU, which has enabled Ubuntu to be installed on your Raspberry Pi 2. In this project we'll install it and configure it for daily use. (Note: this version of Ubuntu is still in its early stages.)

First download the Ubuntu image from **http://bit.ly/Raspuntu** and then extract the contents using an archive manager. This will leave you with a 3GB image file. This needs to be written to your SD card using the **dd** command.

With Ubuntu on your SD card, insert it and your mouse, keyboard, HDMI and power on your Raspberry Pi. Ubuntu will boot to a login screen within 30 seconds.

You will see the username **linaro** on the login screen, make sure that it's selected and enter **linaro** as the password and press Enter. After a few seconds the desktop will load. You will notice that it's not the default Unity interface, rather it's the LXDE desktop, as it's lighter on system resources.

Next, we will install some applications, but first we will need an internet connection. If you have a Wi-Fi dongle make sure that it's inserted into your Pi and then go to the menu and navigate to Internet and select wpa_gui. If your Wi-Fi dongle is listed in the Adapter menu then you can connect using Wi-Fi, if not then plug in an Ethernet cable to continue. For those that are keen to hack Wi-Fi you can create a config file by typing:



The Ubuntu desktop is handled via LXDE, a lightweight desktop environment.

***Inkscape* is a resource heavy vector illustration application and we were really able to pushed it.**

```
sudo leafpad  /etc/wpa_supplicant/wpa_supplicant.conf
```

Inside the file, type the following (inserting your SSID and password in the relevant sections):

```
network={
ssid="your network's ssid here"
psk="your network's password here"
proto=RSN
key_mgmt=WPA-PSK
pairwise=CCMP
auth_alg=OPEN
}
```

For those following the Wi-Fi instructions, once you've completed editing the file, save it and reboot the Raspberry Pi then log back in.

So now lets install some software. First, we shall make sure that our system is up to date. Open the terminal once again and enter the following commands:

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

With our system up to date, let's install *LibreOffice*, the free office suite, by typing the following:

```
sudo apt-get install libreoffice
```

This will take a few minutes to download and install but once installed you can find it in the Office menu. Other applications to install are *Gimp* and *Inkscape*, these are image editing applications and can be found in the Graphics menu.

```
sudo apt-get install gimp
sudo apt-get install inkscape
```

Ubuntu won't replace Raspbian as the default distro for Raspberry Pi, but it's great to see another alternative distro. For the latest developments keep an eye on the official forum at **http://bit.ly/UbuntuForRP2Forum**.

Debian roots

Ubuntu and Raspbian both come from the same Debian upstream source. This means that you can install applications in the same manner for both distros. At the time of writing you can only install from the Ubuntu repositories for 14.10, Utopic Unicorn, but the community are hard at work bringing applications from the Raspbian repositories to the Ubuntu repos. During this tutorial we tested to see if we could add the Raspbian repos to our list of sources and while they imported without an issue, when we tried to install applications from the Raspbian repos it caused many issues with our system and so we stuck to the Ubuntu repository.

Currently, the Ubuntu project for Raspberry Pi is based on Snappy Ubuntu, an extremely lightweight version of Ubuntu that's intended to work with IoT (Internet of Things), and enable makers to use Ubuntu as a base in their projects. You can download the minimal Snappy Ubuntu from the Raspberry Pi website.

Build an arcade cabinet

Partake in some gaming nostalgia – emulate old consoles and retro titles with your Raspberry Pi.

In the 1980s and 90s there was an explosion of gaming consoles and home computers. Computer users of a certain age will have fond memories of machines such as the ZX80, BBC Micro and Megadrive/Genesis. These machines are now quite rare and this is where emulation plays a big part in helping re-live those glory days.

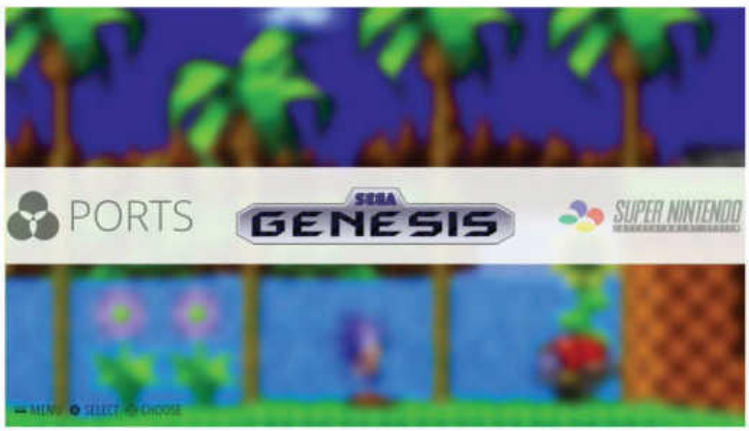
The Raspberry Pi is a popular choice for emulation as it's powerful enough to run emulators from Atari to upright arcade cabinets using MAME. The release of the Raspberry Pi 2 has also meant we now have an even more powerful processor that will enable emulators for later consoles, such as the PlayStation One, to run more smoothly.

We shall be using the *RetroPie* emulator (**http://bit.ly/RetroPieProject**). You'll need to copy the downloaded image to a blank high capacity SD card, the larger the better, and you'll need to use the **dd** terminal command to copy the image to an SD card.

With *RetroPie* on your SD card connect all of your peripherals, including any USB gamepads that you wish to use, to your Raspberry Pi and power up. On first boot *RetroPie* will ask if you would like to configure your joypad to work with the user interface. Note: There's a separate configuration for each of the emulators due to their differing joypad layouts. If you have a wired Xbox 360 controller there's a great guide on the RetroPie GitHub (**http://bit.ly/RetroPieXbox360SetUp**) for helping set it up.

RetroPie uses the popular *Emulation Station* as its user interface and by default there are a number of systems that can be emulated out of the box. These systems are identifiable by being active on the user interface. For each system there's a games library beneath it and this is created by inserting a USB drive into your Pi. A script creates the necessary file structure on the drive. Insert this drive into your PC and copy the ROMs from your computer into their corresponding folder on the USB drive. Now return the stick to your Raspberry Pi and *RetroPie* will automatically copy the contents to your install. By copying ROMs in this way their corresponding emulator is enabled in the Emulation Station user interface.

***RetroPie* pulls in detailed information on each game in your library, including gameplay, history and box art.**



The *RetroPie* user interface is powered by Emulation Station and provides a slick and joypad-friendly manner of navigating your game library.

At this point, we need to advise you that ROMs are copyright material and remain the property of their owners. Their use in emulation is a grey area and many games are now so old that they are no longer commercially available, however, this doesn't mean that they are out of copyright.

With your ROMs installed go back to the user interface and select which system you would like to play. *RetroPie* will now ask if you would like to catalogue your games, before accepting, ensure that your Pi is connected via Ethernet to your router. *RetroPie* will search the internet for your games and download any box art and information about each title. Where there's a conflict it will ask you to choose which is correct, or if nothing is found it will ask you to manually enter the names of the games. *RetroPie* can emulate a number of systems, such as the Commodore Amiga, Sega Genesis and Super Nintendo, and on the Pi 2 there's no need to overclock the system as it runs six times faster than its predecessor. If you are using an original Raspberry Pi then you can streamline your setup using the Advanced Configuration steps on the *RetroPie* wiki (**http://bit.ly/RetroPieAdvConfig**).

For this project you will need

- Raspberry Pi 2 model B (for best results)
- Large capacity SD card
- USB joypad
- Peripherals to use with your Raspberry Pi 2, such as mouse and keyboard

The original Generation Code

There were many different games consoles and computers from the 1970s to early 2000s. Companies such as Commodore, Sinclair, Acorn, Dragon, Sega, Atari all competed for home computing dominance. This drove the rise of bedroom programmers; people who learnt to code from magazines of the time.

In each magazine there would be pages of BASIC code to type into your computer and learn coding via experience. This lead to a boom in the number of games being produced in the UK, and fuelled a generation of coders who are now hoping to reignite that spark with the Raspberry Pi. David Braben, for example, co-wrote the iconic *Elite* game in the 1980s. *Elite* was an entire galaxy full of planets, space stations and pirates in only 32Kb of memory. David Braben also happens to be one of the co-founders of the Raspberry Pi Foundation and is very keen to see children creating and learning with computers, rather than simply consuming content.

First steps with robotics

Start your journey towards Skynet. Build a keyboard-controlled robot using ScratchGPIO 7 and a Pibrella board.

For this project you will need

- A Raspberry Pi 2 model B, Zero, or a Raspberry Pi 1 model A or B
- Pibrella (Cyntech)
- Two Micro Gear motors (Pimoroni)
- Wheels (Pimoroni)
- Ball caster for balance (Pimoroni)
- Arts and craft materials

Optional

- USB battery
- Wi-Fi dongle

Building robots has become a 'rite of passage' for many Raspberry Pi owners, and thanks to initiatives, such as Scratch GPIO and Pibrella anyone can easily build a robot using easy to obtain components. This project will create a keyboard-controlled robot that we can control remotely using VNC.

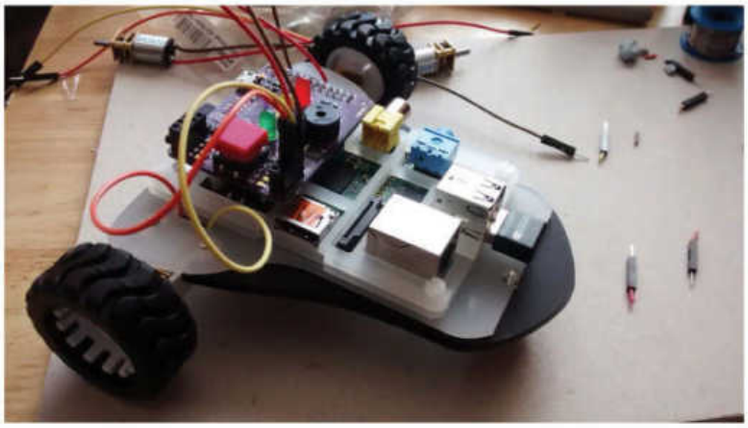
We shall start by connecting the Pibrella board to the Raspberry Pi; it fits over the first 26 pins of the GPIO and hovers slightly over the HDMI port – it might be a bit more fiddly if you're using a Zero. If there's contact between the Pibrella board and the HDMI port use a blob of modelling clay to prevent a short. Next, connect up your peripherals to the Raspberry Pi, except for the power supply which will now connect directly to the Pibrella board. Turn on your Raspberry Pi and head to the desktop.

To use Pibrella we need to install some software first for which you'll need to be connected to the internet, in *LXTerminal* type the following line by line:

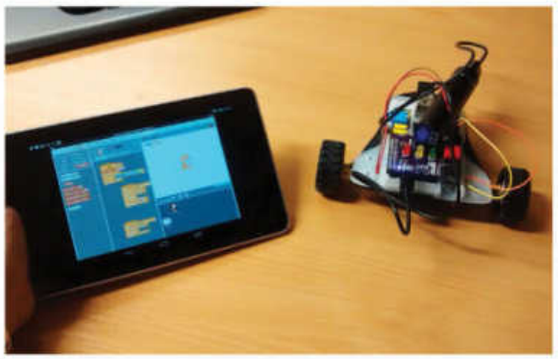
```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install python-pip
sudo pip install pibrella
```

Now we will install Scratch GPIO 7, which uses the familiar Scratch programming language interface to enable anyone to take their first steps with physical computing. In *LXTerminal* enter the following:

```
wget http://bit.ly/1wxrqdp -O isgh7.sh
sudo bash isgh7.sh
```



Our parts including the chassis were supplied by Pimoroni, but you can easily make your own using any arts and crafts materials.



You can also link this project to VNC and then control your robot from a tablet.

You should now see a Scratch GPIO 7 icon on the desktop. Launch the application and you will see the familiar Scratch interface. To use Pibrella with ScratchGPIO we must first create a variable called AddOn and then set the AddOn to Pibrella. Last, we create a Broadcast called AllOff.
The Broadcast block is in the Control Palette and will be blank, requiring your input. To sit on top of these blocks use the Green Flag Start hat block, which means that the Pibrella board will be reset everything to off once the flag is triggered. This will be an emergency stop for our robot. Now we need to create the code that will enable our robot to move when we press a key.

First, lets create a way to move forward when the Up arrow is pressed. In the Control Palette there's a hat block called When Space Key Pressed. Drag that into your code and change the dropdown to Up Arrow. Under that block we will now create two new broadcast blocks: OutputEOn and OutputFOn, and these will turn on our motors once connected. From the Control Palette connect the Wait block and set it for two seconds. This will send two seconds of power to the motors. Last, create two new broadcast blocks OutputEOff and OutputFOff these will stop the motors.

Now connect your motors to outputs E and F and when ready run the code and press the Up arrow. Your robot will now move forward, if it spins then swap the wires around until it moves forward. To turn your robot you will need to turn only one output on, we connected the left wheel to E and right to F. So to turn right we would turn output E on and our left wheel would spin and the right wheel would be a pivot. Have a play with timings and control, and make this project your own.

Picking robot kits

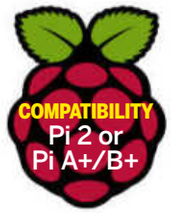
Robotics is a great project to undertake, but there are lots of kits out there that claim to be the best, so which one is best for you? Well, the one that meets your needs.

The Pibrella-based robot in this tutorial is extremely simple in that it has no sensors and no reverse gear, but hopefully it should help spark interest in learning more. The Pibrella board is remarkably versatile and you can easily adapt this project into a motorised dice game or use it to power a quiz game, such as the Wheel of Fortune game (**http://bit.ly/WheelOfFortunePi**).

Pibrella also lowers the barrier for entry so that even young children can learn some simple electronics, using ScratchGPIO and then move on to Python – and all via a £9 board. Other boards to consider are the RyanTeck Budget Robotics Kit (see **http://bit.ly/RyanTeckBudgetRobotKit**) this is a motor control board and robotics platform that can also be programmed with ScratchGPIO and Python.

Build a better alarm

Build a laser tripwire with the Explorer HAT Pro add-on board and catch a burglar... or just your cat.



The Explorer HAT Pro is the latest add-on board for the Raspberry Pi from Pimoroni and it packs in a lot of functionality. In this project we will build a laser tripwire that will raise the alarm when triggered. We'll start by first installing the two Python packages that will power our project. We'll use *pip* the Python packaging tool. To install *pip3* enter the following into *LXTerminal*:

```
sudo apt-get install python3-pip
```

Next, install PyGame and Explorer HAT libraries:

```
sudo pip3 install pygame
sudo pip3 install ExplorerHAT
```

With the libraries installed we move on to creating the code that will drive our project. You can grab a copy from the **LXFDVD** or **http://bit.ly/LXFExplorerHATAlarm**.

For this project, we used IDLE3 which we will need to run from *LXTerminal* using the command

```
sudo idle3 &
```

We need to do this as only root or sudo users can use the Explorer HAT Pro. With IDLE3 open create a new file, or open the code from the GitHub repository.

The first three lines import three libraries that power our project. We're building the project on the Explorer HAT board, so we import that and rename it to eh to make it easier to work with. Next, we import the sleep function from the time library and last, we import the pygame library.

```
import explorerhat as eh
from time import sleep
import pygame
```

The next line starts the pygame audio mixer, which we will need to play our alarm:

```
pygame.mixer.init()
```

Now we move on to the main loop that we will use to form the basis of detecting an intrusion. We start by creating an infinite loop and create a variable which stores the status of the analog pin:

```
while True:
    a = (eh.analog.four.read())
```

Next, we are going to use an if...else conditional statement to control how the alarm is triggered. We measured the light level in our room and found that it was around 2.6V on our analog pin. A torch shining on the sensor would be much higher than that, which meant we decided to set the LDR sensor so that anything above 3V is considered OK and would activate a green LED on the board:

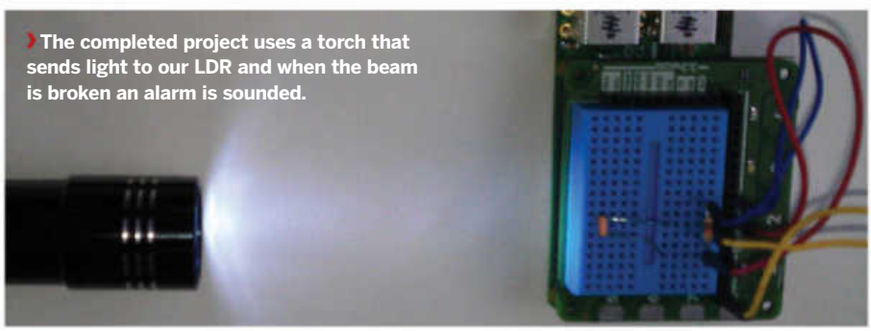
```
    if a > 3.0:
        print("SCANNING")
        eh.light.green.on()
        eh.light.red.off()
```

Our last section of code handles the alarm, which is triggered by the beam being blocked to the LDR. When this happens we turn the green LED off and turn on the red LED, and we trigger the playback of the alarm audio:

```
    else:
        print("ALERT")
        eh.light.green.off()
        eh.light.red.on()
        pygame.mixer.music.load("./alert.mp3")
        pygame.mixer.music.play(1)
```

We now move on to creating the hardware element. The Explorer HAT Pro comes with analog input, which the Raspberry Pi hasn't got. Our circuit is rather simple: we introduce 5V of power in via the LDR, which is then connected to a 10K resistor. The LDR and resistor form a voltage divider and we connect that to the analog pin of the Explorer HAT Pro. Last, we attach the other end of the resistor to Ground (GND). Now we just need to point a Laser pointer or torch directly at the LDR and that completes the hardware and software elements.

Make sure to double check your code and wiring before we continue. Start the project via the Run > Run Module menu. The IDLE shell will spring to life and report that it's scanning. Now go ahead and break the beam, the alarm will sound and you have caught your first intruder!



The completed project uses a torch that sends light to our LDR and when the beam is broken an alarm is sounded.

For this project you will need

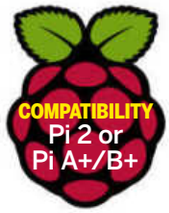
- A Raspberry Pi model 1 A+, B+ or Raspberry Pi 2 model B
- An ExplorerHAT Pro (Pimoroni)
- An LDR (Light Dependant Resistor)
- 10K resistor
- 3x Male to male jumper cables
- A torch or laser pointer
- A speaker or a monitor with audio capabilities
- Raspbian OS

Nice HAT, Harry

The Explorer HAT Pro is the latest HAT-based board from Pimoroni, a company based in Sheffield that produces many Raspberry Pi boards. The Explorer HAT Pro introduces new technologies to an all-in-one board.

It comes with eight capacitive touchpads that enable users to use touch-sensitive buttons on the board and four of these buttons can be linked to conductive objects, such as fruit using crocodile clips and then used to create novel forms of input (*see over*).

The Explorer HAT Pro also comes with a series of analog inputs that aren't available on the Raspberry Pi, and this makes projects using temperature sensors possible. The most challenging aspect of the Explorer HAT Pro is the inclusion of a motor controller with full support for forward and reverse gears, thanks to dual H-Bridge. Add to this a breadboard and a series of digital inputs and outputs, and we have a great platform for future projects and ideas.



Turn a banana into a guitar

If music be the food of love, why not make music with food? Build a banana guitar using Adafruit's Capacitive Touch HAT.

For this project you will need

- A Raspberry Pi 2 model B or a Raspberry Pi 1 model A+ or B+
- Adafruit Capacitive Touch 12x Sensor HAT
- Crocodile leads
- Bananas x7 (one to eat)

What if we told you that you could turn a banana into a guitar? Madness, but we're going to do just that with Adafruit's capacitive touch sensor HAT. But first we have to solder the 40-pin connector to the HAT's underside, so that it can slot onto the Pi GPIO.

Once soldered, attach the HAT, boot the Pi and open *LXTerminal*, where we'll start installing the required software:

```
sudo apt-get update
sudo apt-get install build-essential python-dev python-smbus python-pip git
cd ~
git clone https://github.com/adafruit/Adafruit_Python_MPR121.git
cd Adafruit_Python_MPR121
sudo python setup.py install
```

Now download the code we've adapted for this tutorial:

```
git clone https://github.com/lesp/LXF_BananaGuitar.git
```

Change the directory to **LXF_BananaGuitar** and in the terminal open the file **guitar.py** with **sudo idle guitar.py**. We need to import the necessary libraries, in this case sys, time and pygame (which adds multimedia, gaming and sprites to Python). Next, import the MPR121 library. This is the chip that makes the capacitive touch possible.

```
import sys
import time
import pygame
import Adafruit_MPR121.MPR121 as MPR121
```

Next, we initialise the MPR121 and include an error-handling system in case of issues:

```
cap = MPR121.MPR121()

# default I2C address (0x5A).  On BeagleBone Black will default to I2C bus 0.
if not cap.begin():
    print 'Error initializing MPR121.  Check your wiring!'
    sys.exit(1)
```

We then start Pygame's audio mixer and initialise Pygame:

```
pygame.mixer.pre_init(44100, -16, 12, 512)
pygame.init()
```

And create a library of sounds to use.

```
SOUND_MAPPING = {
  0: './1st_String_E.wav',
  1: './2nd_String_B_.wav',
  2: './3rd_String_G.wav',
  3: './4th_String_D.wav',
  4: './5th_String_A.wav',
  5: './6th_String_E.wav',
}
sounds = [0,0,0,0,0,0]
```

Now we set up how to play each sound in the library.

```
for key,soundfile in SOUND_MAPPING.iteritems():
      sounds[key] =  pygame.mixer.Sound(soundfile)
      sounds[key].set_volume(1);
```

Last, we create the structure that will constantly check to see if an input has been triggered:

```
last_touched = cap.touched()
while True:
    current_touched = cap.touched()
    for i in range(7):
        pin_bit = 1 << i
        if current_touched & pin_bit and not last_touched & pin_bit:
            print '{0} touched!'.format(i)
            if (sounds[i]):
                sounds[i].play()
        if not current_touched & pin_bit and last_touched & pin_bit:
            print '{0} released!'.format(i)
    last_touched = current_touched
    time.sleep(0.1)
```

Connect six crocodile clips from inputs 0 to 5 and attach the clips to the bananas. When ready run the code using Run > Run Module and wait a few seconds before pressing the fruit. You should hear sound from the TV or connected headphones. Congratulations, you've made a banana guitar!

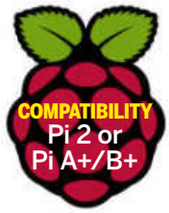
Connect the bananas to the board using crocodile clips. Make sure that you attach them to the top most part of the banana so you can eat them later.

Touchy projects

Capacitive touch works by using conductive materials, such as fruit, tin foil and even water. The Adafruit Capacitive Touch HAT comes with a basic Python library that can be easily integrated into your projects. Have you ever wanted to make piano stairs? Where each step is an ascending or descending note? Well, now you can by using long crocodile leads and cardboard wrapped in tin foil to act as the conductor. Also you can use the HAT as an input for a photo booth, where the banana or other conductive materials can be used to trigger a camera to take your picture.

This board is still relatively new on the market and at the time of writing there are only a few resources provided with it, but the sheer brilliance of this board means that there will soon be lots of projects. The banana guitar project (*above*) can also be undertaken with Pimoroni's Explorer HAT Pro board as it also uses capacitive touch input, and can work with four connected devices and four built-in buttons.

Raspberry Pi-powered disco



Get ready to strut your stuff on the dance floor. Create disco lights with the Unicorn HAT 8x8 RGB LED Matrix.

The Unicorn HAT is an add-on board for the Raspberry Pi Models B+, A+ and Raspberry Pi 2 that's an 8x8 grid of neopixel super-bright LEDs. The same type of LEDs are used to illuminate Blackpool Tower because of their low-power usage yet high brightness. We'll create two scripts to control your grid.

To install the Unicorn HAT on your Raspberry Pi, turn your Pi off and gently push the Unicorn HAT on to all of the GPIO pins until it sittings firmly. Once done, plug in your peripherals, ensuring that you have an internet connection for your Pi, and then boot your Pi to the desktop.

Warning: The NeoPixel LEDs are extremely bright, protect your eyesight by using a piece of paper to diffuse the light. Now we need to install the software, for this we need to open an *LXTerminal* and type:

```
sudo apt-get install python3-pip python3-dev
sudo pip-3.2 install unicornhat
```

Keep the *LXTerminal* open and type **sudo idle3 &** to open the IDLE3 Python 3 editor with sudo powers, so that we can use the GPIO. With IDLE3 open we're immediately presented with the Python shell. This is where we can issue commands directly to Python. Go to the File menu and click on New to open a blank document. For our project we will create a changing colour sequence that will switch all of the LEDs from red to green and then to blue in an infinite loop. First, let's import the libraries that we need:

```
from time import sleep
import unicornhat as u
```

You can see that we've imported the sleep function only from the time library as this saves system resources. On the next line we import the Unicorn HAT library, and rename it u to make it easier to work with. Now we move to the main body of code and we contain this in a **try-except** structure:

```
try:
    while True:
        for i in range(8):
            for j in range(8):
```

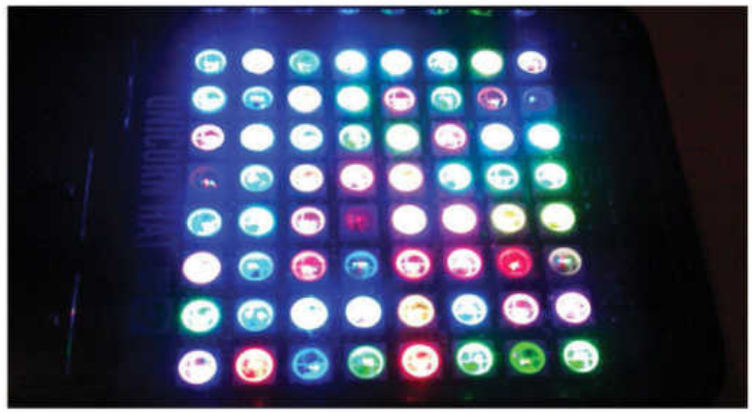
Inside the try structure we use **while True** to create an infinite loop, and inside this loop we have two for loops: **i** and **j**. These contain a value from 0 to 7 that will control our x and y positions respectively in the LED grid. Now we create the actions that will happen while the for loop is working.

```
                    u.brightness(1.0)
                    u.set_pixel(i,j,255,0,0)
                    u.show()
                    sleep(0.01)
```

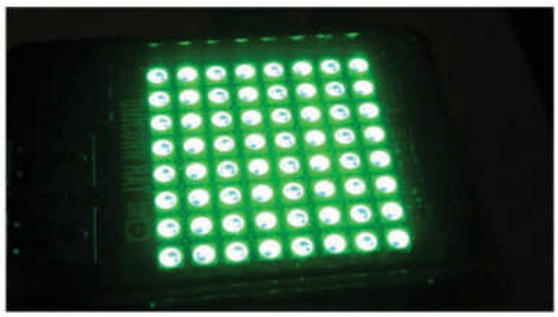
First, we set the brightness to full, which is 1.0 and set the pixel we are on to full brightness red using 255. The colours are handled by mixing red, green and blue as follows: 255,0,0 is red, 0,255,0 is green and 0,0,255 is blue. Next, we instruct the HAT to show the changes made and then wait for 0.01 seconds: We then halt the code for two seconds before moving to the next for loop to handle green and then blue.

The last section of code handles the user pressing Ctrl+C to break the loop, which stops the code and clears the HAT. Save your code and run using the Run > Run Module menu item. You will now see colours cycling on the screen.

For the full code, and an extension project, download the code examples from the GitHub (**http://bit.ly/LXFUnicornHAT**).



Our extension project creates a multicolour disco light using Unicorn HAT by using random numbers to generate values in our code.



Our first project creates a grid of one colour and then quickly changes it via an infinite loop.

For this project you will need

- A Raspberry Pi model 1 A+ B+ or a Raspberry Pi 2 B
- Unicorn HAT (Pimoroni)

NeoPixel powered

The Unicorn HAT was the first HAT board designed by Pimoroni for the Raspberry Pi model B+. The HAT (Hardware Attached on Top) standard uses an EEPROM (Electrically Erasable Programmable Read Only Memory) to communicate with the Pi and automatically configures the board ready for use. We recently had the chance to talk to Pimoroni about this board, and they revealed that the board was developed to enable them to learn more about the HAT standard. The pair also learnt about managing the power requirements for 64 LED, as, initially, the Unicorn HAT drew so much power that it reset the Raspberry Pi.

The LEDs on the Unicorn HAT are known as NeoPixels, or rather this is the name Adafruit uses, but their correct name is WS2812 Integrated Light Source and they are found on devices as small as name badges. Recently, hundreds of thousands of NeoPixels were integrated into Blackpool's famous Illuminations.

HACK IT!

Les Pounder takes us to a world where fridges can tweet, gardens can water themselves and something has an eye on your postman...

Our homes are being changed by technology and changing how we live. Many homes now have smart televisions, central heating systems that know when you arrive home and how warm you like each room in your house and even fridges that tweet reminders to buy more milk. But how easy is it to create your own piece of 21st century technology?

Home automation is becoming a mainstream project for many hobby hackers, largely thanks to the rise of the Raspberry Pi, and in particular the release of the Pi 2, which is a powerful and cost-effective platform. The Pi's GPIO (General Purpose Input Output) can interface with many common electronic components, such as sensors, relays and transistors. All of which can be programmed using Python and other languages. The Pi also comes with an Ethernet connection which supplies a stable connection to the outside world and enables remote control of projects. We also have Raspbian Linux; a stable and secure operating system that has a growing support base. Home automation encompasses many areas, eg environmental control, safety and security.

> **"Dip a toe into the possibilities with a series of projects that make the most of the Pi."**

But how easy is it to get started with automating parts of your home? Can anyone build home automation projects using the Raspberry Pi? Well, in this feature we'll dip a toe into the pool of possibilities with a series of projects that make the most of the Pi and use a series of off-the-shelf components mixed with a little Python code and data that's freely available from external sources.

You'll learn a lot of things if you work through each one: how sensors can be used to detect movement and trigger lights to turn on; how you can hack the humble doorbell so it has SMS functionality, add a sensor to detect when we have post and send a picture to us via email, so we never miss a package again. All of this is made possible thanks to a £30 computer, a few sensors and a little bit of Python magic.

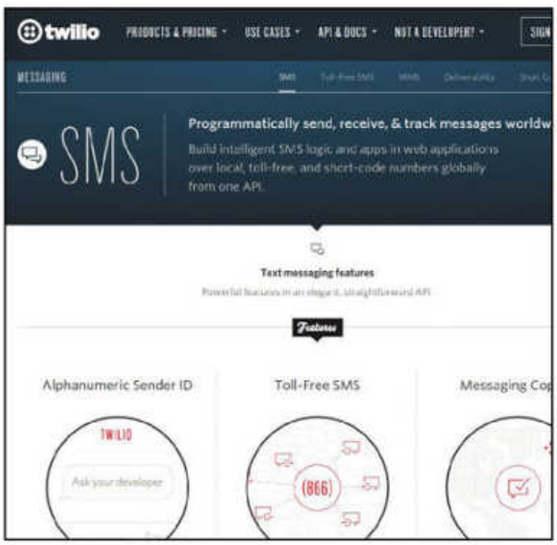
Texting doorbell

Get an SMS alert whenever a visitor drops by.

We purchased a doorbell unit for under £5 and used that as the basis of this wireless project.

The humble doorbell is great for alerting us to visitors as long as we're in earshot, but we could fix that with a little Internet of Things (IoT) knowhow. For this project, we've used a cheap wireless doorbell (found on Amazon for a fiver). We took apart the push button unit and found a circuit which uses a simple momentary switch powered by a 12V battery. The Raspberry Pi GPIO can't directly work with voltages over 5V so we first need to change the power supply for something lower.

You'll need to solder two wires onto the battery contacts for the push button unit. When pressed, the momentary switch connects the power to ground and effectively drops the current, changing the state of the unit from on to off and creating a trigger. Using a multimeter, locate the correct pins for your unit and solder wires to them. For added strength use a hot-glue gun to keep the wires on the contacts. Attach the positive battery terminal to the 3V3 GPIO pin and the GND of the battery terminal to the GND of your Raspberry Pi. On your momentary switch attach the button to pin 17 (Broadcom pin reference) and the other to the 3V3 GPIO pin.



Twilio is our bridge between the doorbell and SMS. It's an online SMS service that we can use via a Python library.

You will need to create a trial account (**https://www.twilio.com**) in order to send an SMS. Boot your Raspberry and navigate to the terminal and type the following to install the Twilio API for Python: `$ sudo w pip3 install twilio` . Open the Python 3 application via the Programming menu, create a new file and immediately save it as **Doorbell-SMS.py**. We start our project by importing the Twilio API, the `time` library and the `GPIO` library:

```
from twilio.rest import TwilioRestClient
import time
import RPi.GPIO as GPIO
```

Afterwards, we need to configure our GPIO to use the Broadcom pin-mapping, set up pin 17 as an input and set its built-in resistor to pull the current down:

```
GPIO.setmode(GPIO.BCM)
GPIO.setup(17, GPIO.IN, GPIO.PUD_DOWN)
```

Next, we create a function that will handle sending a text message using the Twilio API. You will need to replace the account and token details with that of your own and change the `to=` and `from_=` telephone numbers to correspond with our requirements:

```
def sendsms():
  ACCOUNT_SID = "ACCOUNT ID"
  AUTH_TOKEN = "AUTH TOKEN"
  client = TwilioRestClient(ACCOUNT_SID, AUTH_TOKEN)
  message = client.messages.create(
     body="Doorbell has been rung",
  to="NUMBER TO SMS",
  from_="YOUR TWILIO PHONE NUMBER",
)
  print(message.sid)
  time.sleep(5)
```

Our last section of code is a loop that will constantly go round. We look for the current on pin 17 to drop in the loop and when it does the function is called triggering an SMS being sent to your mobile:

```
while True:
  GPIO.wait_for_edge(17, GPIO.FALLING)
  sendsms()
```

Save your code and click on Run > Run Module to test.

You will need...

- Any Raspberry Pi but A+ is best.
- A wireless doorbell
- Soldering skills
- Twilio account
- The latest Raspbian OS
- All of the code can be found at **https://github.com/lesp/LXF-PiHome-SMS-Doorbell**.

External services

Working with external data sources and services is an exciting area to explore with your Raspberry Pi. There are many different sources, such as weather, astronomical and mobile communications data.

Data sources can be used as a method of input to trigger an event in the physical world, eg such as turning on a fan based on the current temperature or a data source can be used as an output, eg such as an air pressure changes log.

In this project we used the Twilio service to access SMS functionality through a Python API. Twilio is a cheap and robust service for projects and after the free trial ends it's pretty cheap to use at $1 charge per month and around $0.04 per SMS. Using Twilio we can go further and turn our simple IoD (Internet of Doorbells) into a truly powerful device with MMS (Multimedia Messages), which contain video and pictures captured by the Raspberry Pi Camera.

There are other SMS providers, one being **www.smspi.co.uk**, which itself uses a Pi to handle sending and receiving SMS messages and comes with 2,000 free SMS.

Entry lights

Welcome lights when you open your door.

You will need...

- Any Pi Zero, A+, B+ or Pi 2
- The latest Raspbian OS
- Energenie power sockets and Pi Remote https://energenie4u.co.uk
- A reed switch
- Jumper wires
- Magnets
- All of the code can be found at **https://github.com/lesp/LXF-PiHome-EntryLight**

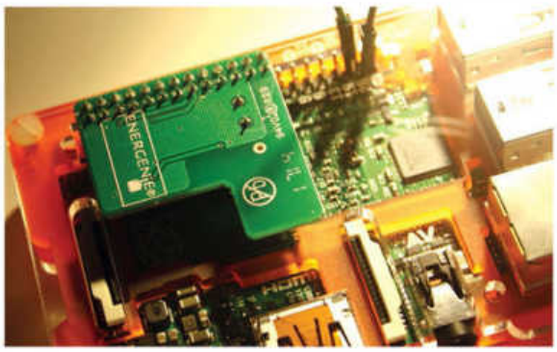
Returning home to a dark house in the winter is depressing so let's use a few off-the-shelf components to build a bright welcome home project. First, we need to attach the Energenie to the first 26 pins of the GPIO on your powered-down Pi. (For reference, pin 1 is the pin nearest the SD card slot.) The board will fit neatly over the Pi with no parts hanging over.

Now attach a female-to-female jumper cable to GPIO20 and GND through the unused GPIO pins. (If you want to extend the jumper cables simply use male-to-female cables until the desired length is reached.) On one end of the female jumper cable attach the reed switch and then the other. Using sticky backed plastic attach the switch to a door frame and attach magnets level to the switch but on the door itself so that the switch is closed when the door is closed.

Boot your Pi and open a terminal. To install the Energenie library for Python 3 use `$ sudo pip-3.2 install energenie`. Once installed open a new Python 3 session via the Programming menu. To pair our Energenie units with our Pi open the IDLE shell and type `from energenie import switch_on, switch_off`. Now plug in your Energenie and press the Green button for six seconds. This forces it to look for a new transmitter. Back in your IDLE shell, type `switch_on(1)`. This will pair your Pi to the unit and designate it '1' and the process can be repeated for four units. With IDLE open click on File > New Window and save your work as **entrylight.py**.

We'll start by importing the libraries for this project:

```
from energenie import switch_on, switch_off
import time
```



The unit from Energenie fits neatly over the first 26 pins of the Pi 2 or over all the GPIO pins of an older Raspberry Pi.

The receiver in the Energenie unit houses a relay to switch the mains power on and off.

```
import RPi.GPIO as GPIO
```

The `energenie` library controls the units for our lights, and `time` is used to control how long the units are powered for and `RPi.GPIO` is the library used for working with the GPIO.

```
GPIO.setmode(GPIO.BCM)
GPIO.setup(20, GPIO.IN, GPIO.PUD_UP)
switch_off()
```

Next, we set the GPIO to use the Broadcom pin mapping and set GPIO20 to be an input with its internal resistor pulled high, turning the current on to that pin. Finally, we turn off the Energenie units to make sure they are ready. The main code uses a try...except structure to wrap around an infinite loop:

```
try:
  while True:
        if GPIO.input(20) == 1:
        switch_on()
        time.sleep(30)
        switch_off()
```

Inside the loop we use a conditional statement to check if the input has been triggered, ie the door has been opened. If true then the units are switched on for 30 seconds and turned off again.

```
      else:
         switch_off()
except KeyboardInterrupt:
   print("EXIT")
switch_off()
```

We finish the conditional statement with an `else` condition. This will turn the units off and loop continually. We close the try...except structure with a method to close the project, pressing CTRL+c will end the project and switch off the units should the need arise. With the code complete. Save your work and click on Run > Run Module to test the code.

Energenie

Controlling high voltage devices is a project for those that know their stuff but with Energenie we can significantly reduce the risk.

Energenie units at their core are simply 433MHz receivers that control a relay; a component that uses a low voltage to control a magnetic switch in a high voltage circuit. On the Raspberry Pi we have a transmitter which can instruct the receivers to turn on and off.

Energenie units are a safe way to control mains electricity. The standard Python library for Energenie is rather cumbersome, requiring the user to control the GPIO pins used by the transmitter in order to connect to each device and issue the correct instruction. This library has been made a lot simpler thanks to Ben Nuttall, a member of the Raspberry Pi Foundation's Education team, and Amy Mather, known to many as Mini Girl Geek a teenage hacker and maker. This improved library, which we've used in this tutorial, requires that we know the number of each unit and can issue an instruction to one or all units at once.

The library can be found on GitHub, should you wish to inspect the code and learn more about how it works. See **https://github.com/RPi-Distro/python-energenie**.

Postie watch

Email a snap of your special deliveries.

You can house this project in any type of case, requiring only a clean line of sight to the letterbox.

Are you always backing kickstarters but never at home to receive your rewards when the postman comes? Well, this project can alert you to a parcel via email. With your Raspberry Pi turned off, attach the camera by the camera slot located near the Ethernet port. Next, connect your Passive Infra-Red (PIR) sensor to the following GPIO pins of your Pi. Please note: we are using the Broadcom pin mapping.

PIR PIN	GPIO PIN
VCC	5V
OUT	17
GND	GND

Boot your Pi and use the configuration tool located in the Preferences menu. Enable your camera and ensure that SSH login is enabled. Reboot and then open Python 3 from the Programming menu. Create a new file, save and call it **emailer.py**. We start our code by importing a series of libraries. (You can view the full list via our source code link, and it starts with `from mail_settings import *` .) These handle sending email, taking pictures using the camera and timing our project. One extra library is mail_settings. This is an external library written just for this project and used to store email usernames and passwords. We'll be using the Broadcom pin mapping and need to set this before we proceed:

```
GPIO.setmode(GPIO.BCM)
global file
PIR = 17
GPIO.setup(PIR, GPIO.IN)
```

We now create two variables: the first is a global variable that we can use between functions and the second, called PIR, stores the number of the pin used for our sensor. We set up our PIR connected to GPIO17 as an input. Next, we create two functions: the first takes a picture with the camera:

```
def takepic():
  global file
  current_time = str(datetime.datetime.now())
  current_time = current_time[0:19]
  with PiCamera() as camera:
    camera.resolution = (800, 600)
    camera.framerate = 24
    camera.capture((current_time)+'.jpg')
    takepic.file = ((current_time)+'.jpg')
```

With `takepic()` we capture the current time and date as the name of the file, and we store and slice the string stored in the variable using only the text that we need. Next, we capture an image and save it with that file name. Our second function handles the email:

```
def email_send(to,file):
  current_time = str(datetime.datetime.now())
  current_time = current_time[0:19]
  msg = MIMEMultipart()
  msg['Subject'] = 'ALERT - AT '+current_time+' THE POST HAS ARRIVED'
  msg['From'] = email
  msg['To'] = to
  with open(takepic.file, 'rb') as pic:
    pic = MIMEImage(pic.read())
  msg.attach(pic)
  server = smtplib.SMTP('smtp.gmail.com',587)
  server.starttls()
  server.login(email,password)
  server.ehlo()
  server.send_message(msg)
server.quit()
```

We can reuse the same method to capture the date and time of the alert. We create a mixed content email with a subject that's composed of an alert string featuring the time and date of alert. Who the email is from is generated by the customer mail_settings library. The recipient is passed as an argument in the function and our image is attached as a file to the email. A variable called `server` stores the location of our mail server, in this case a Gmail account. We open a secure connection to the server, login and then announce to the server that we're there. We then send the message before closing the connection to the server.

With the functions written, we use a while true loop to constantly check if the PIR sensor has been triggered. If this is the case a photo is taken, attached to an email and sent to the recipient. If the sensor isn't triggered then the loop repeats. Now save your work and click on Run > Run Module to start. »

You will need...

- Any Pi Zero, A+, B+ or Pi 2
- PIR Sensor
- Raspberry Pi Camera
- Wi-Fi dongle
- The latest version of Raspbian
- A Gmail account
- All of the code can be found at https://github.com/lesp/LXF-PiHome-PostWatch

Sensors

Sensors are an exciting method of automatically generating input which can be used to trigger events based on movement, sound and light etc. The Raspberry Pi can be connected to many different types of sensor.

In this project we use a simple, passive infra-red sensor to detect movement. It operates by sending a current to the Pi when triggered.

Another type of sensor that could be used is an ultrasonic sensor, which sends a pulse of ultrasonic sound to determine the distance of an object from the sensor. This is a great sensor but it requires a little mathematics in order for it to work. Both the PIR and ultrasonic sensor can be picked up for less than £3 on eBay. The Pi can only directly use digital sensors as the GPIO doesn't support analogue sensors, but for under £10 you can pick up an analogue to digital converter (ADC) which will bridge the gap.

PIR sensors can add a new form of input quickly and easily thanks to their simple operation.

Home heating monitor

Visualise your central heating.

You will need...

- Any Pi Zero, A+, B+ or Pi 2
- The latest Raspbian OS
- A DS18B20 sensor (part of the Cam Jam EduKit 2)
- Breadboard
- Male to Female jumper cables
- 4.7kOhm resistor
- WI-FI dongle
- An www.initialstate.com account
- All of the code can be found at https://github.com/lesp/LXF-PiHome-InitialState

For this project, we'll dunk our heads into the Internet of Things (IoT). We'll determine the temperature of our home using a cost-effective sensor and push that data to the cloud and use it to populate a graph. The sensor we're using is a Dallas DS18B20. These can be picked up relatively cheaply, but an easy solution is to buy the CamJam EduKit 2 as it includes a waterproof Dallas DS18B20. Assemble the hardware and attach to your Pi as per the diagram (*see right*). Next, we set up the sensor and there's a handy Cam Jam worksheet (http://bit.ly/CamJamTempWorksheet) for this. To proceed you'll need an www.initialstate.com account and your API key, which you'll find in your account settings. To install the Initial State streamer type:

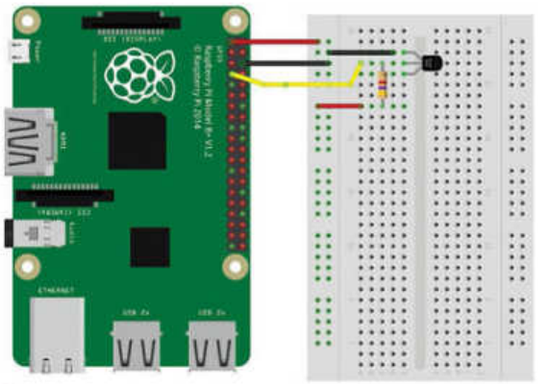
```
\curl -sSL https://get.initialstate.com/python -o - | sudo bash
```

We start our code by importing libraries to work with the OS, time and to stream our data to the cloud:

```
import os, glob, time
from ISStreamer.Streamer import Streamer
```

Next, we load the kernel modules for the sensor using `modprobe`, we wrap the *Bash* commands in an `os.system()` function for Python and tell our code where to find the file for storing the temperature data:

```
os.system('modprobe w1-gpio')
os.system('modprobe w1-therm')
base_dir = '/sys/bus/w1/devices/'
device_folder = glob.glob(base_dir + '28*')[0]
device_file = device_folder + '/w1_slave'
```



We attach the DS18B20 to a breadboard and supply power to its data pin via a 4.7kOhm resistor.

Next, we create a function to handle reading the contents of the file which stores the raw temperature data and stores the data as a variable.

```
def read_temp_raw():
  f = open(device_file, 'r')
  lines = f.readlines()
  f.close()
  return lines
```

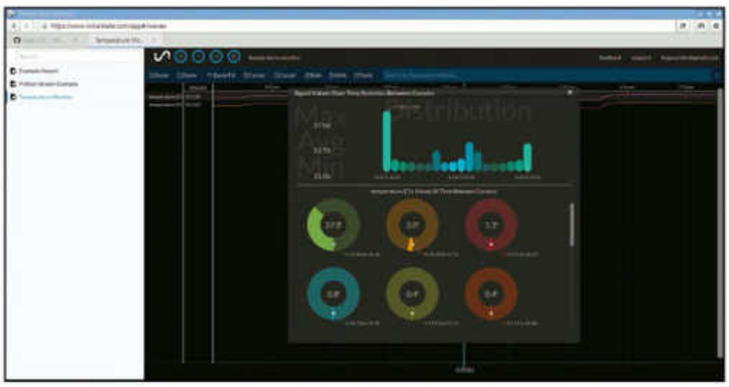
Now we read the data and process it into something more usable. We keep the information and strip the rest of the data before converting the data to a temperature.

```
def read_temp():
  lines = read_temp_raw()
  while lines[0].strip()[-3:] != 'YES':
    time.sleep(0.2)
    lines = read_temp_raw()
  equals_pos = lines[1].find('t=')
  if equals_pos != -1:
    temp_string = lines[1][equals_pos+2:]
    temp_c = float(temp_string) / 1000.0
  return temp_c
```

Our last section is a loop that constantly checks the temperature, performs conversions and streams the data to Initial State every minute.

```
while True:
temp_c = read_temp()
  temp_f = temp_c * 9.0 /5.0 + 32.0
  streamer.log('temperature (C)', temp_c)
  streamer.log('temperature (F)', temp_f)
  time.sleep(60)
```

Save the code and click on Run > Run Module to start.



Initial State can cope with multiple data inputs.

Initial state

In this project we sent temperature data to the cloud using a service called Initial State. This service enables users to graph and manipulate data from multiple sources at once. We used the free tier in this tutorial, which retains our data for 24 hours before deleting it. There are other tiers which can retain data indefinitely for an unlimited number of sensors.

For our project we used one sensor input, a DS18B20, but thanks to the Raspberry Pi and its GPIO we can use many more sensors to gather data about our home, eg in another tutorial we used a reed switch. This too can be used with Initial State so that we can show data when doors are opened. So using this service we can interpret data about our home. Such things as reed switches on windows; temperature sensors in rooms; a clamp on our electric meter and light sensors outside can be used to provide data on how energy efficient our home is and this data can be graphed for many months to show our usage over the seasons. This data can be used with a central heating system to control your home automatically using a humble Pi.

Remote CCTV

Keep tabs on your possessions or pets.

For this project we'll create a remote monitor for tracking activity in a home. Before we begin, make sure that your webcam is plugged into your Pi. To update our system and install the webcam motion software, you'll need to open *XTerminal* and type:

```
$ sudo apt-get update && sudo apt-get install motion
```

With motion installed let's configure it with:

```
$ sudo nano /etc/default/motion
```

You'll see `start_motion_daemon=no` change this to yes. Now press Ctrl+o to save and Ctrl+x to quit. Now we need to make a few changes to our **motion.conf** file. Open it with `$ sudo nano /etc/motion/motion.conf` . Ensure the following is correct before saving (Ctrl+o) and exiting (CtrlL+x) *nano*.

```
daemon on
width 640
height 480
framerate 100
stream_localhost off
```

Reboot your Raspberry Pi before continuing. Now let's test our stream. In a terminal type `$ sudo service motion start` .

Now in a browser on another machine type in the IP address of your Raspberry Pi, you can find this in the terminal by typing `hostname -I` followed by `:8081` so for example my IP address was **192.168.0.3:8081**.

You should now see a video stream in your browser. Now that we have the stream working let's embed it into a live web page. To do this we will need to install *Apache*. In a terminal type `$ sudo apt-get install apache2 -y` . This will also create a new directory in **/var/** called **/www/** which we shall use to serve our pages.

Open the text editor on your Pi. We will now write a few lines of HTML to build a simple web page.

```
<!DOCTYPE html>
<html>
<title>Puppy/Baby Monitor</title>
<body>
<xmp theme="cyborg" style="display:none;">
## I wonder what the dog/baby is up to?
<img style="-webkit-user-select: none"
src="http://192.168.0.3:8081/">
</xmp>
</body>
```

We can easily embed our stream into a webpage.

You will need...

- Any Raspberry but best with Pi 2
- The latest Raspbian OS
- An internet connection
- A compatible webcam

```
<script src="http://strapdownjs.com/v/0.2/strapdown.js"></
script>
</html>
```

We start by declaring the document as a valid HTML document and give the page a title to identify it in our browser. Now we move to the `<body>` where we use a framework called strapdown, which mixes markdown – a popular writing format – with Twitter's bootstrap framework. In essence we can make a nice page rather quickly. We're using the cyborg style as it's dark and looks great on devices. To create a headline we use two hashes (#) and then type the contents of the headline. Next, we add an image whose source is the IP address of the webcam stream. To make sure the IP address matches that of your Pi we add **:8081** at the end. We then instruct the browser to load a JavaScript file containing the strapdown functionality. Save your file as **index.html** to your **home** directory. Open a terminal and type the following to copy the file to our web server:

```
$ sudo cp /home/pi/index.html /var/www/html/
```

Finally, we need to start our web server and restart the motion service.

```
$ sudo service apache2 start
$ sudo service motion restart
```

Now visit your Raspberry Pi's IP address – you no longer need to add :8081 to the end of the IP) – and you will now see a video stream from your Pi.

CCTV

The Raspberry Pi has made many different types of projects possible and one that's popular is CCTV. The official Pi Camera, along with the Pi offer a low cost, high quality and low-power project you can build quickly. In this project, we used *motion* to stream our webcam to a webpage, but *motion* can be used to search for motion and stream as well, eg we can record a video stream to a local or cloud device which will be triggered by a burglar, baby or Jack Russell terrier. Add a Passive Infra Red (PIR) sensor to this code, such as the one used in our delivery watch project, and you have a powerful application that can alert you to incidents and record the evidence. Another great application to use with a webcam is *Zoneminder* (**www.zoneminder.com**) which also works with the Raspberry Pi. Using *Zoneminder*, you'll be able to monitor multiple streams and set up zones which will trigger an alert, eg a zone drawn around a door frame would trigger if a person used the door, but the surrounding area wouldn't be monitored for activity.

This project streams video over a network connection.

Automatic plant waterer

Water your plants when the weather forecast suggests no rain.

You will need...

- Any Pi Zero, A+, B+ or Pi 2
- The latest Raspbian OS
- Piface Relay Plus
- A 12v Peristatic pump
- A 12v 1A power supply
- Barrel jack to screw terminal
- Aquarium airline
- Soldering skills
- Wi-Fi dongle
- An open weathermap.org account
- All of the code can be found at https://github.com/lesp/LXF-PiHome-GardenManager

Having to dig out the ol' watering can and potter about the garden might be some people's idea of bliss, but it's not very 21st century. Besides think of the time you can recoup for another hacking project. In our final Raspberry Pi project in this feature, we're going to automate the whole watering of plants with a Pi linked to a weather forecast service and an add-on board that is connected to a pump.

To kick off, we start by soldering connections to the terminals of our pump. These can be secured with a hot-glue gun or heat shrink. You'll need to use more wire on the barrel jack screw terminals and make a note of which is plus (+) and minus (-). On the Piface Relay Plus, locate relay 3 and insert the GND (-) of your power into the COM terminal and also one of the pump connections. Locate the NO (Normally Open) terminal and insert both of the remaining wires. Next, you'll need to attach the Piface Relay Plus board to your Pi and boot to the desktop. To install the software for your Piface board and use openweathermap with Python 3, open *XTerminal* and type:

```
$ sudo apt-get update && sudo apt-get install python3-pifacerelayplus
$ sudo pip-3.2 install pyowm
```

Open Python 3 IDLE via the programming menu and create a new file. Save your project as **garden_manager.py**. We start the code by importing the Piface, pyowm and time libraries with `import pifacerelayplus, time, pyowm`. Next, we create a variable called key and store our API key from **http://openweathermap.org**.

We now need to create two functions: our first function controls the pump attached to Piface. This function we're calling pump and it takes one argument: how long it should water the garden.

```
def pump(time):
  pfr = pifacerelayplus.PiFaceRelayPlus(pifacerelayplus.RELAY)
  pfr.relays[6].toggle()
  time.sleep(time)
  pfr.relays[6].toggle()
```

We use a variable, `pfr` to shorten the function call for using a relay. Then we toggle the relay on and off, depending on its current state. We then pause using `time.sleep()`, permitting the water to flow, before we toggle the relay off.

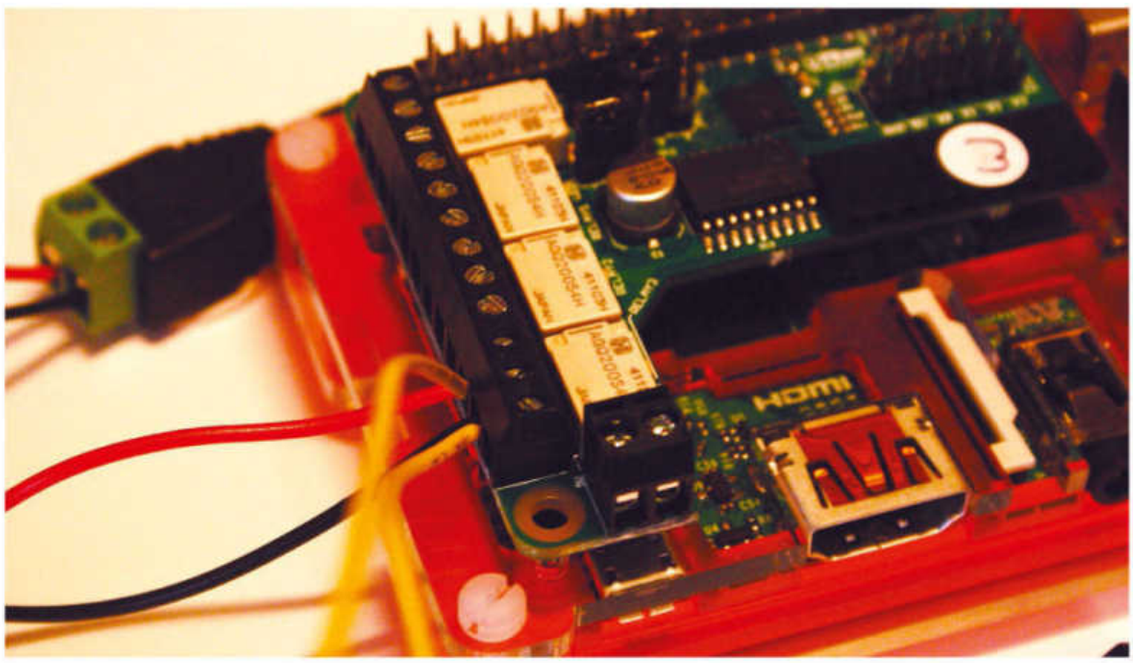
Our second function retrieves the weather forecast for the next 24 hours. It takes two arguments: our location and the number of days to forecast. We then create a variable to store our openweathermap API key, and a further two variables contain the output of the forecast functions.

```
def forecast(x,y):
    owm = pyowm.OWM(key)
    fc = owm.daily_forecast(x,y)
   f = fc.get_forecast()
```

We use a for loop to iterate over the forecast data. This feature comes into its own when used to forecast weather for multiple days:

```
    for weather in f:
        rain_forecast = str(weather.get_status())
```

Finally, the function uses an if...else statement to check the forecast. If rain isn't forecast this information is printed to the shell before calling the pump() function. If rain is forecast,



The Piface Relay Plus Board offers a number of relays that can be directly controlled via Python.

Working with high voltages

In this project we used a 12V power supply to power a pump, but you may be asking why we had to use a relay? The Pi can't tolerate voltages over 5V and to use anything above that would risk damaging the GPIO or the Pi itself. A relay is a magnetic switch triggered by a circuit connected to the Raspberry Pi. This circuit is 5V tolerant and when activated it enables a magnet which pulls a switch inside the relay closed. There's no connection between the Pi and high voltage circuit, which means we can safely control high voltages.

We used the Piface Relay Plus board which comes with four relays attached. Alternatively, you could use a relay on a breadboard, but for safety reasons we would suggest only using a maximum voltage of 12V on the board, as anything more would require a more robust solution.

Relays are not the only solution, a transistor can also be used to control higher voltages. Transistors work in a similar way to a relay, in that they isolate the high voltage circuit but are controlled by a low-power circuit. Both relays and transistors are low-cost methods of controlling high voltage projects. Remember if you are unsure about a circuit, ask someone who does before applying power!

this information is printed to the shell before waiting 24 hours before checking again.

```
    if rain_forecast != "rain":
        print('Rain is not forecast')
        pump(300)
        time.sleep(86400)
    else:
        print('Rain is forecast')
        time.sleep(86400)
```

Finally, we need to create a loop that will call the `forecast()` function for Blackpool for the next 24 hours. Of course, you can change the location to where ever you live.

```
while True:
    forecast('Blackpool,uk',1)
```

As usual, you will want to save your code at this point and click on Run > Run Module to test. For testing it would be prudent to reduce the `time.sleep()` duration to something much shorter.

Taking the Pi further

A world of home automation awaits.

Home automation covers an extensive range of products, services and new concepts. If you've been bitten by the IoT bug and wish to expand your knowledge and dive into a home automation set up then here are a few areas for further research.

Nest

Purchased by Google in early 2014 for $3.2billion, Nest is a big player in home automation. Its range of products started with a central heating control system that linked to mobile devices. Now its range covers smoke and carbon monoxide detectors and an IP camera. The main issue with these devices are their rather high cost, eg a simple smoke/carbon dioxide detector retails for £89. What you pay this high price for is convenience and all of the hacking has been done for you packaged in a sleek device.

Nest has added a lot of style to central heating.

X10

This is a protocol for electronic devices that primarily use powerlines to send control and signalling. X10 has been around since 1975 and while it may not be the latest protocol it does have a large user base, thanks largely to a low cost of components. X10 also enjoys support for a range of boards, including the Arduino and Pi, which enables it be used to control appliances in your home in different ways.

Wireless Things Open Pi

The Open Pi project uses the lesser-known member of the Pi family, Compute. This is model is a smaller SODIMM package that's ready for embedding in a project and Open Pi places this, already slim module into a small plastic case. The Open Pi is designed to enable hackers to use it for various IoT projects, using a mix of Bluetooth Low Energy, an infra red receiver and an SRF shield, which offers long distance radio communications to devices equipped with an SRF.

Devices that are available include: an Arduino-compatible board called Xino RF; an SRF GPIO add-on for the Pi and a USB stick for computers. The SRF and its higher-powered version ARF can transmit over distances far greater than standard wireless IoT devices.

Bluetooth LE

Bluetooth has been with us for many years but recently we've seen a new low energy version that offers a low-power short distance connection called Bluetooth LE. These have been made into beacons, like Estimote (**http://estimote.com**) that can be programmed to react to Bluetooth devices in inventive (or intrusive, depending on your viewpoint) ways, eg they will broadcast an open Bluetooth connection which can push data to your device. These can be used in a house setting to recognise when a user returns home and can interact with appliances via X10 to set up your home ready to relax. These beacons can be built using a Raspberry Pi and a Bluetooth LE dongle, enabling a low-cost and non-proprietary solution of your own.

Kore and Yatse

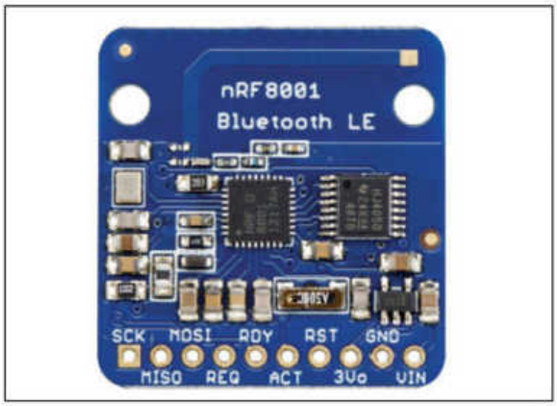
One of the most common Pi projects is a media centre, especially since the release of the Pi 2 in early 2015. Instead of using a wireless keyboard and mouse, why not use your Android phone to control your entertainment? *Kore* is the official remote control for Kodi and *Yatse* is a an unofficial yet powerful app. Both apps enable you to navigate your media collection using a well-designed and intuitive interface. *Yatse* also has a series of plugins to enable gesture control and push SMS messages to your TV.

Xbee

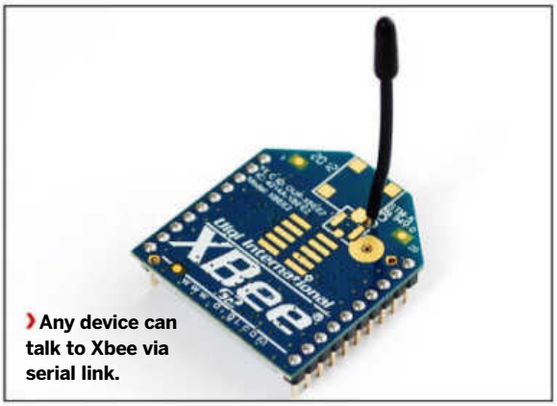
This is one of the easiest ways of automating your home. Xbee uses only four connections for power, ground and data connections, and any device can talk to Xbee via a serial link. Xbee has been used by a lot of early home automation hackers, who've used it to integrate Arduinos to make wireless devices that aren't connected to the internet but still automated.

Particle

The Particle range of boards started life as the Spark Core, an Arduino-compatible board which featured built -in Wi-Fi for



A Bluetooth LE can be used to push data from objects in your home as you wander from room to room.



Any device can talk to Xbee via serial link.

Creating mobile apps to control your home

Creating your own interface for a project is quite an undertaking. For instance, you'll need to consider what language, framework and protocol to use. But what is common among all these considerations is the likely need to control your home automation project using a mobile device. These devices have taken over our lives and it's now just as common for a user to control their TV, music and lighting from their tablet or phone as it is for them to surf the web via such devices. So how can we control our home automation project with a mobile device?

Build your own Android application

Coding Android apps is an involved process that requires downloading the Android Studio SDK (Software Development Kit) and learning how to write apps using it. There is more information via their website **https://developer.android.com/training/index.html**.

A simpler way to write Android apps is to use MIT App Inventor. This uses a web-based development environment which can be used to design the layout and content of a project and code the project using a block-based interface similar to Scratch. The interface, while looking simple and child-like, hides a powerful framework that has access to Google power.

Our first project uses a speech to text application which uses Google's servers to process your voice into text with ease. This project can be adapted to send an SMS to a dedicated number, such as Twilio which can then be pushed to a Pi controlling your home. This means that even from the office you can make sure the central heating is ready for your return home. You can learn more about MIT App Inventor at the official website **http://appinventor.mit.edu/explore**.

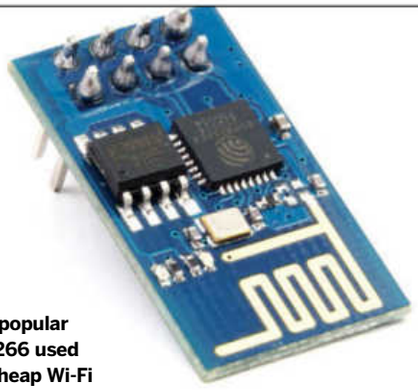
Building a GUI for your Pi

Recently, the Raspberry Pi Foundation has released its latest product, a seven-inch touchscreen for under £50. The Raspberry Pi attaches to the back of the screen and can be powered with just one power supply. Building a user interface for the touchscreen can be accomplished in Python using many methods and two common methods are: first, the Tkinter framework for creating menus and dialogs in a similar manner to a traditional OS and, second, creating a custom interface using pygame, a library for media/video game creation, eg Spencer Organ used the pygame library to create a radio player with a custom user interface (**http://bit.ly/PiInternetRadioPlayer**).

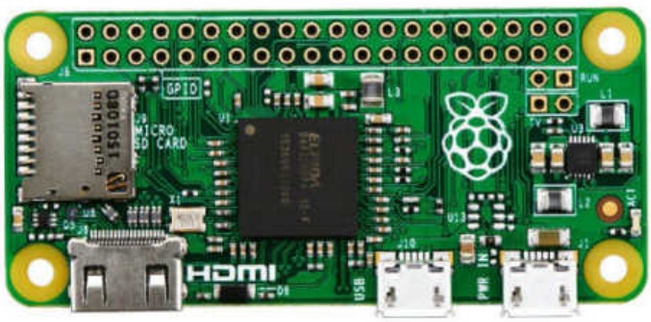
Flask

Flask is a micro web development environment for Python that will slot into your project with relative ease and convert a project into a web app that will work with all devices using a browser. Flask bridges the gap between the web and your project by running a server on your Pi that intercepts input on a web page, eg a hyperlink or button, and it calls a Python function to perform an action. To illustrate here is the code to control an Energenie using Flask (see **http://bit.ly/EnergenieFlask**) created by Ben Nuttall from the Raspberry Pi Education team.

```
from flask import Flask, render_template
from energenie import switch_on, switch_off
app = Flask(__name__)
@app.route('/')
def index():
```



The popular ESP8266 used as a cheap Wi-Fi capable board.



Raspberry Pi Zero is a new £4 unit for creating smart connected home devices.

control and programming from a remote location. After a successful Kickstarter campaign the team produced a cheaper new version called Photon, which provides all of the functionality of the Spark Core at half the price. However, the latest board, called Electron, also offers connections over a 2G or 3G cellular network and enables data to be sent and received from isolated locations, eg an Electron could be placed inside a doorbell where it could send an SMS without any external SMS providers, so If you need to tweak your project from the beach, then you can and upload the code directly to the board at your home.

ESP8266

It's no exaggeration to say that this board has changed home automation and IoT forever. The ESP8266 is a cheap Wi-Fi-capable board that can be programmed using the Arduino IDE, which enables quick integration into any existing project. The ESP8266 has become the go-to board for home automation hackers on a budget, because there are development boards which both enable access to the GPIO and are programmable using the Lua scripting language and Micro Python.

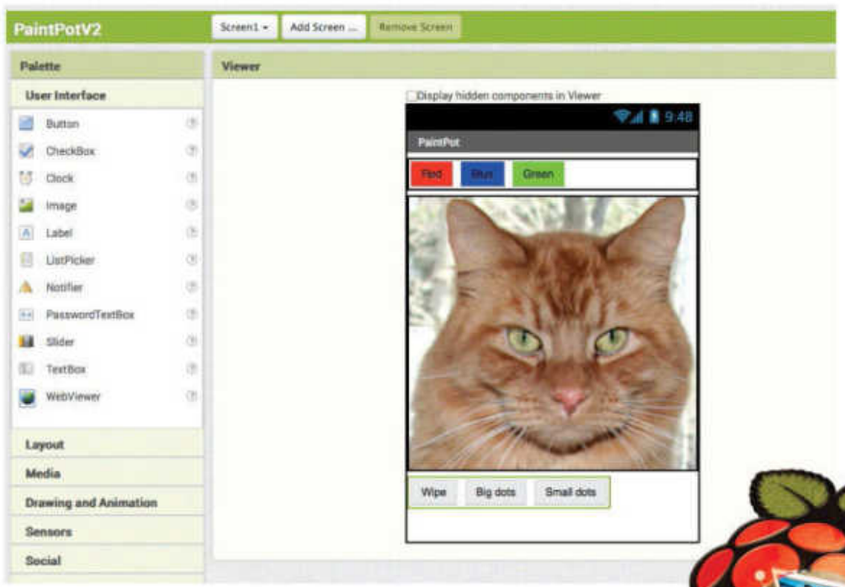
Raspberry Pi Zero

The Foundation has managed to do it again, introducing a £4 version of the Raspberry Pi with a lower-power, cut down Zero model. This is going to make the Pi the go-to device when it comes to creating smart connected devices around the home. With power draw down to as low as 70mA while idle (without HDMI or other peripherals connected) it's even practical to power the Pi Zero off AA batteries. This is still significantly more power than an Arduino device, but then the Pi is way more capable! The Pi Zero also supports the standard GPIO and Linux software which makes it easy to prototype designs on standard Pi boards, optimise them and then deploy on the Zero.

At this point we hope you have enough ideas, starter projects and technical knowhow that you can turn your home into an automated nirvana.

```
  return render_template('index.html')
@app.route('/on/')
def on():
  switch_on()
  return render_template('index.html')
@app.route('/off/')
def off():
  switch_off()
  return render_template('index.html')
if __name__ == '__main__':
  app.run(debug=True, host='0.0.0.0')
```

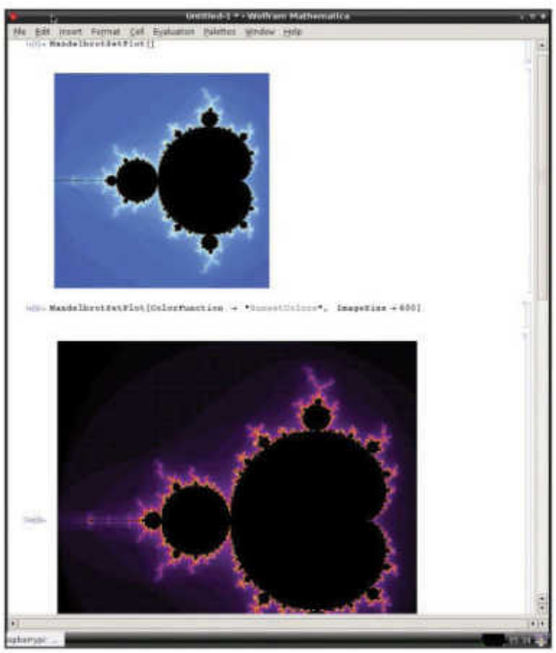
Here we can see that we're importing the Flask and Energenie libraries and creating an instance of the Flask class. Next, we use the route decorator to tell Flask what URL will trigger our functions. We go on and create three functions that will handle loading the index.html template and switching on and off the Energenie devices around the home. Last, we run the Flask app in debug mode, enable a verbose output to the Python shell and set the app to accept connections from all IP addresses. The Python code works with an HTML template that contains the layout and content of the web interface. CSS can also be used to style the web page.



Build a smartphone app to control your smart home cat?

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.

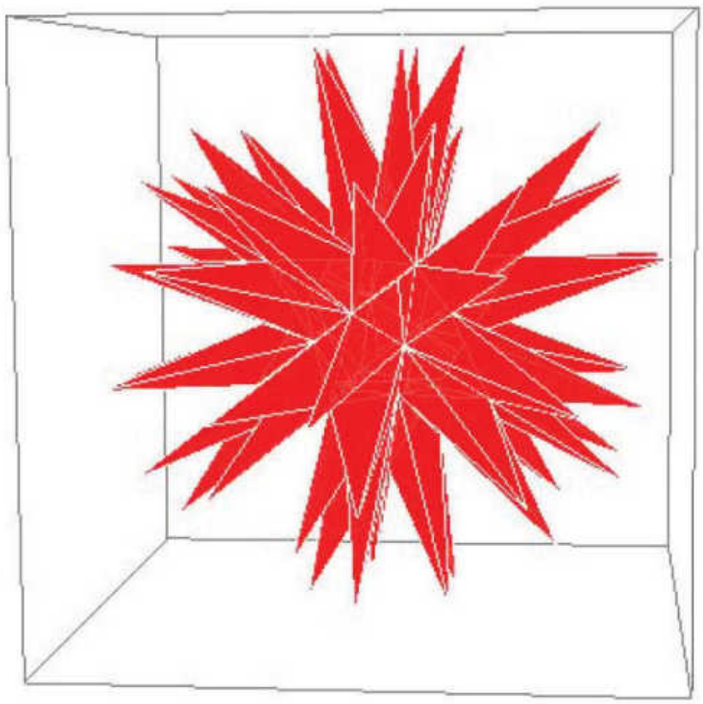
For no real reason, let's try and make Mathematica invert a 20x20 matrix of random floating point values.

```
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:

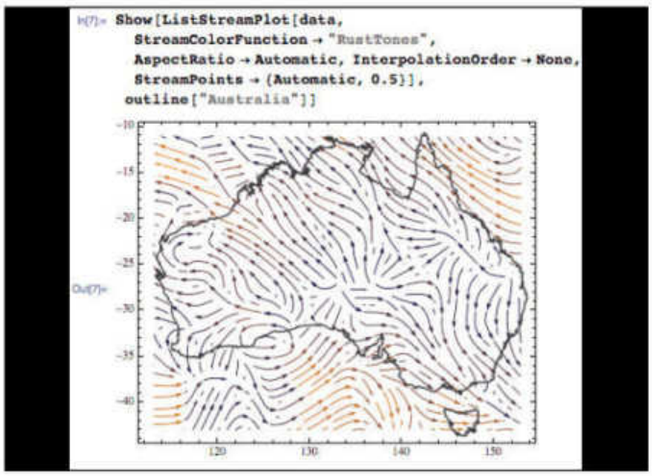
```
D[tan^-1[x^x],{x,3}]
```

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

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

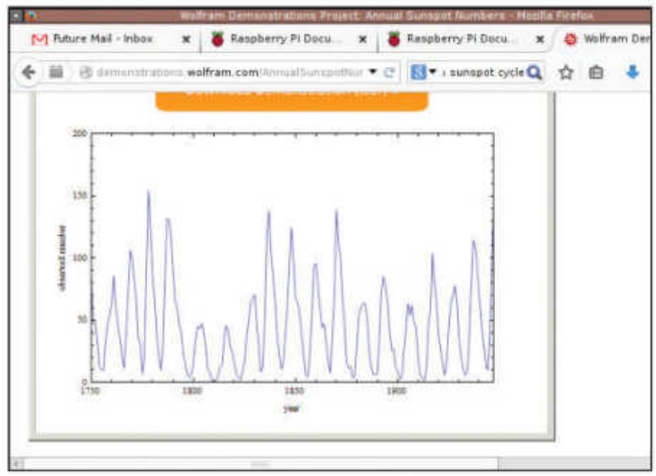
Students often find integration harder than differentation, and for many years now have been using the online integrator at **http://integrals.wolfram.com** to do their homework. As you would expect, Mathematica can symbolically integrate pretty much any function where this makes sense (there are

Things to try with Mathematica



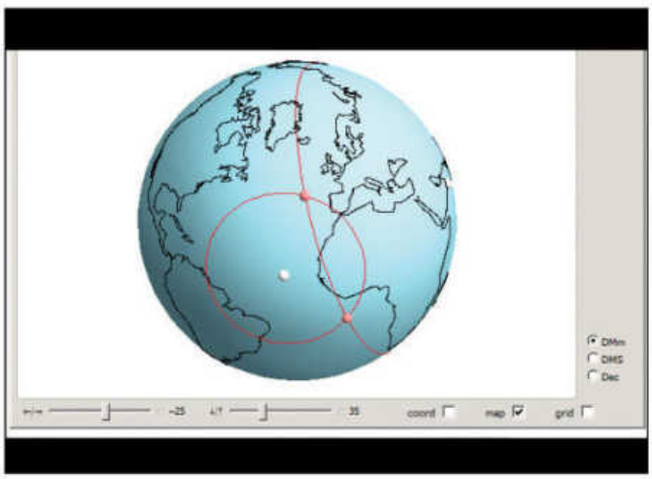
1 Stream plot

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**).



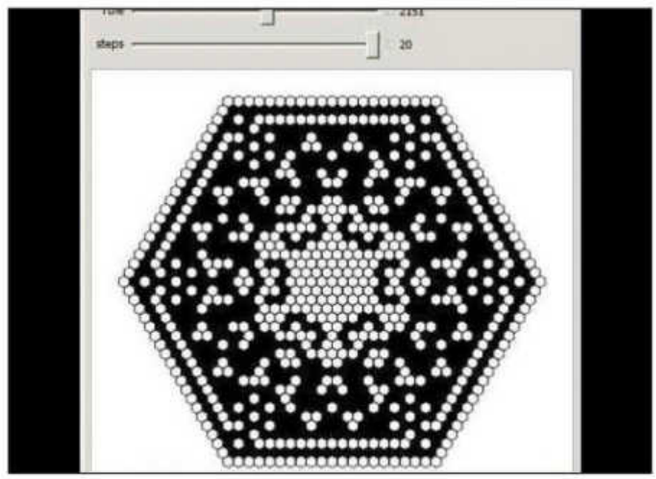
2 Sunspot Cycle

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"]
```

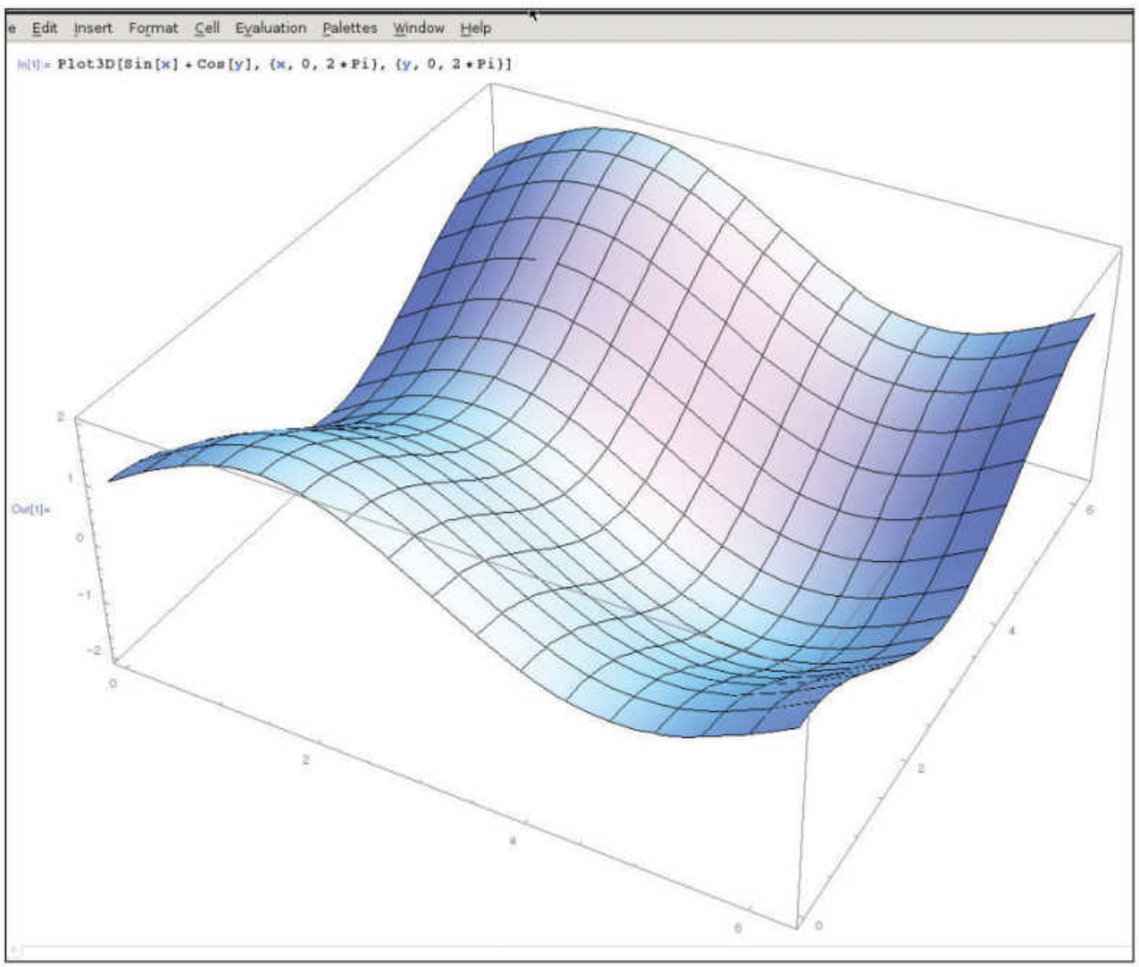
And then you can export it as a JPEG, or whatever format you like using the following:

```
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.

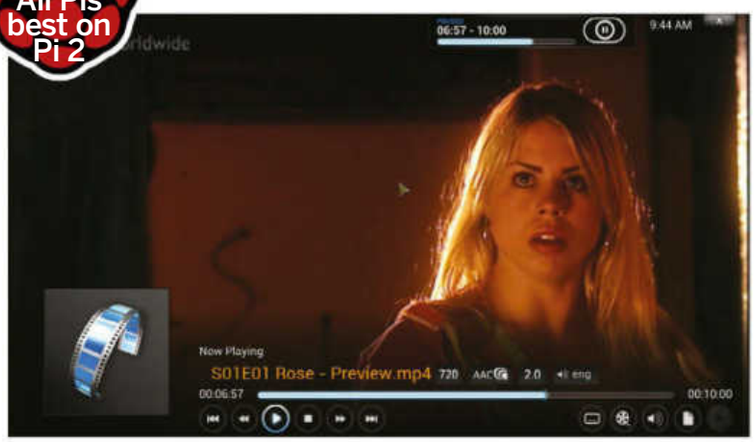
Kodi 14: Set up a media centre

Upgrade to the latest OpenELEC 14 and take your home entertainment setup to the next level.

You can use the Raspberry Pi to do all kinds of geeky things but one of the most interesting uses for the 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.

COMPATIBILITY All Pis best on Pi 2



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 for with OpenELEC for many people 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 Raspberry 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 you need to insert the card into the Raspberry Pi, hook it up to your TV via the HDMI port and power it on. OpenELEC boots up pretty quickly and will take you straight into *Kodi*. If you've used the media player (or its predecessor *XMBC*) before on the desktop, you shouldn't have any issues navigating it on the Pi. However, you will need to spend some time configuring the media centre 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

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 HDTV. 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'.

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.

Enjoy software freedom

Discover Linux!

OUT NOW!

DELIVERED DIRECT TO YOUR DOOR

Order online at **www.myfavouritemagazines.co.uk**

or find us in your nearest supermarket, newsagent or bookstore!

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.

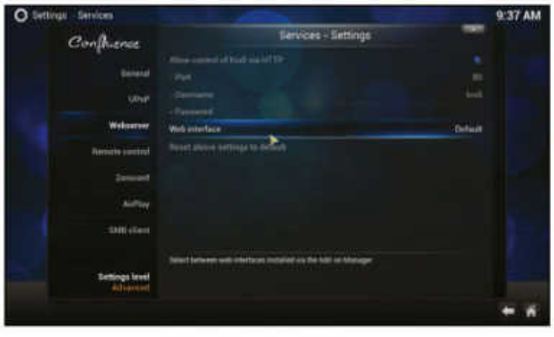
configuration options related to the distro under five different categories. From the System category, you can change the hostname of this OpenELEC installation. This is handy for distinguishing multiple instances, in case you have more than one system on your network – one in the living room, say, and 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 and ask the distro to fetch the update without asking for your approval.

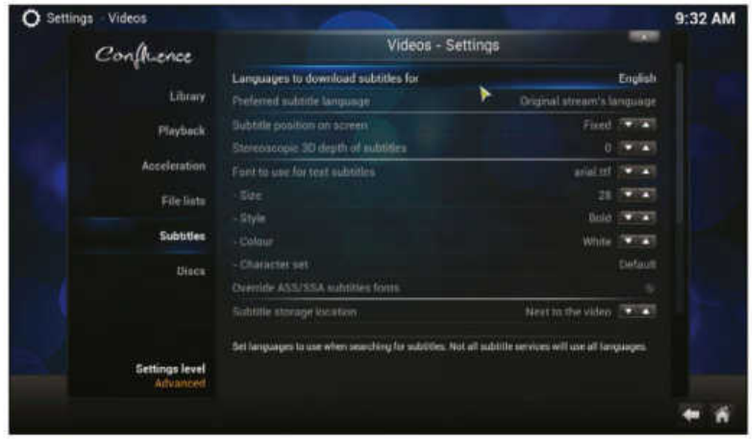
For a better HTPC experience, you can use OpenELEC with a Raspberry Pi-compatible Wi-Fi adaptor. Once you've connected the adaptor, head to System > OpenELEC > Network, and toggle the Active option under the Wireless Networks section. Then switch to the Connections section and select your Wi-Fi network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

There are also a whole host of settings available in *Kodi* , which are listed under Settings > System. Using these settings you can configure such things as audio output, calibrate the monitor, set up remote controls and set up *Kodi's* built-in PVR etc.

To watch TV on your Raspberry Pi HTPC, head to Settings > System > Live TV, and toggle the Enabled option. You will then be 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 altering the behaviour of the onscreen display, the length of the recording and parental controls etc.



Your HTPC includes a small web server that enables you to control *Kodi* from a skinnable web interface.



You can set up *Kodi* to search for and download subtitles for all of your media files 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.

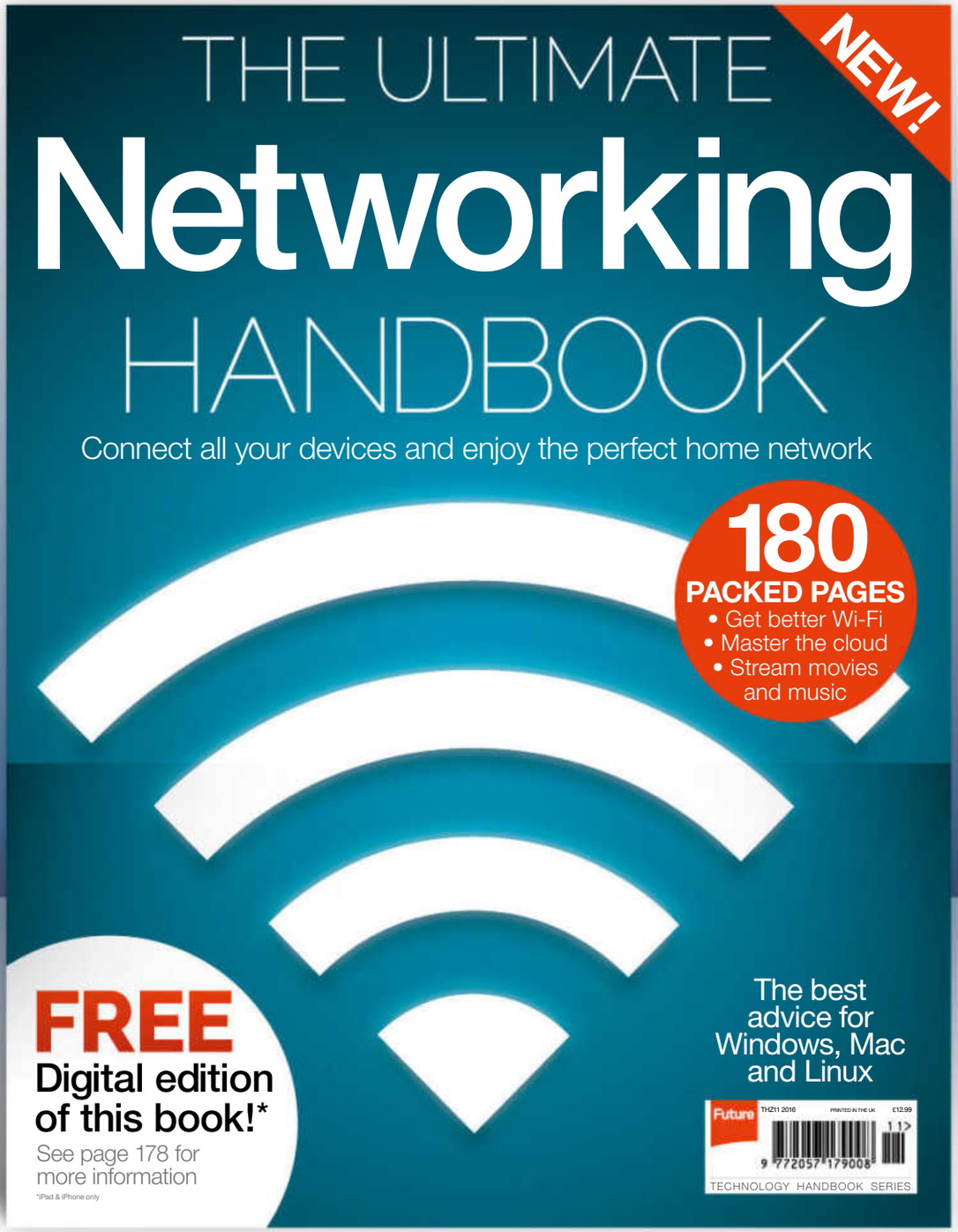
To define a media source, enter any of the Videos, Music or Pictures entries on the screen, and click Add Sources. Use the dialog box that pops up to browse to a source that contains some media. Adding media on the card or plugged-in USB drives is pretty simple and straightforward. But if you wish to pull in content from another computer on the local network, you have to define the network shares first.

Kodi supports several 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* allows you to choose a scraper – a special plugin that fetches metadata »

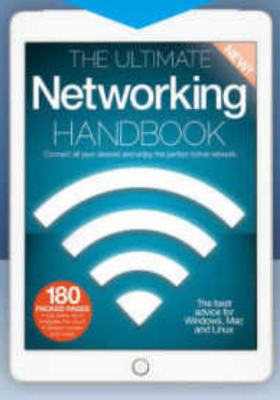
Quick tip

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

GET CONNECTED ON ALL YOUR DEVICES



OUT NOW!
WITH FREE DIGITAL EDITION



DELIVERED DIRECT TO YOUR DOOR

Order online at **www.myfavouritemagazines.co.uk**
or find us in your nearest supermarket, newsagent or bookstore!

» 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 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.

Quick tip

Use NFS instead of SMB to access media quicker.

Remote playback control

Your HTPC will now enable 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 Android and iOS devices, and, shockingly enough, 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 Google Play Store.

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 (repos), including the official OpenELEC and *Kodi* repo. Select the repo you wish to install from. There's also the All Add-ons option, which displays plugins from both these repos.

2 Select category

Once you've selected a repo, you're shown a list of add-on categories. The *Kodi* repo includes a lot more categories than the OpenELEC one. The OpenELEC repo mostly includes drivers for various devices, while the *Kodi* repo 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 its category, eg video plugins install under Video on the main page.

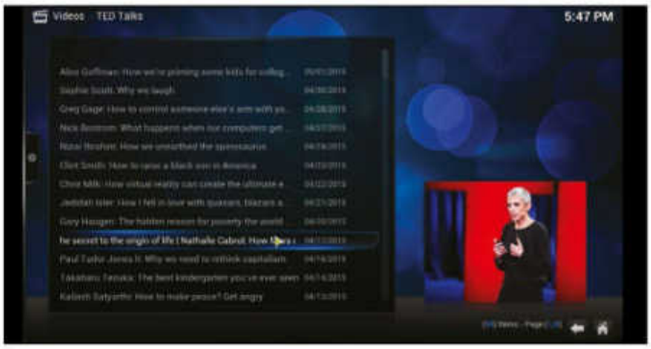
Must-have add-ons

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 with 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 enable you to pull in images from these websites, and others such as the Hubble Space Telescope. Once enabled, different websites give you different options, eg 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 etc. You can find add-ons for each of them and others 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 can use a nice interface to browse through music and control playback.

5 Monitor downloads

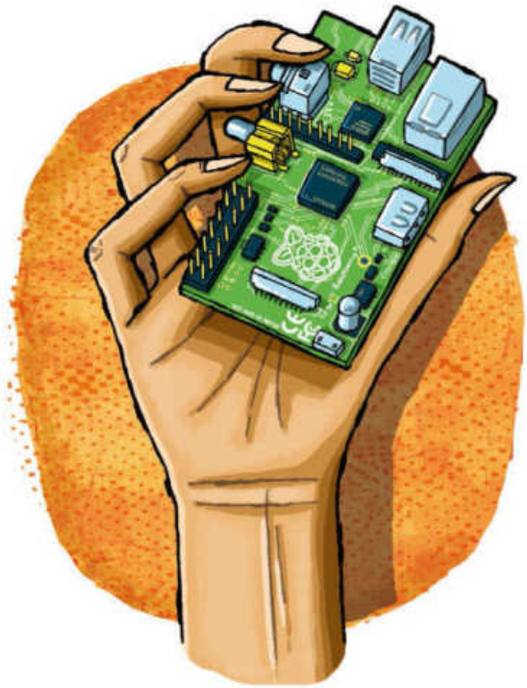
There are other things you can control remotely, eg you can connect to, monitor and control downloads on a remote computer. The *Transmission Client* add-on available under Programs can connect to another *Transmission* client on the network. For this to work, you need to enable the Remote Control feature 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 and allows you to play your classic games. Setting it up is quite involved, so go to **http://bit.ly/ROMCollectionBrowser** to get it to work.

CUPS: Printing

Using the Raspberry Pi as a wireless print server is easy when you have a Raspberry Pi and CUPS. Let's find out how.



COMPATIBILITY ALL Pis

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.

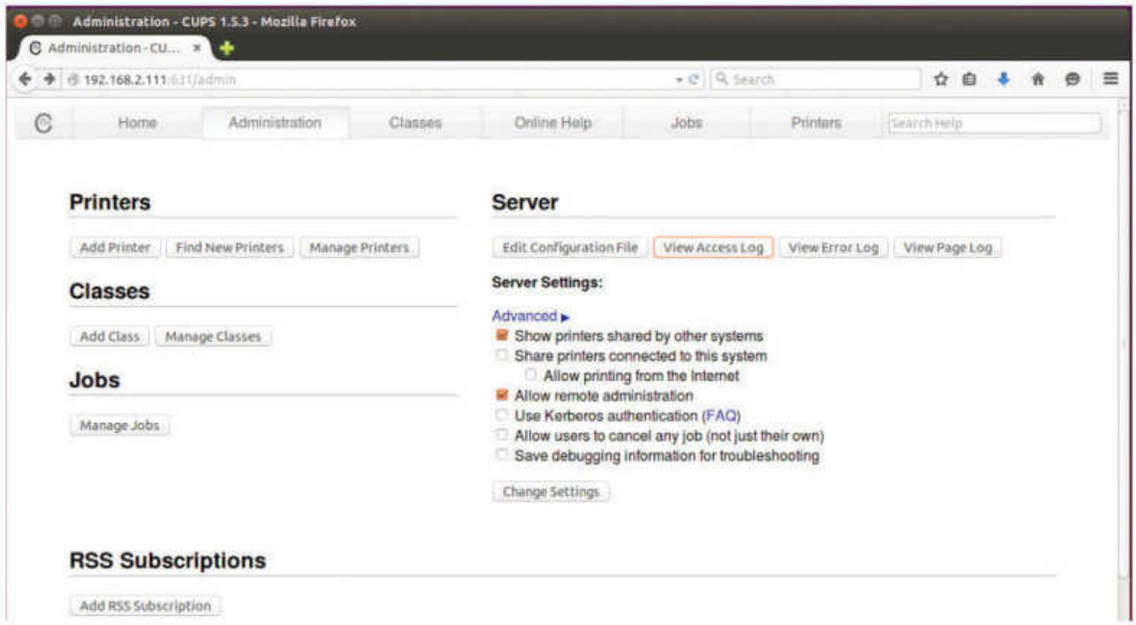
If you haven't already used your printer on Linux, before you get started with this project head to **www.openprinting.org/printers** to check whether your printer is compatible with the *CUPS* printing server software. If your printer is listed, hook it up to the Raspberry Pi using one of the USB ports. 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 with:

```
$ sudo ssh pi@192.168.3.111
```

Install CUPS

Once you're inside Raspbian, update the repositories (repos) 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 called pi. Use the following command to allow it to



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 two of the walkthrough (*see p66*). 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.

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 configuration 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 Raspberry Pi via Ethernet instead of a wireless adaptor, modify the command and replace `wlan0` with `eth0`. 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.

Network-wide access

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 line:

```
guest ok = yes
```

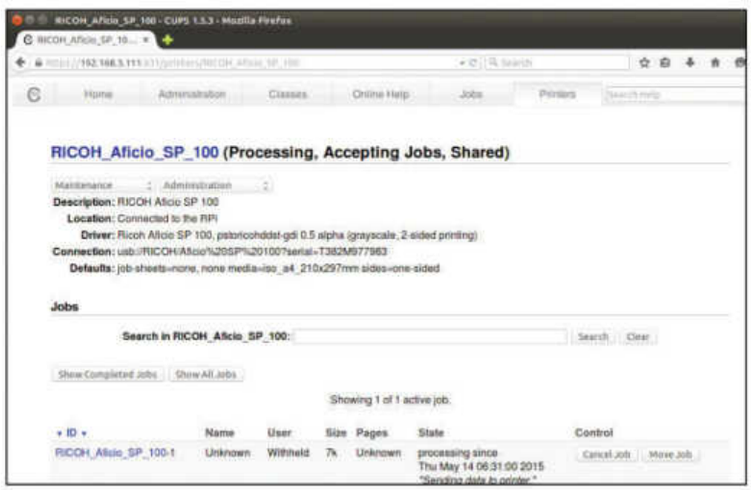
Then scroll down to the `[print$]` section and change its path to the following:

```
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 line `wins support = yes`.

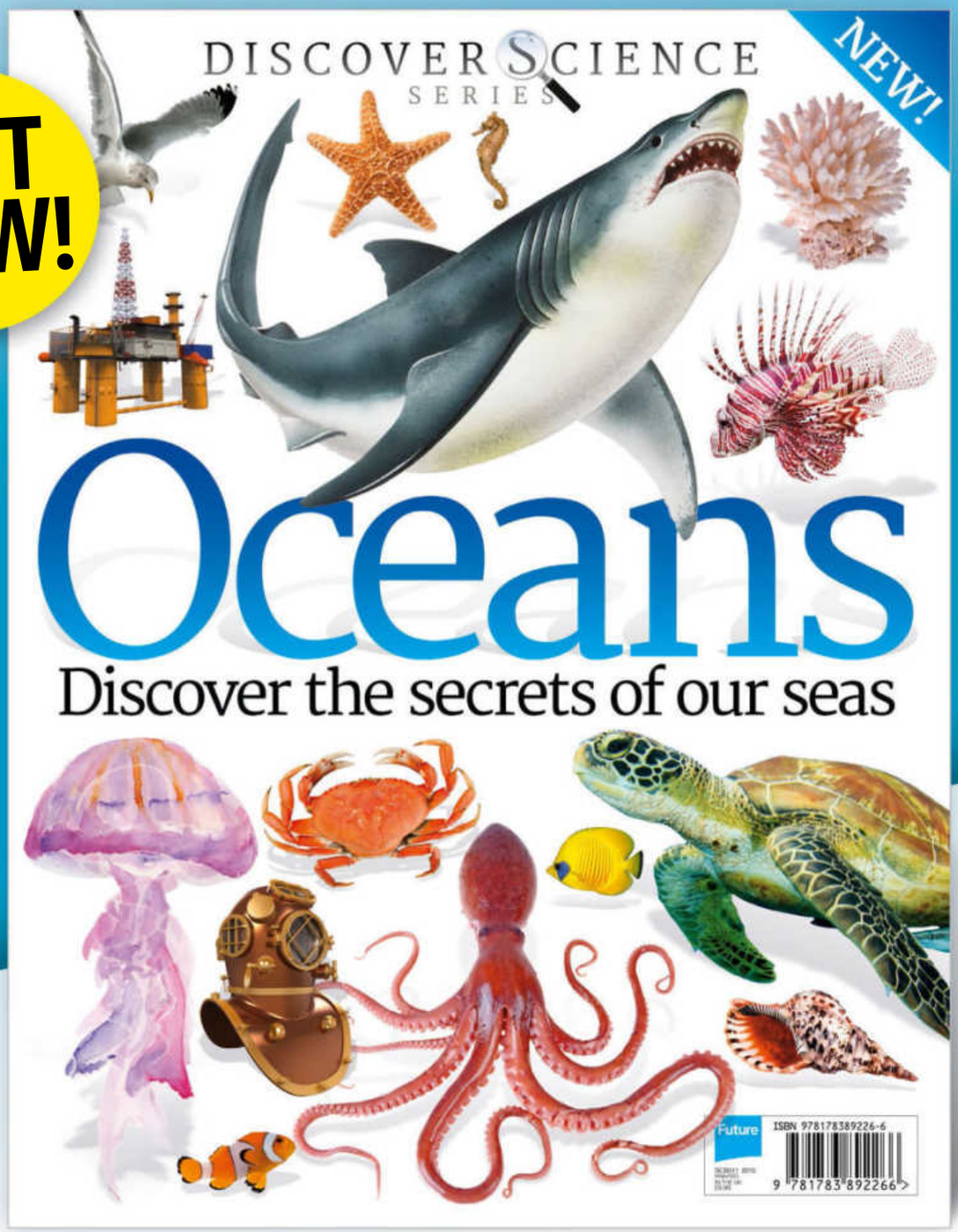
Now save the file and restart *Samba* with `$ sudo /etc/init.d/samba restart`.

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 will detect and list any printers hooked up to the Pi. If you have Apple devices, you can enable support for Apple's 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. »



From the Printers tab, you can track the status of every job on every printer.

DELVE DEEPER INTO NATURE



OUT NOW!

DELIVERED DIRECT TO YOUR DOOR

Order online at **www.myfavouritemagazines.co.uk**
or find us in your nearest supermarket, newsagent or bookstore!

In addition to the ability to use our network printer from within graphical applications 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.

Print from Python

The *CUPS* printing server installs a bunch of command-line tools *(see Administering CUPS on p67)* 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 by using 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"]
```

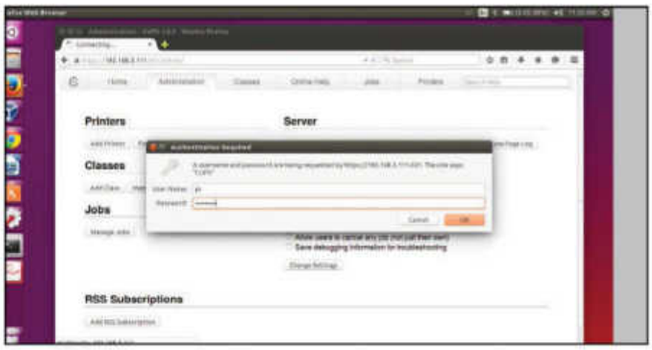
The script fetches details about all the printers managed by *CUPS* and prints their name and device address to the screen. When you execute the script, it produces an output similar to the following:

```
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 printFile function, by specifying them in the format:

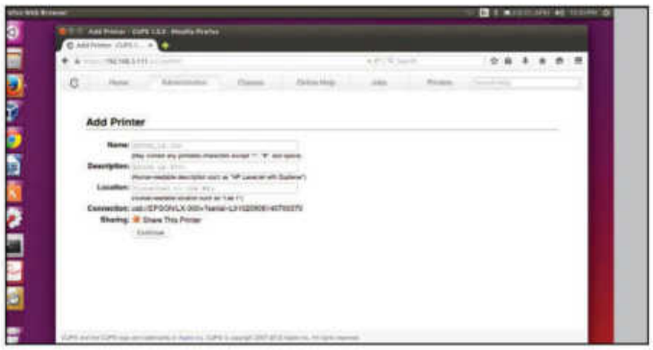
```
$ printFile (name of the printer, filename to print, job title,
options)
```

Add a printer



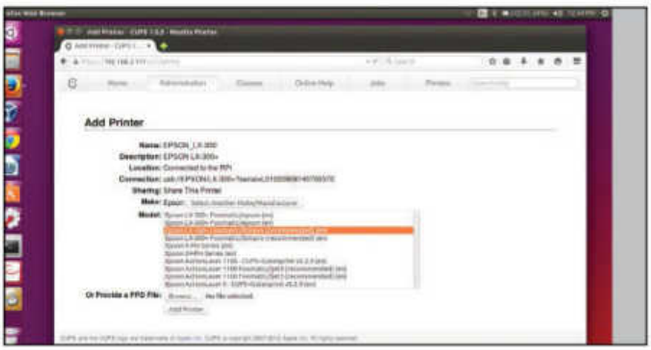
1 The CUPS dashboard

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 into the interface using the credentials of the user that you've added to the lpadmin group, which in our case is the pi user.



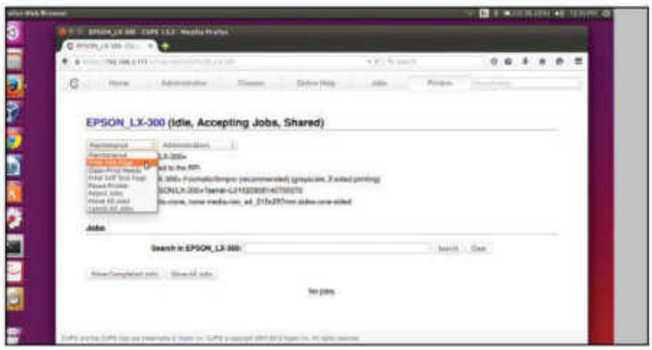
2 Add a printer

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 be 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.


T3
DISCOVER THE FUTURE OF AUTO TECH IN TODAY'S CONNECTED WORLD
ONLINE • PRINT • TABLET
APPLE WATCH
Pre-condition and open your car
BMW i3
The compact electric vehicle to die for
LIFE'S BETTER WITH T3
t3.com

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.

Converting from HTML to PDF

A more interesting way to convert HTML pages into PDF file is to use the *wkHTMLtoPDF* toolkit, which passes on the PDF 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
```

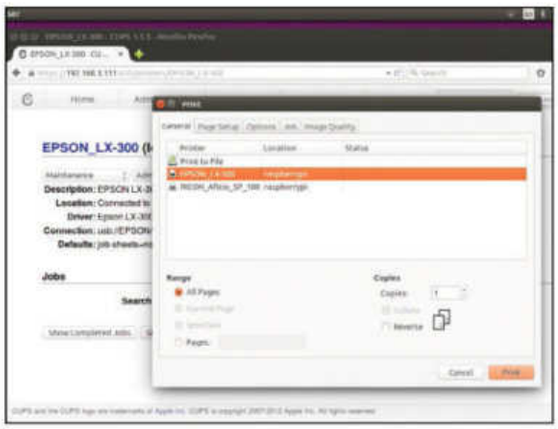
before installing the Python wrapper with:

```
$ sudo pip install git+https://github.com/qoda/python-wkhtmltopdf.git
```

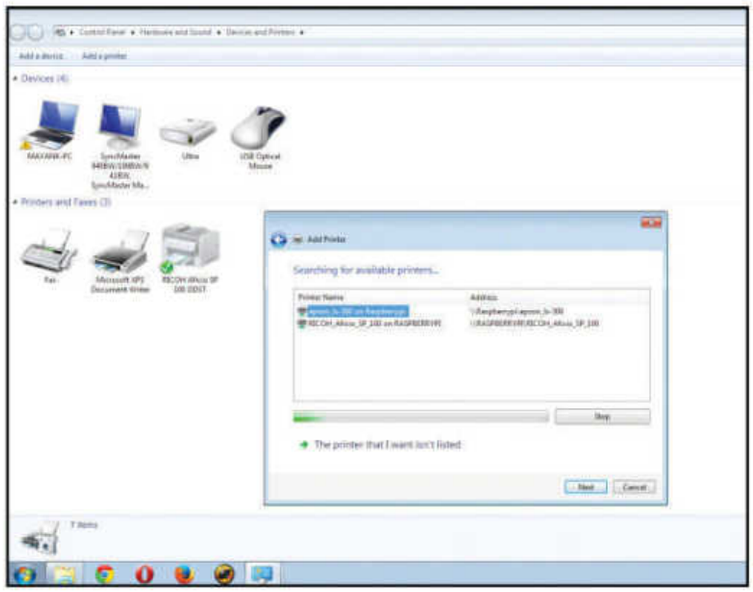
You can now use the following to convert a web page into a PDF file:

```
from wkhtmltopdf import WKHtmlToPdf

wkhtmltopdf = WKHtmlToPdf (
    url='http://www.linuxformat.com',
    output_file='/home/pi/docs/lxf.pdf',
)
wkhtmltopdf.render()
```



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.

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 all the pieces fit together – *wkHTMLtoPDF* 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.techradar.com',
    output_file='/home/pi/techradar.pdf',
)
wkhtmltopdf.render()
conn = cups.Connection()
printers = conn.getPrinters()
for printer in printers:
        print printer, printers[printer]["device-uri"]
file="/home/pi/techradar.pdf"
printer_name=printers.keys()[0]
conn.printFile (printer_name, file, "PDF Print", {})
```

The script first converts the **www.techradar.com** home page 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!

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.

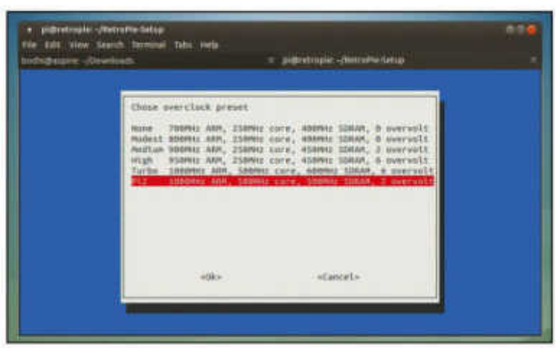
RetroPie: Use an Xbox gamepad

Relive the golden era of gaming by hooking your Pi up to a game controller.

COMPATIBILITY ALL Pis

Video games in the 1980s 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 storylines 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 that golden era.

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 distribution (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.



Just to be on the safe side, try playing games with the default clock speeds, before you try to overclock the Pi.

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
```

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, and RetroPie can work with various controllers, from cheap no-names ones to PS3 and Xbox 360 controllers.

Once you've 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 *XTerminal*.

Basic setup

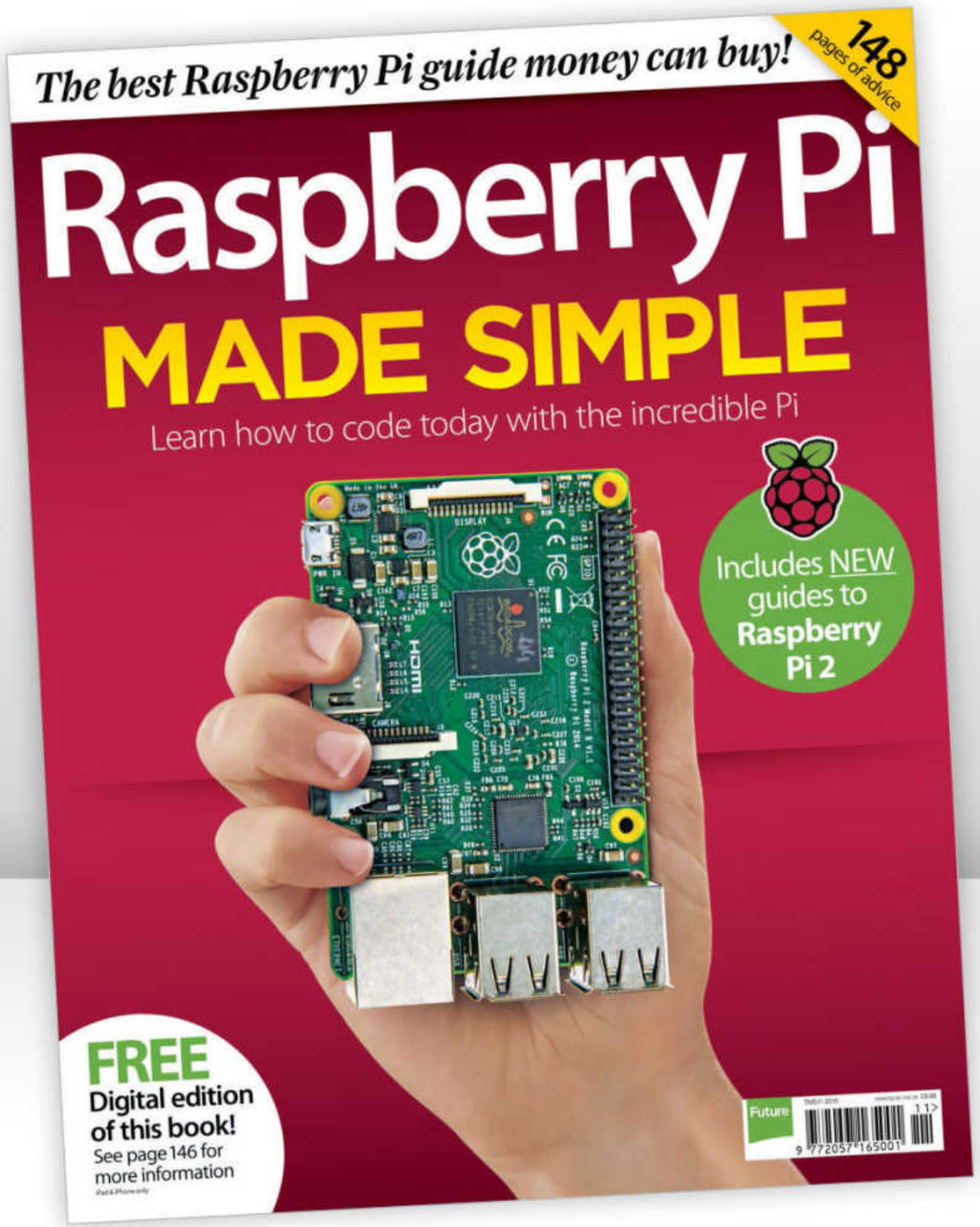
The first order of business is to expand the image to take over the entire card. In order to do this, bring up Raspbian's configuration utility with

```
$ 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 'Finish' to restart the Raspberry Pi and save changes.

CODING, PROJECTS AND LINUX SKILLS



OUT NOW!
WITH
FREE
DIGITAL EDITION

DELIVERED DIRECT TO YOUR DOOR

Order online at www.myfavouritemagazines.co.uk
or find us in your nearest supermarket, newsagent or bookstore!

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 a*nd enter the following in the CLI:

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

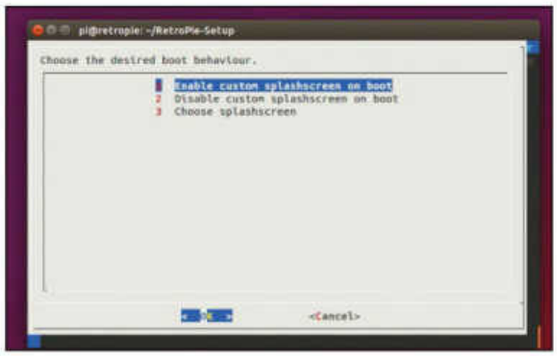
```
$ sudo apt-get upgrade
```

The above commands refresh the distro's repos 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 want to, you can make changes to the splash screen using the RetroPie-Setup script.

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 in the `wpa-ssid` line 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's **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 '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 command line. 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 '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 chosen 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 will pick it up. Press and hold any key on the controller to help the distro correctly identify the controller. You will then be asked to map the keys on the controller. Be aware that this basic mapping is only for navigating around 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

Emulation Station displays the number of games inside a particular emulator.

The home of technology

techradar.com

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 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.

Quick tip

When using multiple controllers at the same time, it's best to use identical controllers to avoid configuration and gameplay issues.

Controller drivers

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 – '318' to install the PS3 driver and number '332' to install the driver for the Xbox 360.

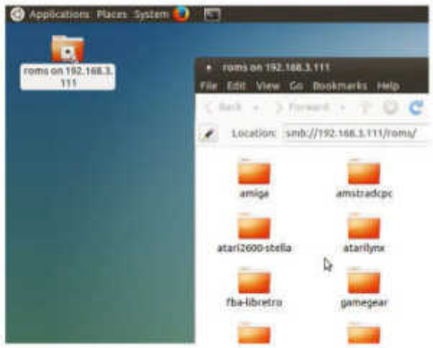
The Xbox360 script downloads the xboxdrv driver and edits the **/etc/rc.local** file to start the driver on boot. The script adds entries for wired 360 controllers. If you are using wireless controllers, open the **/etc/rc.local** file in a text editor, hunt for the lines that begin with `xboxdrv` and 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
```

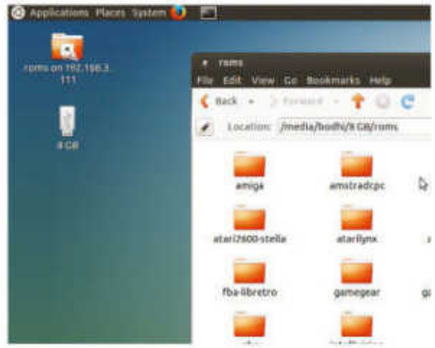
This nifty little utility should detect the Bluetooth adaptor

Transfer ROMs



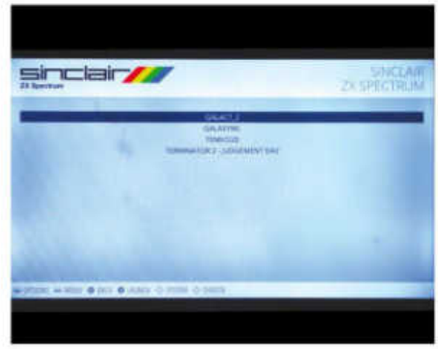
1 Network transfers

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 to the *XTerminal* 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 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.

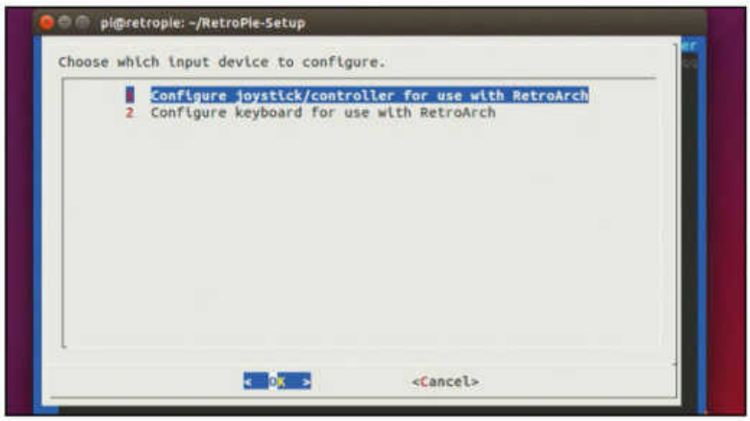
and make it known to RetroPie.

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:

```
$ 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 websites, 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.

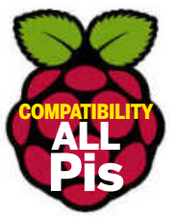
Whatsapp: Build a PiBot monitor

Find out how to interact with and control your Raspberry Pi via popular messaging platform Whatsapp.

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, then your Pi is probably tucked away somewhere that's not that 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 Raspberry 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 use this to send notifications to us via instant messages whenever a predetermined event is triggered. First up, you'll need to 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 **https://jabber.hot-chilli.net** service, which gets a top-notch security rating from **https://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




```
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 isn't much documentation for the Yowsup library, but it does ship with some useful example applications, which make for an interesting study.

Video chats

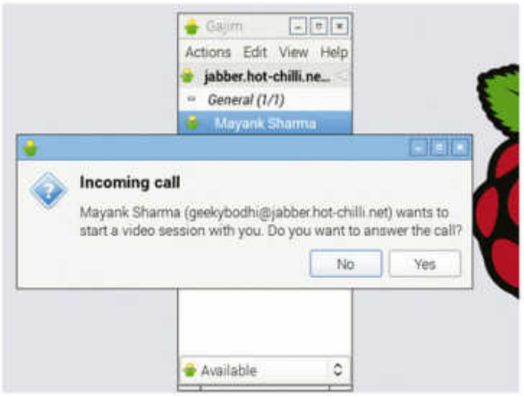
If you're using the Raspberry Pi 2 as a regular desktop, you can install an IM client and converse with your friends either via text or video. The XMPP protocol we've discussed in this tutorial is one of the most popular ones for exchanging instant messages and is used by services, such as Google Chat.

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* 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.**

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 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 with:

```
$ 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'll 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 (*see p69*) or nab it from **http://pastebin.com/NdQw5frt** 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. eg 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 in 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 phone with the SIM card for the number you've entered in the **mydetails** file. The message contains a six-digit code. Use this to register the 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'll receive a response from *WhatsApp* on the Pi that will look 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 application, which you can use to send and receive messages. Bring it up with:

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

This brings up the *Yowsup* command line 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, no matter how many times we 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, eg 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 *(right)* 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 *(right)*. Now create a shell script called **talktome.sh** that calls the **pitalk.py** 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.

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.

Parsing the script

Then we define the functions that do the actual task we query the Pi for via the *WhatsApp* messages, eg 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, eg `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 applications.

Listings

In this tutorial we use two scripts which are available here: **http://pastebin.com/NdQw5frt**.

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")
```

OMV: Build a low-power NAS

Discover how to manage your data better with OpenMediaVault and your own low-powered, Pi-centric network attached storage box.

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.

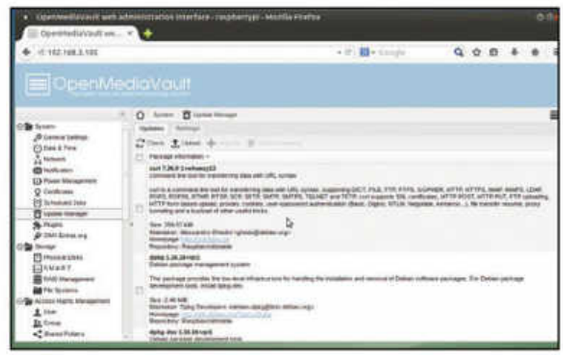
Get installed

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.

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 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.



A useful tip to bear in mind: head to System > Update Manager and make sure you install all available updates.

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

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 according to 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 option 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 in 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* 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. In addition, 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.

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, then 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.

Adding Users

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.

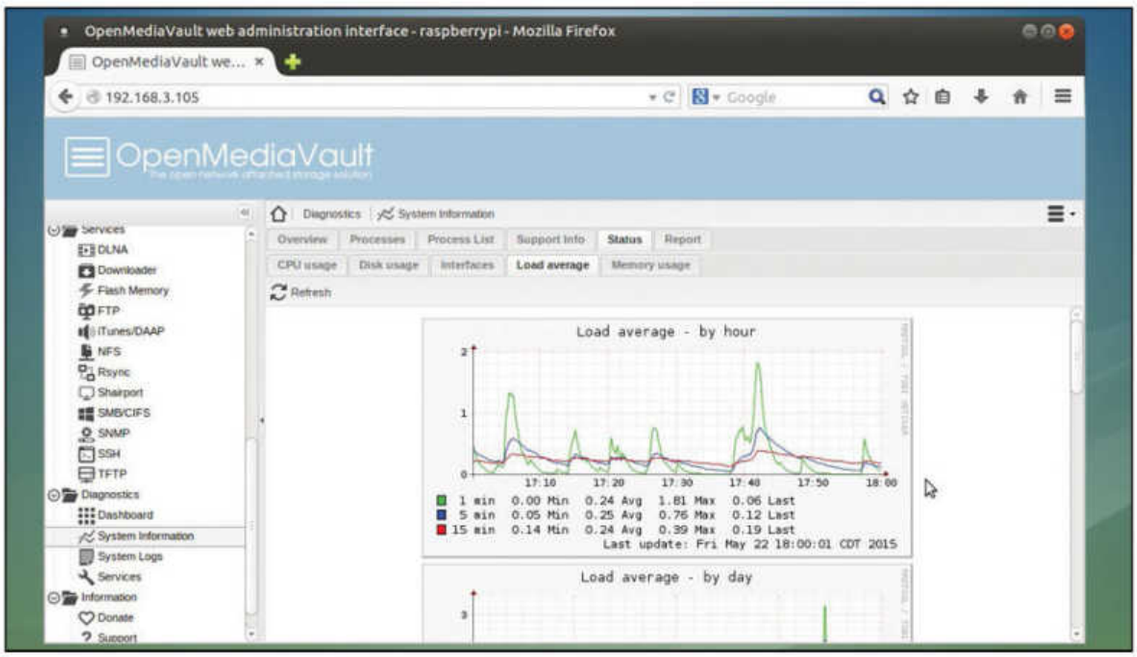
If you want users to have their own **home** directories in the OMV server, switch to the Settings tab and tick the box to enable the **home** directory for the user. You must also specify the location for the **home** directory by selecting an existing shared folder on the NAS server or creating a new 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 with different 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.

Quick tip

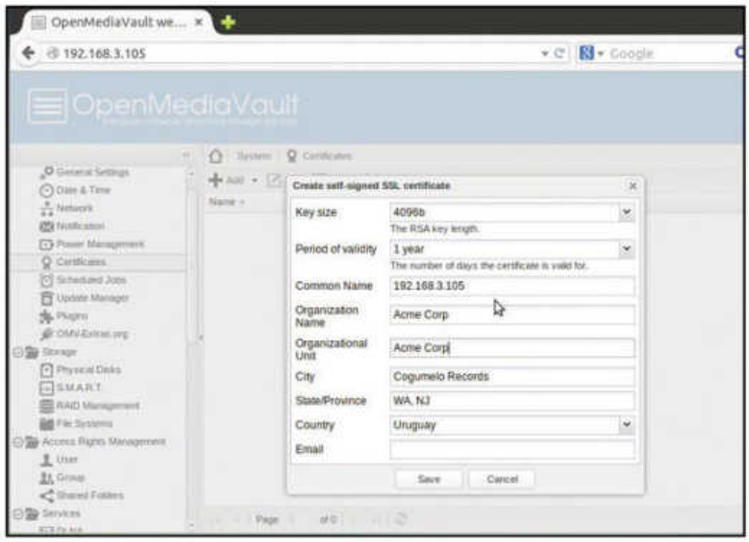
If you wish to use the NAS as the target location for storing backups, enable the FTP service. Also enable the SSH service to manage the OMV installation from the CLI.



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 on which it's running. Go to Diagnostics > System Information to see for yourself.



You can create self-signed security certificates if you don't wish to transfer data to and from your NAS device over unsecured HTTP.

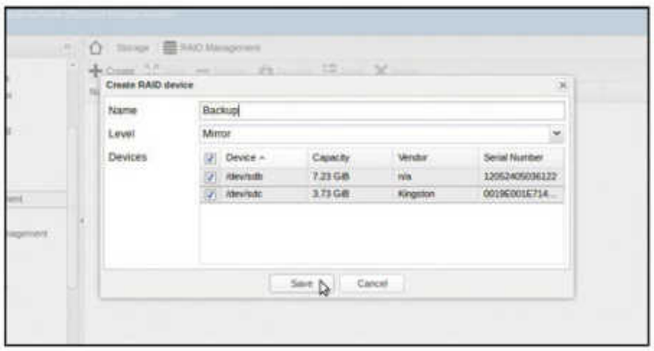
Play close attention to the Permissions setting. By default, OMV only allows the administrator and any users you've added to read and write data to this folder, while others can only read its contents. This is a pretty safe default for most installations, but the distro offers several permutations and combinations of permissions that you can select from the pull-down menu.

Fine-tune permissions

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 tickboxes for controlling their access to that folder, so for example you can allow read-only access.

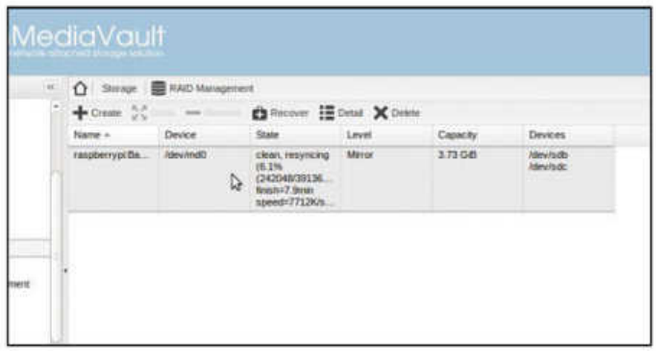
With the users and shared folders set up, you're now ready to share the NAS storage with your network. Follow

Set up a RAID



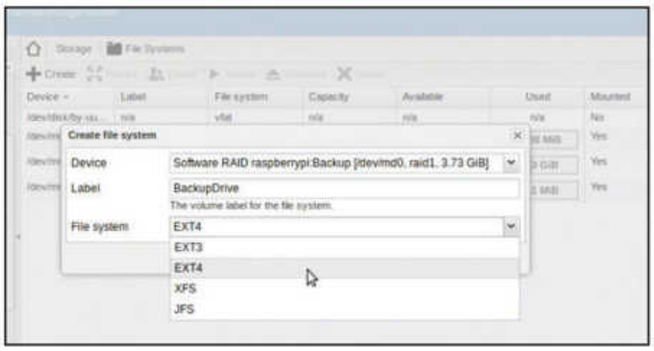
1 Select RAID Level

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.



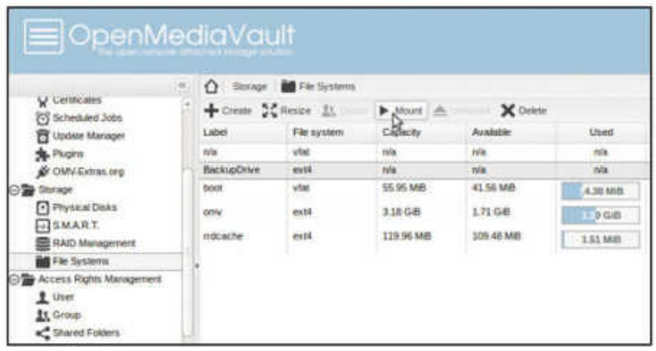
2 Initialise the RAID

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 the 'Create' button. In the dialog box that pops up, select the device you want to format using the pull-down menu, which will have the RAID device you've just created in the list. Then label it and select one of the supported filesystems.



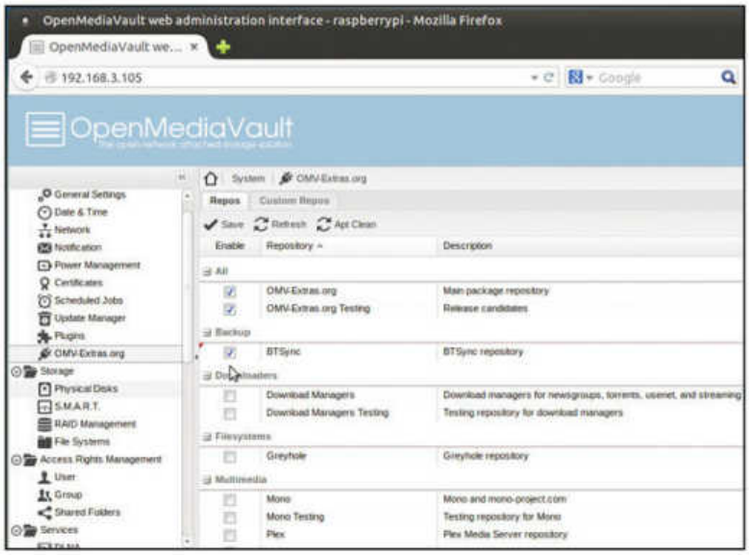
4 Mount the device

After the filesystem has been created and the disk has been initialised, the RAID device will be 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.

the walkthrough to enable a network service that people can use 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.

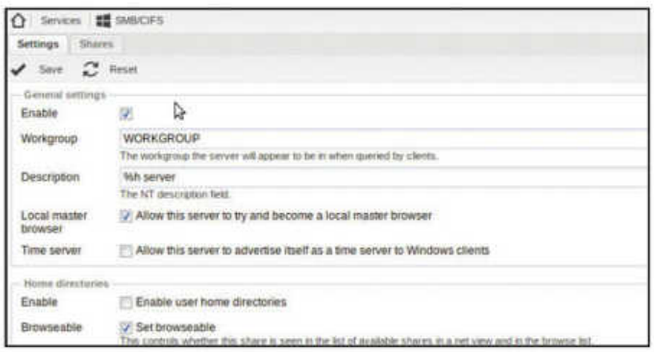
Once you've created a network share, you can access the shared folders from anywhere on the network, irrespective of whether they reside on an individual disk or a RAID array. You can either use your file manager's built-in Network feature to access the network shares, or enter the IP address of the 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 a 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 the case of a regular folder.

It might take a little 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

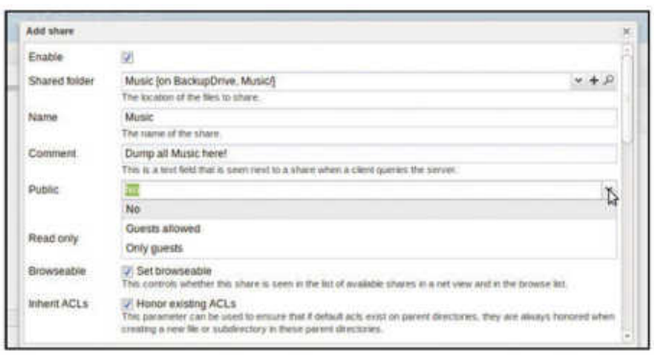


1 Enable Samba

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 click the 'Enable'. The other settings mentioned on the page are optional, so leave them for now. When you're done, click the 'Save' button.

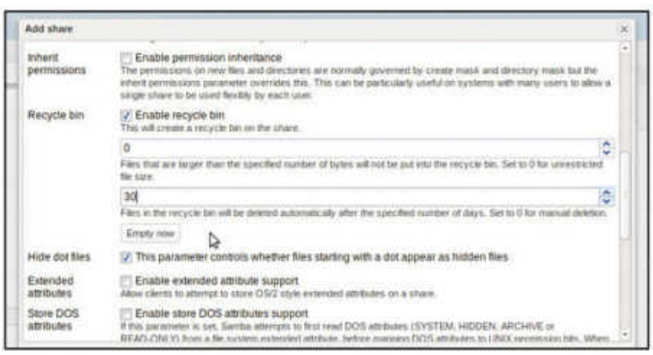
2 Add folders

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 click 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 this is 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.

Scratch: Hack it

If you wish to capture the Space Dragon, first you'll need to build and code a rocket using the visual programming tool, Scratch.

September 2015 saw the release of Raspbian Jessie, and with it the Raspberry Pi Foundation released its own version of Scratch. In this project we'll learn more about Scratch and use it to hack together a space game, which we will control using our very own controller.

For this project you'll need any model of Pi, the latest Raspbian, three momentary switches, Male to Female jumper cables, three 220-ohm resistors, a breadboard and three LEDs.

We'll start by building our controller. We'll connect our buttons to the breadboard and then use the jumper cables to attach one corner of the button to a GPIO pin, and another corner to a Ground (GND) pin. Similarly, we connect our LED's anode, the longer leg of an LED, to a GPIO pin and connect the shorter leg, the cathode, to GND via a 220-ohm resistor. For a detailed overview and all the required code see **http://bit.ly/LXF207-Scratch-Diagram**. The preferred layout of the GPIO pins is Broadcom (see **http://pinout.xyz**) For the GPIO pins we'll use for the buttons, please see the table on this page (right).

GPIO pins for the buttons		GPIO pins for the LEDs	
Game start	GPIO 2	Red	GPIO17
Left	GPIO14	Yellow	GPIO27
Right	GPIO23	Green	GPIO22

Let's power up our Pi and start building our game. Raspbian will automatically log you into the desktop. Navigate to the main menu and open Scratch in the Programming menu. In Scratch, the left-hand column contains a palette of blocks, separated into groups according to their function. The blocks can be dragged into the centre column, which is where we build our code. In the final column is the Stage, where all the output for our game will take place. At the bottom right you can see all the sprites used in the game.

To start, let's make the Stage look more spacey. Click the 'Stage' icon, at the bottom right of the screen. This changes the focus of any code to the Stage. In the centre column, find the tab labelled 'Backgrounds'. Click this, then Import a new background. We chose 'Stars', found in the Nature folder. Now click on the cat sprite, and change its name at the top of the screen to something appropriate like 'Rocket'.

Click on 'Costumes' and you will see that this sprite has two, used for animation. To change our cat into something more space-related, we can paint a new costume. I used a small rocket, which is included in the code download for this project. Click 'OK' to load the new costume, and then select that costume as the default by clicking on it.

Keeping your focus on the Rocket sprite, click 'Scripts'. We'll now create our first script for the Rocket. From the Control palette, drag 'Click on Green Flag' to the coding area. Also from the Control palette drag the Broadcast block to the coding area and attach it to the Green Flag block. In the Broadcast block click on the drop-down, select New/Edit and type `gpioserveron`. This will start the Scratch GPIO server when our game starts, enabling access to the GPIO in Scratch. Now we need to configure our buttons as inputs. Add another Broadcast, and in its drop-down select New/Edit and type `config2in` to set GPIO2 as an input. Add two more Broadcasts and repeat, using `config14in` and `config23in` respectively for GPIO pins 14 and 23, our left and right buttons. To ensure that the pins are configured, click on the

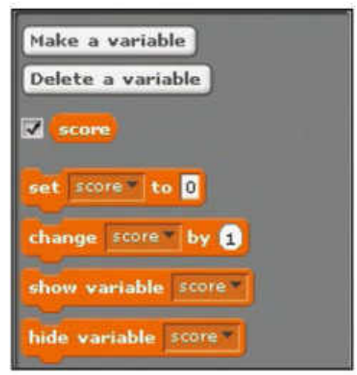
COMPATIBILITY ALL Pis

Extension activities

Our game is good, but what will make it great is scoring and a time limit. Firstly our score is a variable. Select the Rocket sprite, click on the Variable palette and choose 'Make a new variable'. Call it score and set it for all sprites. We'll use a 'When I receive start' Broadcast to trigger the scoring system to life. Every time the game starts, we set the score to 0, found in the Variables palette.

Now add a Forever loop from the Control palette and then use three If statements stacked on top of each other in the loop. The first If conditional uses the 'touching sprite' block in Sensing to advance our score by 10 points if we touch the Space Dragon, before waiting for 1 second. We use the 'change score by' block to do this. The other two If statements work in the same way but deal with touching the obstacles, and points are deducted using minus values.

Our timer is linked to the Stage, so change focus to this. Again we'll use a 'When I receive start' Broadcast to trigger the timer. Create a new variable called timer and using the 'set __ to 0' block, set the timer to 30. Now use a 'repeat 10' loop from the Control palette but change the 10 to 30. Next use 'Change __ by' to change the timer by -1 every time the loop iterates. To control the speed, put a 'Wait 1 second' from the Control palette in place. Outside of the loop, we place 'Stop All' to stop the game after 30 seconds.



Variable are containers that can store anything, with handy names to identify their purpose/content.

'Green Flag', just above the Stage.

Next grab another Green Flag block from the Control palette, and add a 'Wait Until' block to it. Inside the blank for the Wait Until block place a '__ = __' block from the Operators palette. Now move to Sensing palette and look for 'Slider Sensor value'. Click the drop-down and change it to `gpio2`. To the left you will see a tickbox; tick it and the sensor value for GPIO2 is printed to the top left of the Stage. Currently the sensor is reading 1, or high. Press the button and the sensor value changes to 0, or low. Test complete, untick the Sensor Value tickbox. Now that we know it works, drag the 'gpio2 sensor value' block to the first blank in the '__ = __' block, then type 0 (zero) in the remaining box. Our last block for this section is another Broadcast called Start.

Interaction with sprites

Now let's create more sprites for our game. Draw a planet by clicking the 'New Sprite' icon. The planet has no code attached to it. Now we'll create two sprites to act as obstacles for our ship to avoid. Add a new sprite by clicking the 'Add Sprite' icon – the middle icon in the row below the Stage. We'll choose a robot, but at its default size it is a little large, so right-click on the sprite and choose Resize.

Earlier, we created a Broadcast called Start, and in the Control palette we can see that block now exists. Drag 'When I receive start' to the coding area, followed by a Forever loop. In the Motion palette drag 'Turn clockwise 15 degrees' and 'Move 10 steps'. Place both of these blocks inside the loop.

Our last section of code for the robot starts with another 'When I receive start' block. Under that we add a Forever loop, and inside the loop we add an If statement. This will constantly check to see if we are touching the Rocket sprite – the touching block is the top block in the Sensing palette. If this is True, then our game will play a sound effect, found in the Sound palette, then say "Ouch" for 0.5 seconds. The 'Say' block is found in the Looks palette. If you want to add more obstacles, right-click on the robot sprite and duplicate until you have the required number.

Our next sprite is an enemy to hunt, a Space Dragon. Choose a new sprite and then drag a 'When I receive start' Broadcast from the Command palette, and also grab a Forever loop. Drag the Space Dragon to some part of the

We built a custom controller from spare components, an old slide box and – for that authentic space experience – a launch switch found on eBay.

screen. Go to the Motion palette and look for 'Glide 1 secs to x: y:'. This will be pre-populated with coordinates, but they'll be wrong. To fix this, change to another palette and then back to Motion, and it will update. Drag the 'Glide' block to the loop. Repeat this action four or more times to create a pattern for our enemy to follow. Remember that we had a sound play when an obstacle is hit. Well, the same can happen with our Space Dragon: just add the same code from our Robot sprite.

Return to the Rocket sprite and create another section of code that starts 'When I receive start', using a Forever loop. Now turn on our LEDs by using a Broadcast to turn a pin on and off – for example `gpio17on` and `gpio17off`. Our LEDs are on 17, 27 and 22, so construct a light sequence, remembering to use 'Wait' blocks to control the LEDs' speed.

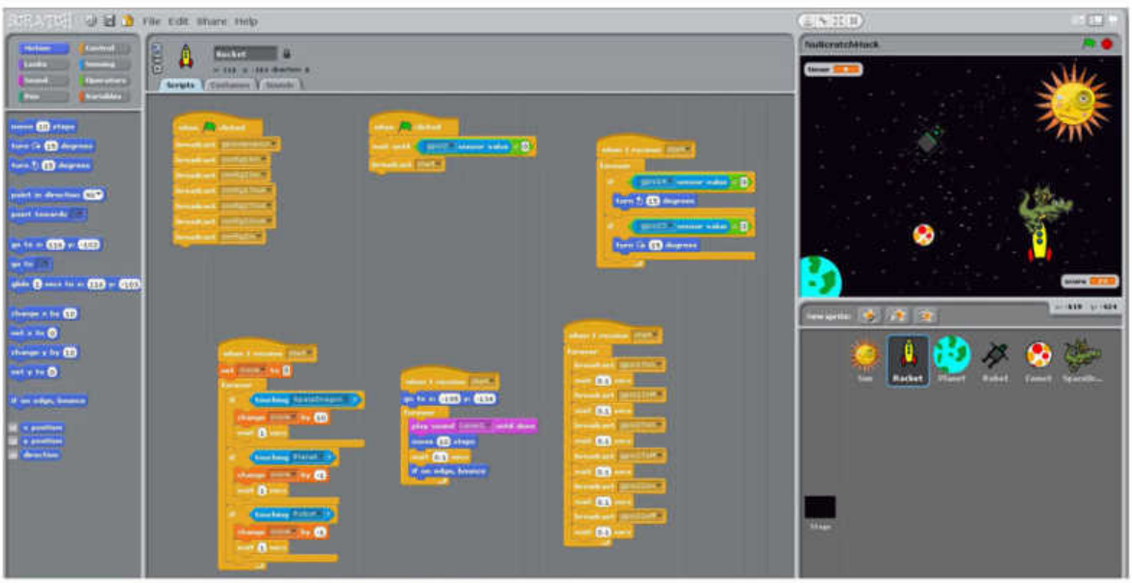
To create controls for the rocket drag another 'When I receive start' block into the coding area, along with a Forever loop and two If statements. Both If statements will be inside the loop, on top of each other. Grab two '__=__' blocks from the Operators palette and place one in each If statement. Next grab a gpio14 sensor value from Sensing and place it in the first blank of '__=__' and type 0 in the other. Repeat for the second If statement but change gpio14 to gpio23.

To finish our Rocket code, add one more 'When I receive start' block. Use 'Go to x: y:' to set a starting position for the rocket, say the bottom left of the screen (x:0, y:0). Next, we use a Forever loop to play a laser sound, then move the ship forward 10 steps before waiting for 0.1 seconds. We add 'If on edge, bounce' to stop the Rocket getting lost.

With all of the code complete as shown below, save your work and blast off for an adventure!

Quick tip

Scratch is very intuitive but often we get a little stuck. If your code breaks, pull it apart and rebuild it and test it section by section. You can also quickly duplicate code by right clicking on the code and selecting Duplicate.



Our project is a crazy space game where our rocket must attack the mysterious Space Dragon while also avoiding a deadly Space Robot and a Comet.

Python 3: Build your first robot

Find out how to build a simple but elegant, budget-busting custom robot perfect for scaring the cat.

Robotics is an exciting way to introduce people to programming but it can also be a little difficult sometimes for newcomers to get to grips with as well as being expensive. Enabling anyone to create an easy to build and cost-effective robot is a significant step in their learning. So in this tutorial we shall build our own robot and create a Python 3 library that enables anyone to control it. For this project you will need: any model of Raspberry Pi; Raspbian (**www.raspberrypi.org/downloads**), a Wi-Fi dongle and Pi connected to your home router; a USB battery pack, a robot chassis kit (**http://bit.ly/LXF203_Robot_kit**), an L298N motor controller (**http://bit.ly/LXF203_L298N**); four AA batteries and some Blu-tack.

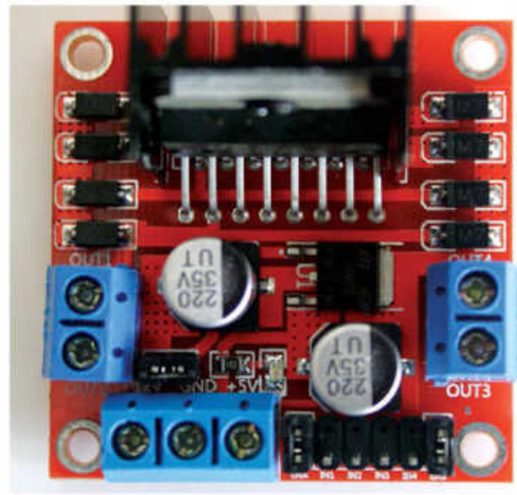
Building a robot chassis is a great activity and the kit (mentioned above) comes with everything that you need to get started. You will need to solder the red and black wires to the motor terminals, if you can't solder then now is a great time to learn from a friend or a local hackspace.

With the chassis built, we now focus on the motor controller which is an L298N H bridge controller. An H bridge enables a motor to go forwards and backwards. Our L298N has two outputs for our motors, the left side is served by OUT1 and 2, the right by OUT3 and 4. Connect the wires from your motors to these terminals and ensure they are secure. Our AA battery pack connects to +12V and GND terminal. We also need to connect one of the GND from our Raspberry Pi to the L298N GND terminal. On the L298N we can see four pins marked IN1 to IN4. These are inputs that we use to connect the L298N to our Raspberry Pi GPIO (General Purpose Input Output) pins.

By turning a GPIO pin on or off we can trigger the input pins accordingly and control the motor direction. We connected our inputs to the following GPIO pins: IN1 to 17, IN2 to 22, IN3 to 18 and IN4 to 23. We used the Broadcom pin mapping, a standard set by the Raspberry Pi Foundation. A great reference for the GPIO is **http://pi.gadgetoid.com/pinout** which explains Broadcom pin mapping.

Software setup

Boot your Raspberry Pi to the desktop and open a terminal, you can find the icon in the menu bar at the top left corner of the screen. In the LXTerminal type the following and press Enter to run: `$ sudo raspi-config`. Using the arrow keys navigate to Advanced Options and press Enter. In the



The L298N board is packed full of components. The screw terminals enable connections between the batteries, Raspberry Pi and motors. There are more images in our repository (http://bit.ly/LXF203_Robot).

Quick tip

Securing your components to the chassis is important, otherwise your Pi will be dragged along by the robot. We used Blu-tack but for a long-term project cable ties are better.

Remote connection

We set up an SSH server on our Raspberry Pi at the start of this project, so now let's use it to remotely control our robot. On Linux we can run the `ssh` command from the terminal. To SSH into our robot we need to know its IP address, which we wrote down earlier, and the name of the user; typically pi for a Raspberry Pi. We then type the following into a terminal to proceed: `$ ssh pi@IP ADDRESS`.

Replace `IP ADDRESS` with your Pi's IP. You will be prompted for the Pi password, which is typically raspberry, and once logged in any command entered will output on your Pi. Navigate to the directory where you saved the **robot.py** and **test.py** files. To run the test code in the terminal type `sudo python3 test.py` .Press Enter and the robot will come to life and perform the test sequence. Great it works!

But lets open an interactive Python 3 session and live code the robot with `$ sudo python3 -i`. We can now import the robot library and run the same functions as per the *test.py* file. To return to the terminal just press CTRL+d.

Soldering

In this project we bought a robot chassis kit from eBay that included two DC motors. These motors come assembled but require soldering two wires to the terminals for power. Soldering is an essential maker skill and it is really easy to learn, though adult supervision is essential for our younger would-be solders out there. There are many YouTube tutorial videos, but the best is from Carrie Anne Philbin (**http://bit.ly/LXF203_Solder**).

Soldering irons sets can be bought for around £10, but a good example is the Antex XS25 for around £25, which is a great starter to intermediate soldering iron. Soldering should be undertaken in a spacious, well-ventilated room with a clear workspace. Soldering is great fun and your local hackspace/LUG can help you to learn in a safe manner.

Advanced menu navigate to the SSH Server option, press Enter and in the new screen choose to Enable the SSH server. Exit from the menus and reboot your Raspberry Pi. Reboot back to the desktop and open another LXTerminal and type the following for your IP address and write the address down: `$ hostname -I` .

In the same terminal type the following to launch the Python 3 editor with superuser powers: `$ sudo idle3 &` . We'll start our code by importing two libraries, the first enables our code to talk to the GPIO pins on our Raspberry Pi while the second provides the time library:

```
import RPi.GPIO as GPIO
import time
```

When using the GPIO pins we will refer to them using their Broadcom pin numbering and we must, in turn, configure our code to use those numbers with GPIO.setmode(GPIO.BCM). Rather than refer to each pin throughout our code we shall create four variables to store the GPIO pin connected to each of the inputs on the L298N:

```
fwdleft = 17
fwdright = 18
revleft = 22
revright = 23
```

In order to use each GPIO pin we need to instruct the code what each pin will be: an input or output. As we will be sending current from the GPIO pins they will be an output. So using a list, known in other languages as an array, and a for loop, we shall iterate over each item in the list, which are our variables, and configure each GPIO pin as follows.

```
motors = [fwdleft,fwdright,revleft,revright]
for item in motors:
        GPIO.setup(item, GPIO.OUT)
```

Driving our robot

We now create four functions that will handle driving our motors in a particular direction. Each of the functions will take an argument, a duration of time that's expressed as an integer or a float:

```
def forward(i):
        GPIO.output(fwdright, True)
        GPIO.output(fwdleft, True)
        time.sleep(i)
        GPIO.output(fwdright, False)
        GPIO.output(fwdleft, False)
```

Our first function, `forward(i)` , will turn on fwdright and fwdleft pins and then wait for the value of `i` , our argument before turning the motors off. On to our second function:

```
def right(i):
        GPIO.output(revright, True)
        GPIO.output(fwdleft, True)
        time.sleep(i)
        GPIO.output(revright, False)
        GPIO.output(fwdleft, False)
```

Our second function, `right(i)` , spins our robot on the spot in a clockwise direction for the duration provided as the argument `(i)` . To turn right we set the right motor to reverse and the left motor to forwards, wait for the user defined number of seconds and then turn off the motors.

For our left and reverse functions you can refer to the full code at **http://bit.ly/LXF203_Robot**.

The last section of code is a try and except test:

```
try:
        print("R E A D Y")
except KeyboardInterrupt:
        print("E X I T")
        GPIO.cleanup()
```

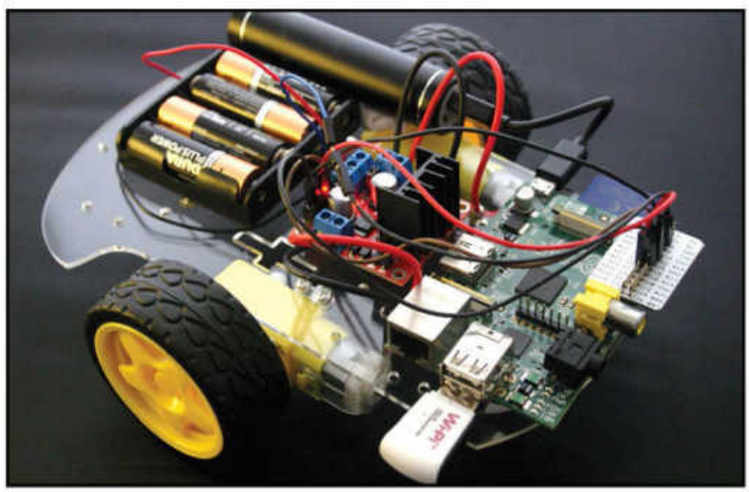
This will print R E A D Y when the code is executed, but if we press CTRL+c it will print E X I T and then clean up the GPIO pins ready for use by another project:

Save your code as **robot.py** but we won't be running the code, rather we will now create a new file and save it as **test.py** in the same directory as **robot.py**.

Next, we'll import our **robot.py** code and use the functions inside of it to control our robot.

```
import robot
robot.forward(1)
robot.right(2)
robot.left(2)
robot.reverse(1)
```

Save the code and click Run > Run Module to test. Remember to pick up the robot before pressing Enter or you will have to chase after it!



Our finished robot is simple yet elegant. Its utilitarian design enables easy access to all of the components for any last minute tweaks or fixes.

Bring R2-D2 to life

We use a Pi Zero to set the little bleep-blooper wheeling around.

Gear...

- For this project you will need:
- A Raspberry Pi Zero, or another Raspberry Pi (A+,B+ or Pi 2)
- The latest version of the Raspbian operating system
- Wi-Fi connection for your Raspberry Pi
- 2x Micro gear metal motors
- Wheels for R2
- Explorer pHAT board
- An LED
- A 220 Ohm resistor (RED-RED-BROWN)
- Male-to-female jumper cables
- Male-to-males jumper cables
- 2x Terminal block with screw terminals
- Hot glue gun
- Soldering Equipment

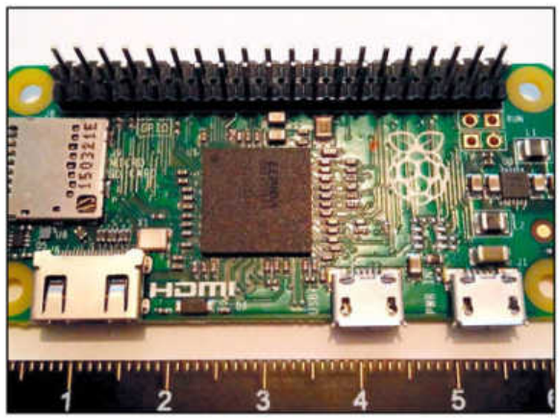
The majority of these components can be found at Pimoroni and The Pi Hut.

What Star Wars fan hasn't dreamed of owning their own droid? The droids C-3PO and R2-D2 serve as the narrators to an epic story encompassing good versus evil. C-3PO is a protocol droid who lives to serve his master, while R2-D2 was first seen (chronologically) in Episode 1: The Phantom Menace, and saved Queen Amidala and her party as they left the planet of Naboo. R2-D2 is an Astromech droid, a robotic engineer that can fix any problem.

In this tutorial we'll construct our own R2-D2 robot by hacking a store-bought toy with the latest Raspberry Pi, the Raspberry Pi Zero.

The Raspberry Pi Zero is an ultra-low-cost Raspberry Pi, and retailing at $5 (£4, around AU$7) this board provides a full computer experience for very little money. Coming with 512MB of RAM, 1GHz ARM CPU and able to output 1080p video, it's been flying off the shelves, and the Raspberry Pi Foundation is working to restock.

We'll also be using the latest Explorer pHAT board from Pimoroni. This board is a cheap and easy to use motor controller and experiment board, and will provide the functionality for our R2 unit.



Step 1 Solder the GPIO

Before we can use the Raspberry Pi Zero, we will need to solder the GPIO, a series of 40 pins, onto the board.

These pins enable our Explorer pHAT add-on board to interact with the Raspberry Pi. For the best results use a blob of modelling clay to support the board while you solder. If you can't solder, ask a local hackspace, DIY expert or friends for help. You will also need to solder the header pins for the Explorer pHAT board.

If you're using another type of Raspberry Pi then you can skip this step, as the GPIO comes pre-soldered.

Step 2 Hack R2-D2

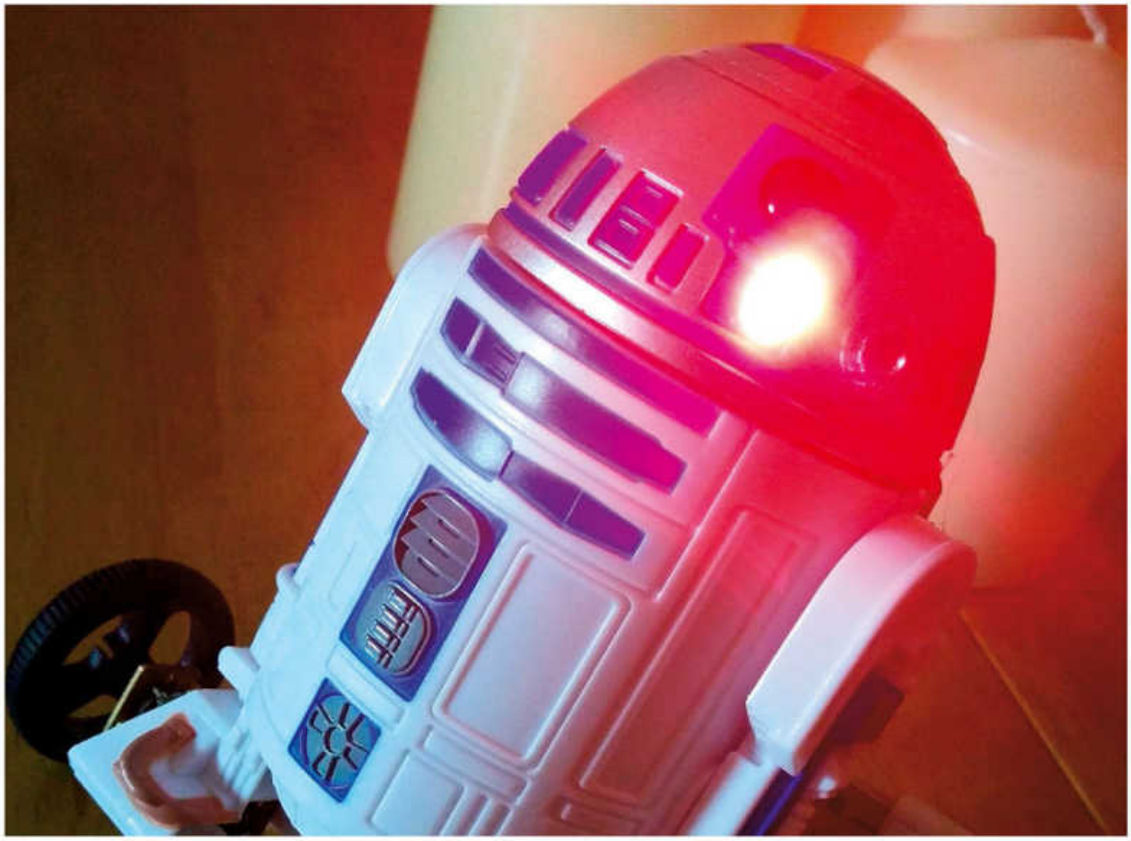
We chose a small R2-D2 toy costing around £20 (around $30, AU$42). We started by removing all of the internal components, to see what room we had to hack. It was quite tight even for the Raspberry Pi Zero, so we chose to house the major components on the rear of R2-D2.

We removed R2-D2's legs and measured the size of our motors on the feet. Using a Dremel tool we carefully cut and shaped a space for the motors on each foot.

Before attaching the motors we soldered wires to the terminals for later use, so give yourself plenty of spare wire, and attach the wires for each motor to the Motor 1 and 2 header on Explorer pHAT. Use hot glue to secure the motors in place on the feet.

Step 3 Add an LED

For added authenticity we drilled a hole in R2-D2's red eye socket, and replaced the plastic with an actual working red

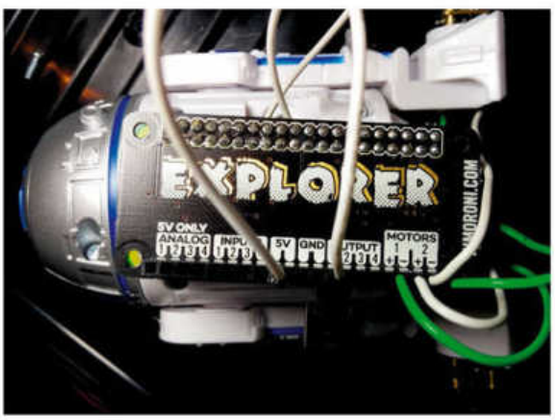


R2 may have a nice bright LED, but he doesn't project any holograms. Sadly.

LED. You might want to use a slightly smaller one.

Inside R2-D2's head we used a terminal block, commonly used in electrics to secure wires without soldering, and secured the legs of the LED into each hole.

The long leg of the LED will receive 5V power from the Explorer pHAT via a 220-ohm resistor screwed into the other end of the block and linked to the Explorer pHat via a female to male jumper cable. The short leg will connect to Output 1 of the Explorer pHAT.



Step 4 Power up

Attach the Explorer pHAT to the Raspberry Pi, then connect your peripherals before powering up the Raspberry Pi.

Once the Raspberry Pi has booted you'll need to have an Internet connection before proceeding. Open a terminal (its icon is a dark screen in the top left of your desktop), and type the following to install the Explorer pHAT software:

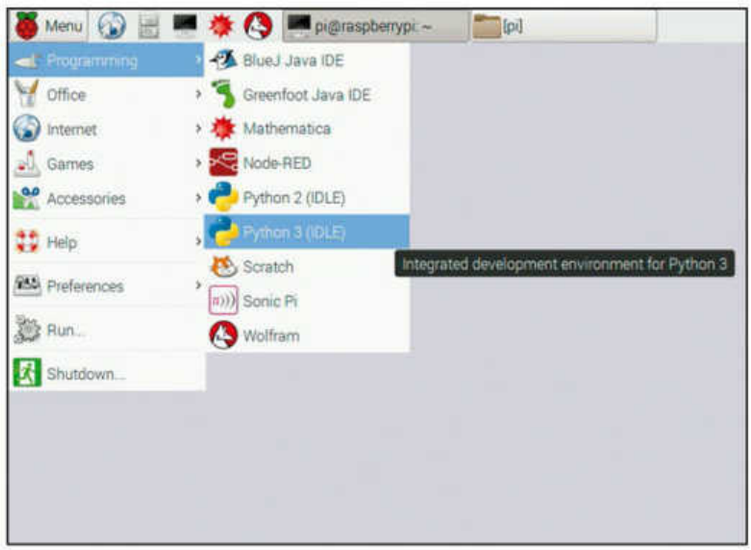
```
curl get.pimoroni.com/i2c | bash
```

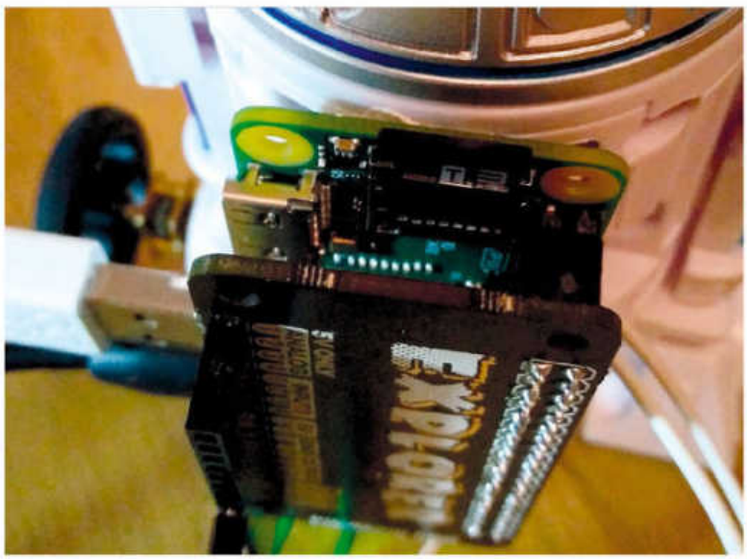
You'll be asked a series of questions, to which you can answer yes, as it's quite safe in this instance. After a few moments the software will be installed, and to ensure that it has been done correctly now is a good time to reboot your Raspberry Pi. When done, return to the desktop.

Step 5 Start coding

To code our R2 unit we'll be using Python, a really easy to use language that is well suited to the Pi, and which the majority of the Raspberry Pi community uses as their primary language. To code in Python we need to open the Python3 application, which you can find in the Programming menu.

When Python3 is open, click on File > New Window to open a new blank document. Now save the project as R2P10.py before proceeding. Remember to regularly save your work – this is a good habit to get into, as it will limit any accidental deletions.





Step 6 Import code

Our first three lines of Python code are imports – these bring in external libraries of code for our project to use.

```
import explorerhat
from time import sleep
from random import randint
```

Firstly we import the library for Explorer pHAT – this will enable our project to interface and use the board.

The next two imports are used to import one aspect of each library, as we don't need to pull the whole thing into our project. It'd be a bit wasteful. From the time library we import the sleep function, used to control the pace of our code. From random we import a random integer generator, used later.

Step 7 More coding

We now move to the main body of code:

```
try:
        while True:
                explorerhat.output.one.blink(0.5,0.5)
                duration = randint(1,5)
```

Here we create a method to test our code, and inside it we use a loop that will run forever, while True; this loop will run the code to set output 1 of our Explorer pHAT so it blinks every half a second. We now create a variable, a container that can store any type of data – in this case we're using it to store a random number, an integer, between 1 and 5.

The indentation of code is important, and Python 3 will help you as you type.

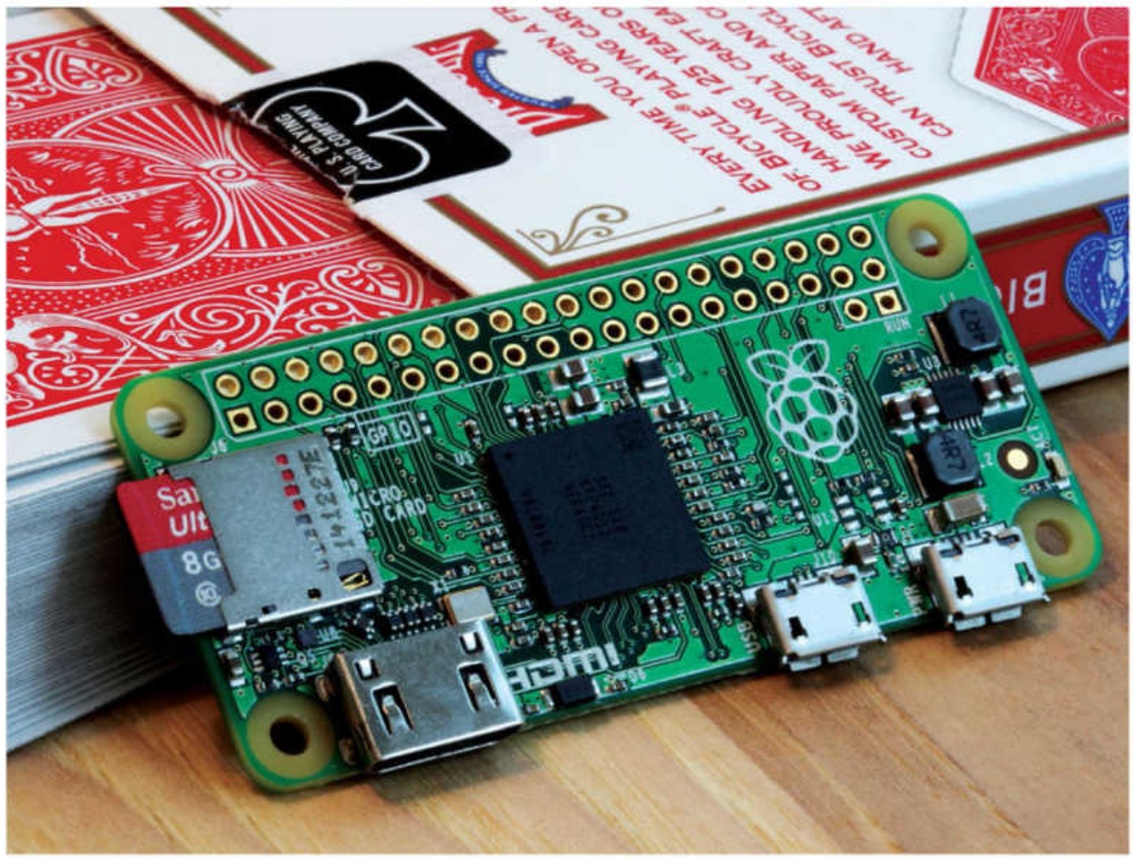
Step 8 Add movemen

For our next section of code, we will turn on the motors and control R2-D2:

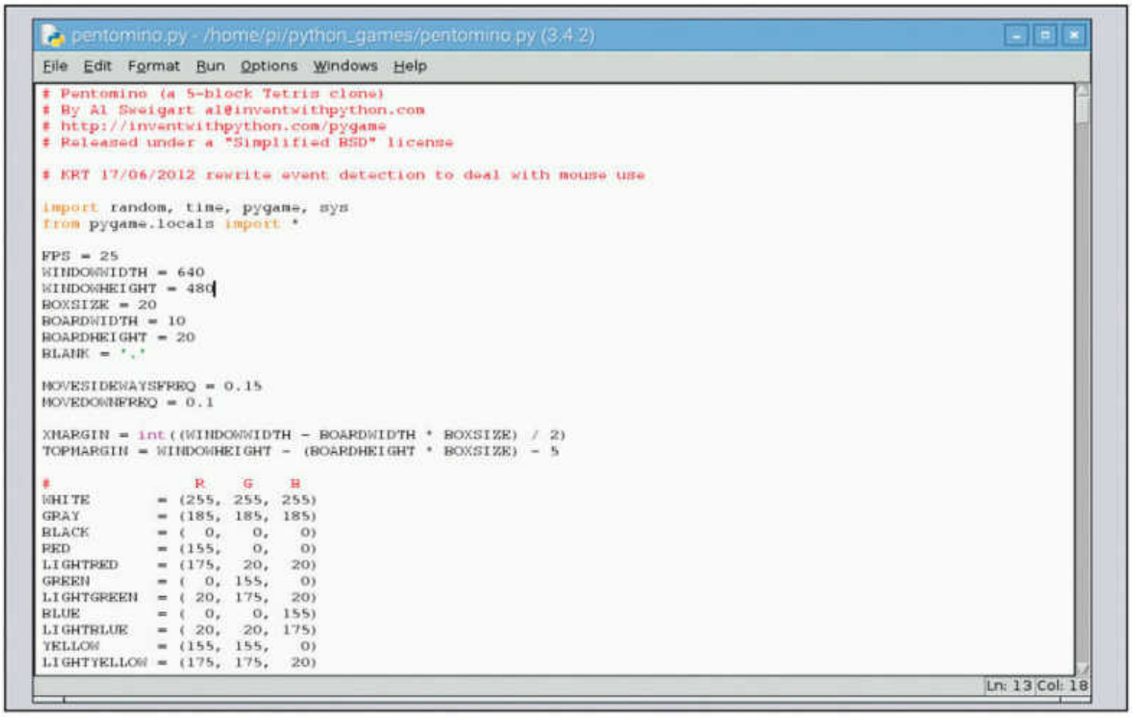
```
explorerhat.motor.one.forward(20)
explorerhat.motor.two.forward(20)
sleep (duration)
explorerhat.motor.one.backward(20)
explorerhat.motor.two.forward(20)
sleep (duration)
```

We start by turning both of our motors on and forward at 20% power, a nice glide for R2-D2 to move along on. If your motor is running in the wrong direction, swap the wires for that motor so that they're reversed in the Explorer pHAT.

With the motors on we use a delay, sleep, to prevent the motors stopping. We use the random number stored in the variable, duration, to control how long R2 will run for. Next we run one motor backwards to enable R2 to turn on the spot; this is also controlled for a set duration.



The Pi Zero. So small you could hide it in a pack of cards. But why would you when it looks this cool?



New to Python? Jessie comes loaded with tons of example code, so check it out.

Step 9 Test your code

This next section of code exists outside of the while True loop:

```
except KeyboardInterrupt:
        explorerhat.motor.stop()
        explorerhat.output.one.off()
```

We started the code by using a test, try, now we need to add an exception – in this case when we press CTRL+C to stop the code running. When this occurs we tell the motors to stop, and for the output to turn off, stopping R2-D2's eye blinking.

Save your code and prepare to test your R2 unit! For the best results hold on to the R2-D2 unit, otherwise it will run away.

Step 10 You're in control!

With our code complete you can now run the code by clicking on Run > Run Module in the menu. R2-D2 should move forward for a few seconds, then turn left. Then he'll move forward again. Then turn left again.

This code will then repeat until you press CTRL+C. For a truly portable solution you can use a USB battery pack, such as those used to charge your mobile devices, to power the Raspberry Pi. You can also control R2 over a Wi-Fi connection using technologies such as SSH and VNC.

And there you have it – your very own R2-D2 unit, ready to respond to your commands! Make sure you keep him pure and away from the dark side, now...

You might want to attach your Pi Zero slightly more securely...

Astro Pi: Primer

Get to grips with the Sense HAT board that's made its way to the International Space Station with ESA astronaut, Tim Peake.

The Sense HAT is a remarkable platform with sensors covering temperature, humidity and pressure. It also has an accelerometer gyroscope and compass which can be used for positional data. To top it off the board has an 8x8 LED matrix and joystick. In this project we'll be using this marvellous device to measure temperature, humidity, pressure and to draw animations on that matrix.

As well as the Sense HAT, you'll need either a Raspberry Pi 2, A+ or B+, a copy of Raspbian and an internet connection. All of the code for this project can be found at **http://bit.ly/LXF204AstroPi**. Installing the Sense HAT on top of your powered-down Pi is really easy and then you'll need to connect all of the cables to your Pi and boot to the Raspbian desktop. To use the Sense HAT we'll need to install its software. To do this open a *LXTerminal* (there's an icon for this in the top left of the screen, it resembles a monitor), and in the *LXTerminal* enter the following:

```
$ sudo apt-get update
$ sudo apt-get install sense-hat
$ sudo pip-3.2 install pillow
```

Here we're ensuring that the list of available software is up to date, then we install the Sense HAT software. We use the *pip* package manager to install pillow, a fork of the Python Imaging Library (PIL). We'll need to reboot after installation and return to the Raspbian desktop. Open *LXTerminal* again and type the following to launch IDLE for Python 3.

```
$ sudo idle3 &
```

With IDLE3 open click on File > New to create a new blank document. Best practice is to save your work now, so click on File > Save and save your work as **sensors.py**. Let's start building the project, by importing the libraries:

```
import pygame
from pygame.locals import *
from sense_hat import SenseHat
import time
```

First, we're importing the `pygame` library followed by the `pygame.locals` library. We'll use the pygame.locals library when detecting the joystick being used. We import the `SenseHAT` library to be able to use HAT board and, finally, we import the `time` function so that we can control the pace of our project. We create a variable called `sense` with `sense = SenseHat()` and store the `SenseHat` function, reducing typing strokes and we initialise the pygame library ready for use with `pygame.init()`.

In the next section of code we create a series of functions that will handle actions in the project. We start by displaying an image onscreen:

```
def image():
    pygame.display.set_caption("Linux Format presents...")
    picture = pygame.image.load("image.png")
    screen = pygame.display.set_mode((688,361))
    screen.blit(picture,(0,0))
    pygame.display.flip()
```

We call the function `image` and it creates a window with a title. We then create a variable called `picture` and use it to store an image ready for use. Next, we create another variable called `screen` and set the screen size according to the image which we will load. We then `blit` the image into memory which rapidly updates the contents of memory with the image data. Last, we update the screen using the `pygame.display.flip()` function.

Temp and joystick functions

Our next function handles joystick input. We use an if...elseif conditional statement to control the actions for four events, pressing up, down, left and right:

```
def joystick(event):
    if event.key == pygame.K_DOWN:
        invader()
    elif event.key == pygame.K_UP:
        pressure()
    elif event.key == pygame.K_LEFT:
        temperature()
    elif event.key == pygame.K_RIGHT:
        humidity()
```

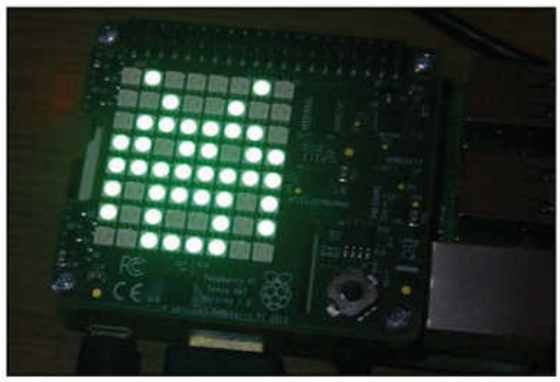
When a condition becomes true the corresponding function is called. Next, we create a function called `temperature` and this function reads the current temperature and stores it as a variable called `temp`:

```
def temperature():
    temp = round(sense.get_temperature(),1)
    if temp < 20:
        sense.show_message("The temperature is %s C" %
temp, text_colour=[0,0,255])
    elif temp > 20 and temp < 30:
        sense.show_message("The temperature is %s C" %
temp, text_colour=[0,255,0])
    else:
        sense.show_message("The temperature is %s C" %
temp, text_colour=[255,0,0])
```

COMPATIBILITY A+/B+ & Pi2

Quick tip

Fancy changing the colour of your text but not sure how to mix RGB colour values? You can find these values in many image-editing applications,eg Gimp. Experiment with different colours and make note of their values and try them out on your Sense HAT.



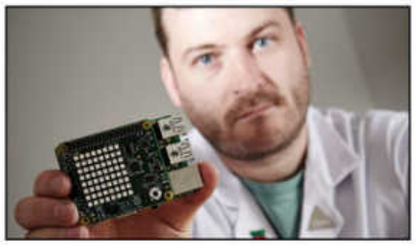
The Sense HAT fits neatly on the Pi and leaves plenty of space to attach the new Pi display and camera.

The Astro Pi Project

Astro Pi is a joint project created by the Raspberry Pi Foundation and the European Space Agency (ESA). The project created a hardware sensor platform, Sense HAT, that has been sent to the International Space Station (ISS), where ESA astronaut Tim Peake will run a series of experiments coded in Python by school children. The projects were created as part of a competition for primary and secondary schools across the United Kingdom.

The code was be carefully checked for bugs inline with an incredibly exhaustive hardware checklist to ensure that both the HAT and the Pi were flight-ready for the trip to the ISS.

Code written by the winners will be run by Tim Peake while in orbit aboard the ISS and the results will be shared with everyone. This project enables children to replicate the experiments on Earth and compare their results with those from the ISS. The project has generated a lot of interest in the space community most notably from NASA, which is keenly viewing the results of the project with an aim to replicate it for a future competition for children around the world. Potentially, Astro Pi's promotion of STEM (Science Technology Engineering Maths) via the Pi may just inspire the next generation of coders, engineers, mathematicians and scientists.



Dave Honess has been working on the Astro Pi project for over a year and has liaised with the European Space Agency to certify the project fit for flight.

You can see that we use `round()` to round the temperature value to one decimal place giving us a precise value. Next, we compare the value of temp using an if...elseif... else conditional statement so that when the temperature is below 20°C the text is scrolled with a blue colour. If the temp is greater than 20°C and less than 30°C, it is green, and for everything else we get red text. For each condition we write the text to the LED matrix using the `show_message` function. We incorporate the value of temp in the text using `%s` this instructs Python to replace this value with a variable and we instruct it to use the temp variable using `% temp` . We can also change the text colour using RGB (Red, Green Blue) colour values.

Humidity and pressure functions

Next, we create a function to measure humidity. This uses a similar structure for storing and displaying the information as we used for the temperature function:

```
def humidity():
    humid = round(sense.get_humidity(),1)
    sense.show_message("Humidity: %s %%" % humid)
```

We reuse the structure of our humidity function to create another function to retrieve and display the local air pressure:

```
def pressure():
    pressure = round(sense.get_pressure(),1)
    sense.show_message("Pressure %s Millibars" %
pressure)
```

Our final function is a bit of fun and draws an animated space invader on the LED matrix:

```
def invader():
    for i in range(8):
        x = [0,255,0]
        o = [0,0,0]
        invader = [
            o,x,o,o,o,o,x,o,
            o,o,x,o,o,x,o,o,
            o,x,x,x,x,x,x,o,
            x,x,o,x,x,o,x,x,
            x,x,x,x,x,x,x,x,
            x,o,x,x,x,x,o,x,
            x,o,x,o,o,x,o,x,
            o,o,x,x,x,x,o,o,
        ]
        sense.set_pixels(invader)
        time.sleep(0.5)
```

Called `invader` , this function uses a for loop that iterates eight times. Next, we create two lists which are called `x` and `o` , where `x` is given the value `0,255,0` or green in RGB. For `o` we use `0,0,0` which is no colour, effectively turning off an LED. We create a list called `invader` and store eight rows and eight columns of x and o. Where an x is displayed, the colour green is seen on the matrix. Python is instructed to set the pixels using the invader list as a template. After waiting 0.5 seconds, we alter the configuration of the invader list then reset the pixels to show the changes. Once the final iteration of the loop is complete the screen is cleared. Not all of the code is shown here, but it's in the final code example.

Our next block of code handles setting the LED matrix for low-light mode, effectively ensuring that the bright LEDs are slightly dimmed to save eyesight. We then call the `image()` function to display the instruction screen. Last, we clear the LED matrix before printing a `"READY"` message in green.

```
sense.low_light = True
image()
sense.clear()
sense.show_message("READY",text_colour=[0,255,0])
```

Finally, a loop will constantly look for joystick input and if any input occurs then the event triggers one of the corresponding functions that we created earlier. We also create a quit method that can be called by closing the Instructions window or by pressing Ctrl+c.

With the project complete. Save your work and click on Run > Run Module to run the code. Congratulations you have taken your first steps with Sense HAT and produced a tool to measure multiple forms of inputs.

Executing your code

Running code in IDLE is easy but there comes a time when you really want to make your code run as an executable. The simplest way to do this is to add `#!/usr/bin/env python3` as line 1 of the **sensors.py** project. With that change made, open a *LXTerminal* and navigate to the directory where **sensors.py** is located. To make the file executable we must change its permissions and we do this with: `$ chmod +x sensors.py` . So now we have the file ready to be executed and we do that in the same terminal by typing `$ sudo ./sensors.py` .

AstroCam: Take photos with a Pi

Take one faithful Pi, a Sense Hat and the official Pi camera and create a camera for budding space adventurers.

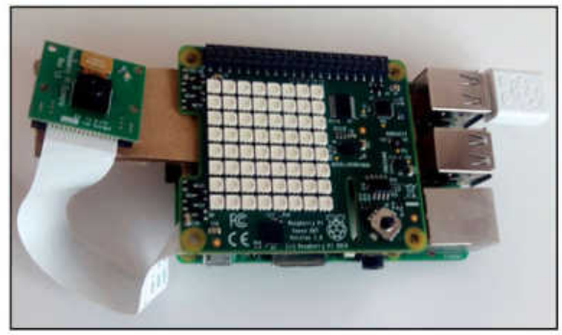
Previously we introduced the Sense Hat which is powering the Astro Pi mission aboard the International Space Station. For this tutorial we'll use it to control a Raspberry Pi camera, and use a joystick to control settings, such as the timer and flash and to trigger taking a picture. For the whole project you will need a Raspberry Pi 2, A+ or B+, the latest release of Raspbian, as well as the HAT and camera. All of the code is at **http://bit.ly/LXF205AstroCam**.

Installation of the camera and Sense Hat is a little tricky, and requires threading the ribbon cable of the camera through the Sense Hat via the slot to the left of the joystick. You'll need to ensure that the blue edge of the connector is facing the Ethernet port of your Pi, then lift up the plastic holder on the Camera port, insert the ribbon, replace the port to retain the ribbon and affix the hat to all 40 GPIO pins.

For this project, we strongly recommend that you install the latest version of Raspbian Jessie as this comes with the Sense Hat library pre-installed. Jessie also enables any user to access the GPIO pins via Python and no longer requires IDLE to be run via `sudo`. You can download the latest version at **www.raspberrypi.org/downloads**.

Boot up your Pi and it will start the desktop environment. Go the main menu and navigate to Programming and then select 'IDLE 3 (Python 3)'. IDLE 3 will open with an interactive shell, which we won't be using, navigate to File > New Window to open a blank document and then save the blank document, File > Save as **astro-cam.py**. We'll start the code by importing the libraries that will power our project.

```
import pygame, time
from pygame.locals import *
from picamera import PiCamera
from sense_hat import SenseHat
```



Our finished project is rather easy to assemble, but it can be integrated into a photo booth with relative ease.

```
sense = SenseHat()
import datetime
```

We're using `pygame` to enable input via the Sense Hat joystick, which replicates the cursor keys and Enter key of a keyboard. `PiCamera` is used to access the camera. We import the Sense Hat library and use a variable to make easy work of its syntax. Last, we import `datetime`, which we use to generate timestamps for file names.

```
pygame.init()
screen = pygame.display.set_mode((640,480))
pygame.display.set_caption('Pygame Test')
```

To use pygame we must initialise it, create a display with a resolution of 640x480 pixels and name the window. Next, we create two functions, the first handles taking a picture.

```
def takepic(timer,toggle):
  for seconds in range(timer):
```

COMPATIBILITY A+/B+ & Pi 2

Quick tip

The Raspberry Pi camera must be protected from touching the Raspberry Pi, especially the GPIO pins. You can purchase a case for the camera from CPC (**http://cpc.farnell.com**) and Pimoroni (**https://shop.pimoroni.com**) for less than £5.

Picamera

The official Raspberry Pi camera is one of the oldest add-ons that has been made for the Raspberry Pi. It uses a 5MP sensor to create 1080p video and high quality still images at 2,592x1,944 pixels. It works with a robust and well written Python library that is very capable.

High speed photography at 90fps is possible by reducing the resolution to 640x480 pixels, and this enables the camera to be used in scientific experiments and sports photography. The camera can also be used as a novel form of input using colour detection built into the Picamera Python library.

Another form of input is OpenCV which enables a computer to 'see', requiring the user to program the computer to look for and interpret objects/faces and act accordingly. This form of input is computationally expensive and reduces the frame rate of the camera to around 10fps but it can be used successfully. The camera uses a fixed focus which can't be altered in software, but it can be hacked using a craft knife to break the glue seal around the lens, but this is not something we recommend. The latest version of the Raspbian operating system, Jessie, comes with the camera pre-configured and ready for use, but if you are using an older Raspbian version, you will need to use *raspi-config* to enable the camera, followed by **pip3-2** to install the Picamera libraries to your Raspberry Pi.

```
    sense.show_message(str(seconds), text_colour=[255,0,0],
scroll_speed=0.05)
    time.sleep(1)
```

First, we define the name of the function, in this case `takepic`, we also create two arguments that can be passed to the function. These arguments are the timer value and toggle used to control the flash unit. We use a for loop to iterate for the number of seconds that the timer is set for. It will scroll the timer value across the LED matrix on the Sense Hat before sleeping for one second and repeating the process:

```
  a = str(datetime.datetime.now())
  a = a[0:19]
  flash(toggle)
```

We come out of the for loop but remain in the function and create a variable called `a` which stores the current date and time and converts it to a string. We slice the string by only using characters from positions 0 to 19 (`[0:19]`) of the string, effectively the real date and time. Last, we call the `flash` function with the value of toggle. Still inside the function we set up the camera to take a picture:

```
with PiCamera() as camera:
  temp = round(sense.get_temperature(),2)
  camera.resolution = (800, 600)
  camera.framerate = 24
  camera.start_preview()
  time.sleep(5)
  camera.stop_preview()
  camera.annotate_text = 'This image has a temperature of
%s C' % temp
  time.sleep(0.1)
  camera.capture('/home/pi/'+(a)+'.jpg')
```

Taking pictures

We start by creating a `temp` variable which will contain the current temperature. We use the Sense Hat temp sensor to take a reading, which is rather precise, and round the reading to two decimal places. We fix the resolution of the camera to 800x600 pixels and the framerate to 24 frames per second and open the preview window for five seconds allowing time to frame the shot before the preview window is closed. Next, we annotate the image to include the current temperature, before sleeping for 0.1 seconds, and taking the picture and saving the image to **home** with the current time and date as the file name. Our final function handles the flash.

```
def flash(toggle):
  print(toggle)
  if toggle == 'on':
    sense.clear(255,255,255)
  elif toggle == 'off':
    sense.clear()
```

We touched on this function in the `takepic()` function and we call this function from inside of `takepic()`. The `flash` function has one argument and this controls whether the flash is on or off. If the value of toggle is `'on'` then all of the LEDs are set to full brightness. If the value of toggle is `'off'` the LED matrix is turned off. We now move to the main code (*see* **http://bit.ly/LXF205AstroCam**):

```
try:
  timer = 0
  while True:
    for event in pygame.event.get():
      if event.type == KEYDOWN:
```

We start by using a try...except test which will enable our code to exit gracefully, if needed. In there we set the value of the timer to 0 (`timer = 0`) and use a loop to check for any user input on the joystick. If the joystick is pressed then a key press is detected.

```
        if event.key == pygame.K_UP:
          print('Adding time')
          timer = timer + 5
          sense.show_message(str(timer), text_
colour=[255,0,0])
```

If the key press returns that up has been pressed on the joystick then five seconds is added to the timer and reported to the user via the LED matrix. Likewise, if down is pressed on the joystick then five seconds is removed from the timer.

```
        elif event.key == pygame.K_LEFT:
          print('Add flash')
          sense.show_message('Flash ready', text_
colour=[255,0,0], scroll_speed=0.05)
          toggle = 'on'
```

Here we use pressing left on the joystick to turn on the flash by changing the value stored in the variable toggle to `'on'`. Pressing right on the joystick will turn off the flash. Our final condition triggers taking a picture.

```
          elif event.key == pygame.K_RETURN:
          print("Takepic")
          takepic(timer,toggle)
          flash('off')
```

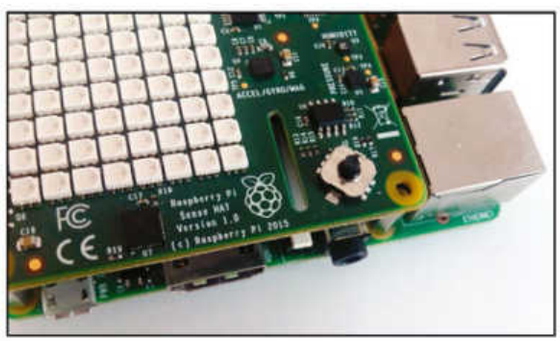
Pressing the joystick in is the same as pressing Enter or Return and triggers the `takepic()` function and passes the values of the timer and the toggle variables as arguments to the function. A picture is taken and then the flash is turned off. Our final section of code is the except part of the try... except test.

```
except KeyboardInterrupt:
  pygame.display.quit()
  pygame.quit()
```

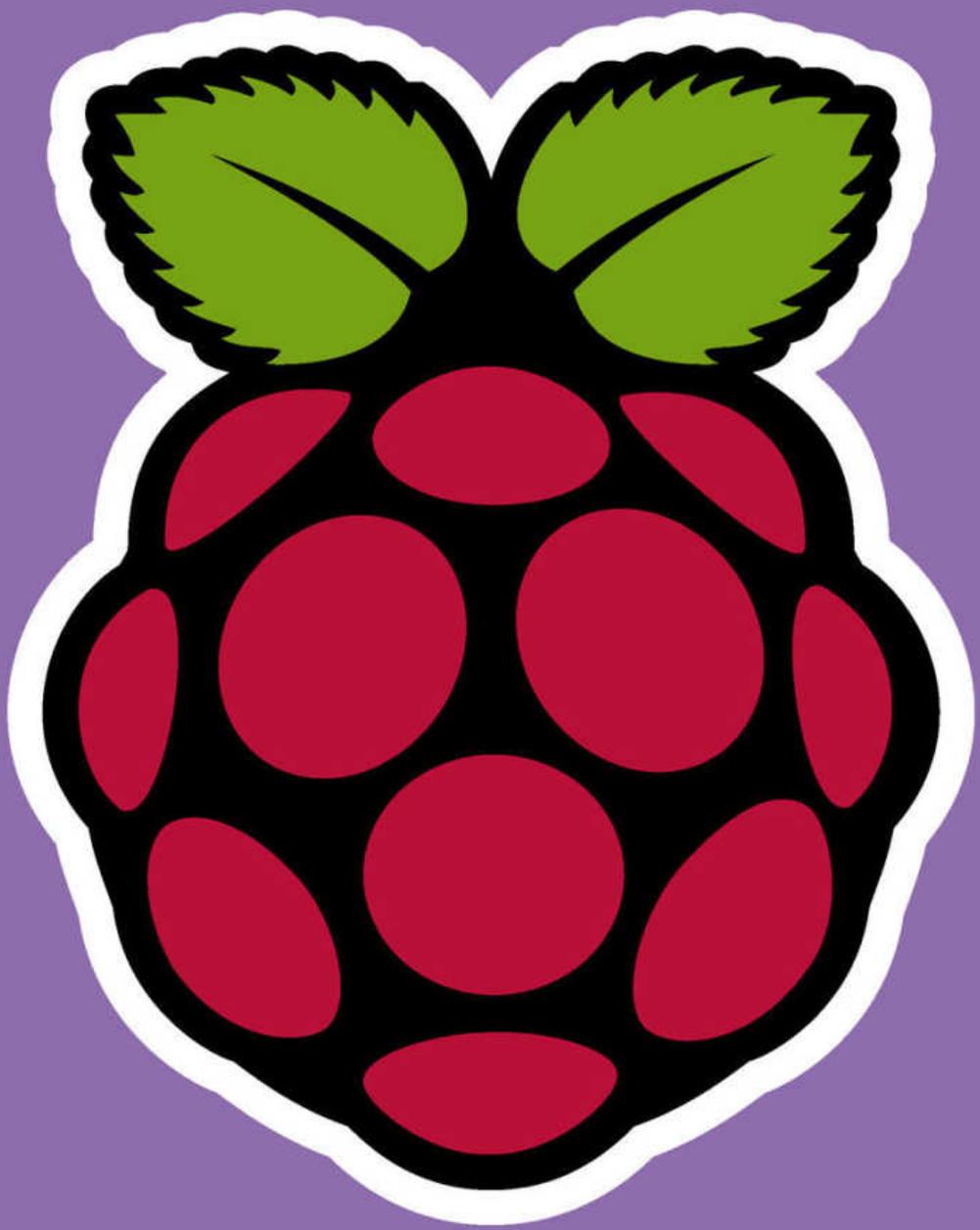
If the user pressed CTRL+c the project will cleanly close any open pygame windows and exit. With all this code complete, save you work and click on Run > Run Module to start taking pictures with your AstroCam!

Scratch update

In the latest version of Scratch, released with the new Jessie image, the Raspberry Pi camera can be used to generate sprites which are controllable using the Scratch palette of commands. The new version of Scratch also supports add-on boards, such as Sense Hat, to enable younger children to recreate projects like AstroCam. Work is still ongoing for this project but the Raspberry Pi Foundation have invested a significant resources to create a version of Scratch which will meet the needs of many different users.



The Sense Hat is a remarkable platform for running scientific experiments, and thanks to a robust Python library we can integrate it into lots of different projects.



Linux skills

Essential knowledge for navigating the world of Pi by yourself.

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.

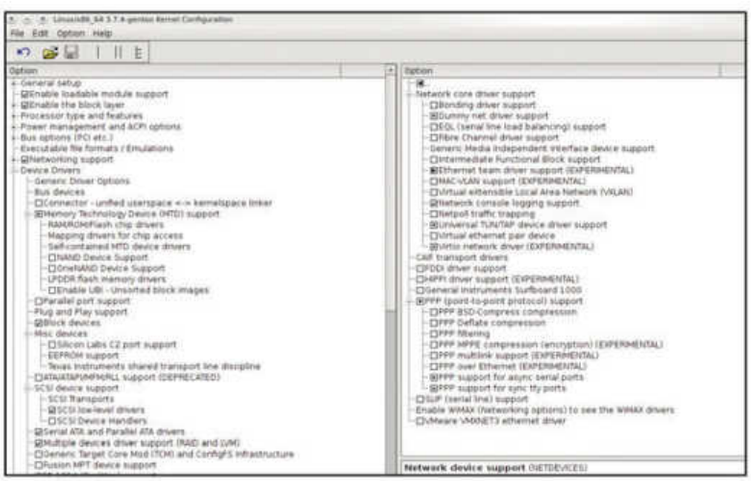
Even more modules

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.

> **"So much hardware is supported, with no need to download drivers"**

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

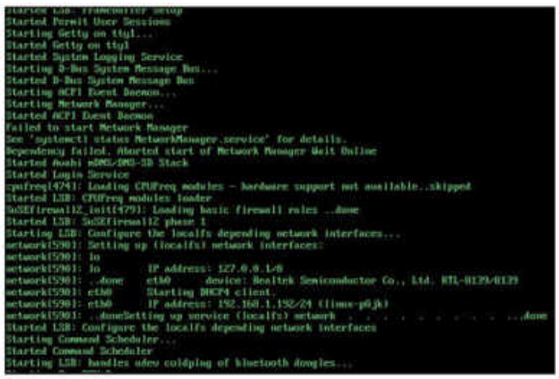
The mysterious sequence of flashes and beeps at startup.

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

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.

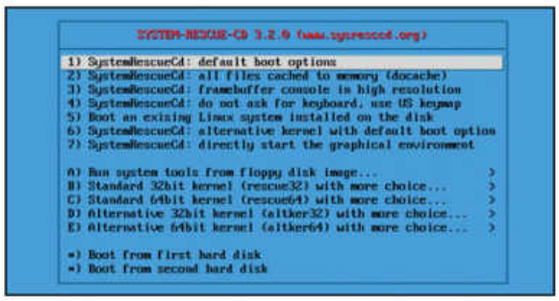
The two were put together and GNU/Linux was born – an operating system using the Linux kernel and the GNU toolset.

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.

notice any difference. There are also moves to replace the 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.

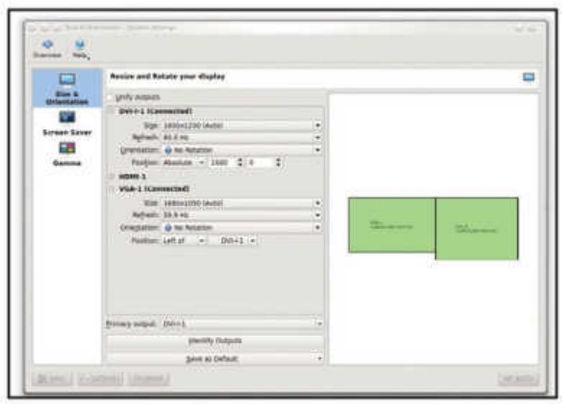
The *X Window System* is the standard basis for providing a graphical interface. While the likes of KDE and Gnome provide the user interface and eye candy, it is through *X* that they communicate with the hardware. For many years, this was a mass of arcane configuration options and required a lengthy 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.



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.

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.

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.

Future

techradar pro

IT INSIGHTS FOR BUSINESS

THE ULTIMATE DESTINATION FOR BUSINESS TECHNOLOGY ADVICE

- Up-to-the-minute tech business news
- In-depth hardware and software reviews
- Analysis of the key issues affecting your business

www.techradarpro.com

twitter.com/techradarpro facebook.com/techradar


T3
HELPING YOU MAKE THE MOST OF LIFE IN TODAY'S CONNECTED WORLD
ONLINE • PRINT • TABLET
AUDIO TECHNICA ATH-MSR7
Escape to a world of high resolution audio
SAMSUNG JS9000
Relax with the latest home entertainment
SONY XPERIA Z3
Putting you in control of your life and home
APPLE WATCH
Your health and fitness upgraded
CARL ZEISS VR
Embrace a new era of virtual reality gaming
LIFE'S BETTER WITH T3
t3.com

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.

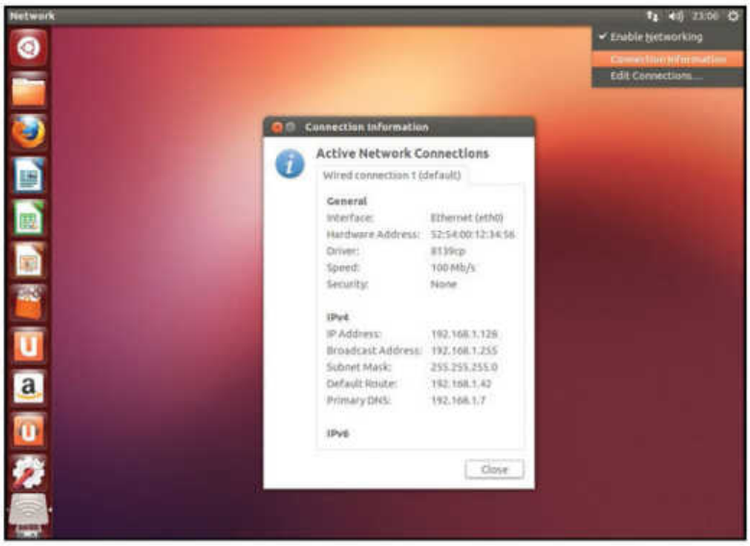
The protocol of kings

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.

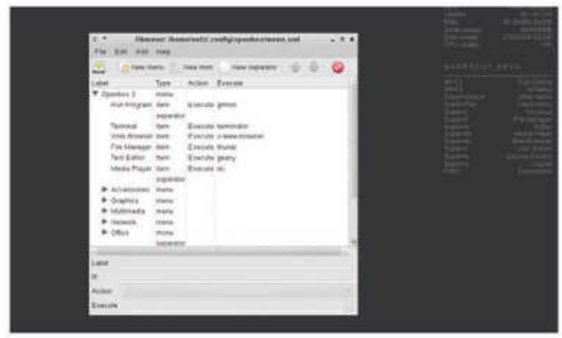
If you consider the kernel to be the lowest level of the system, the highest level is the user interface. Everything else, from the kernel through the drivers and hardware interfaces, is of no use until you can use the computer. This generally means a graphical desktop, and here we come across more layers. 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. 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.



There are also plenty of lightweight window managers, like *OpenBox* running on CrunchBang here.

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

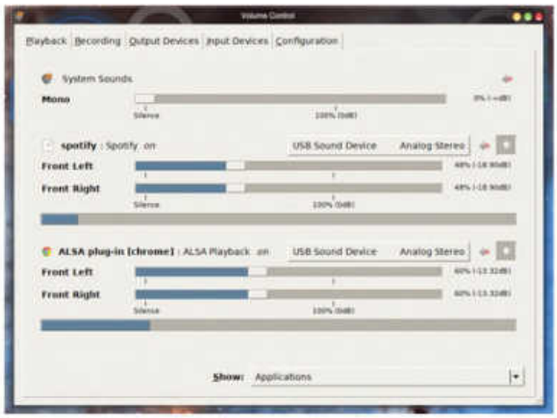
A once thorny subject.

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.

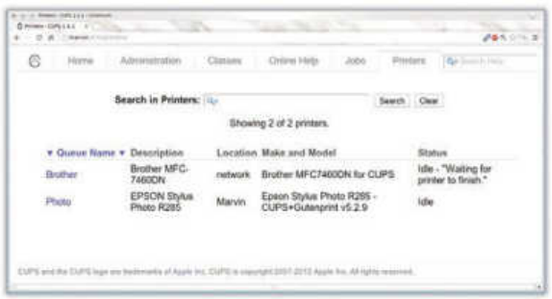
Printing

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

```
-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 4.2K 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.
-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.

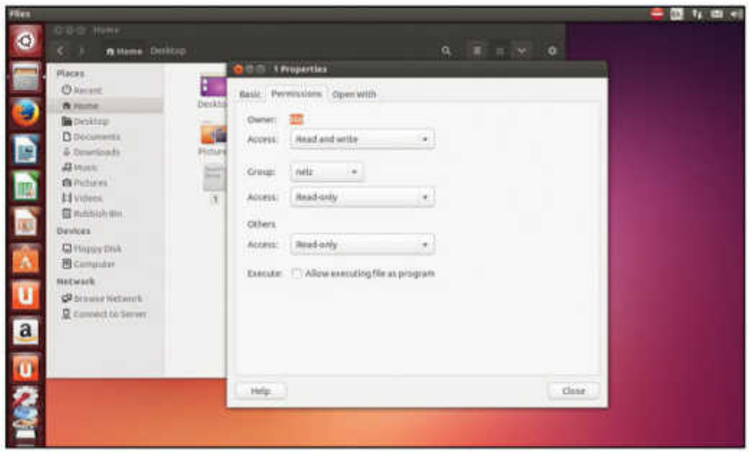
Directory paths can be either relative or absolute. An absolute path starts with **/** so **cd /usr/local** goes to the same place wherever you are starting from. A relative path starts at the current directory, so **cd Documents** goes to the Documents sub-directory of wherever you are, and gives an 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 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.



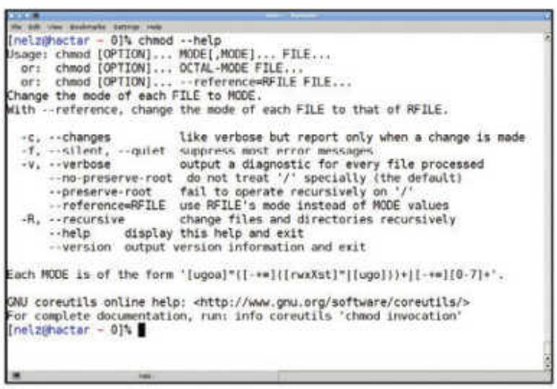
Here is the GUI way of changing file permissions. You would need to do this for each file you wanted to change, and click a separate box for each permission.

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).



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.

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.

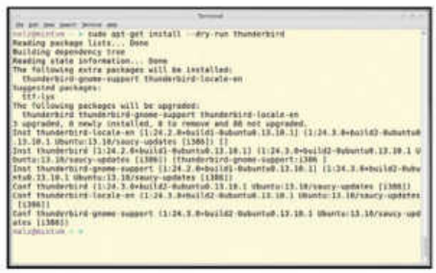
Ubuntu has also introduced the concept of the PPA, or Personal Package Archive, which are small repositories for individual projects. These may each be added individually to your package manager, but be careful about adding any untrusted sources.

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

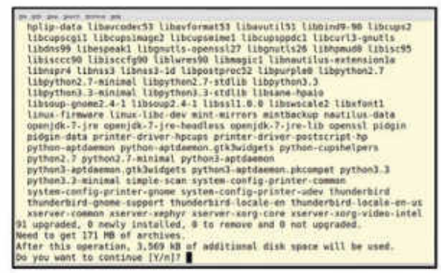


1 Install

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

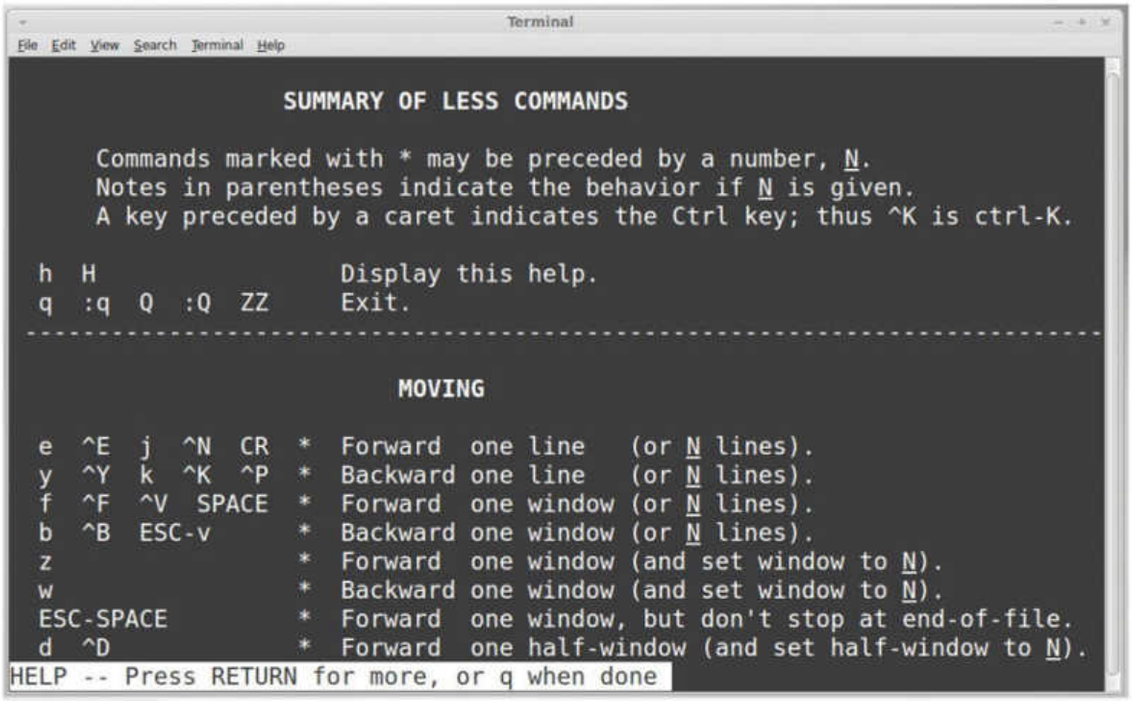
Use **apt-cache search** to find what's available. The **--names-only** option can give a more manageable set of results if you know the program's name. Otherwise let **apt-cache search** go through the descriptions, too, and view the results in **less**. You don't need to use **sudo** as **search** 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**):

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
```

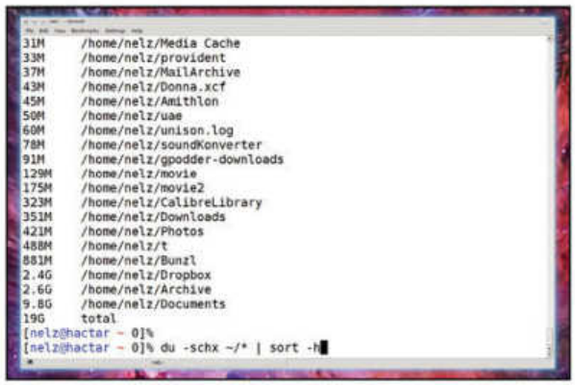
with

```
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.

```
grep -r -I sometext somedir
```

Be careful when you are searching large directory trees, 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.

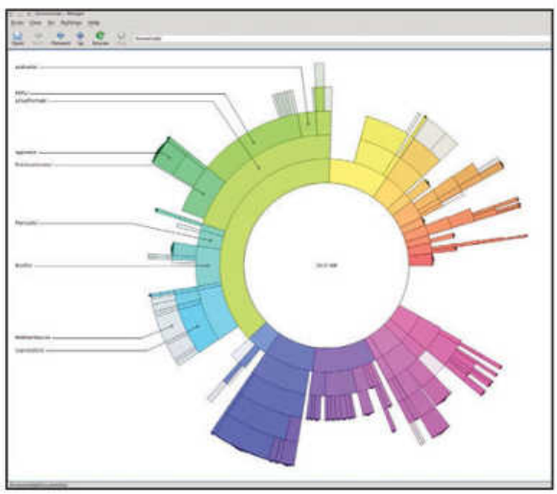
Text is also the preferred way of passing data between many programs, using the pipes we looked at previously. Sometimes you want to pass data straight from one program to the next, but other times you want to modify it first. You could send the text to a file, edit it and then send the new file to the 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
```

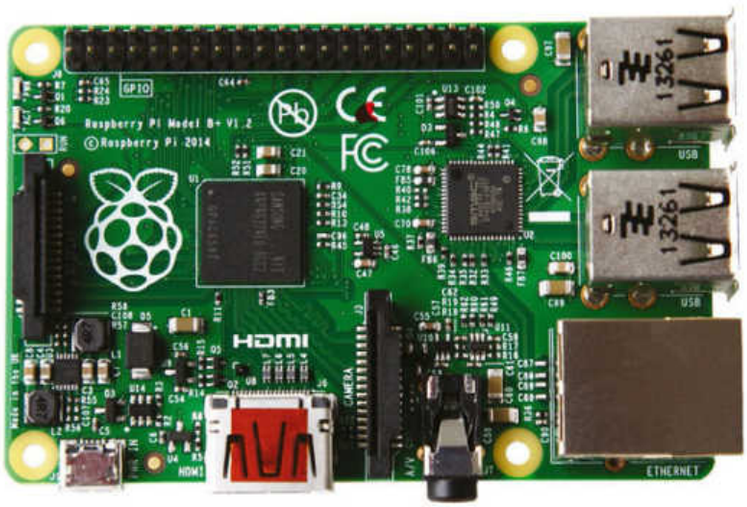
wget Downloads files. The only argument it needs is a URL, although it has a huge range of options that you will not normally need.

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>
```

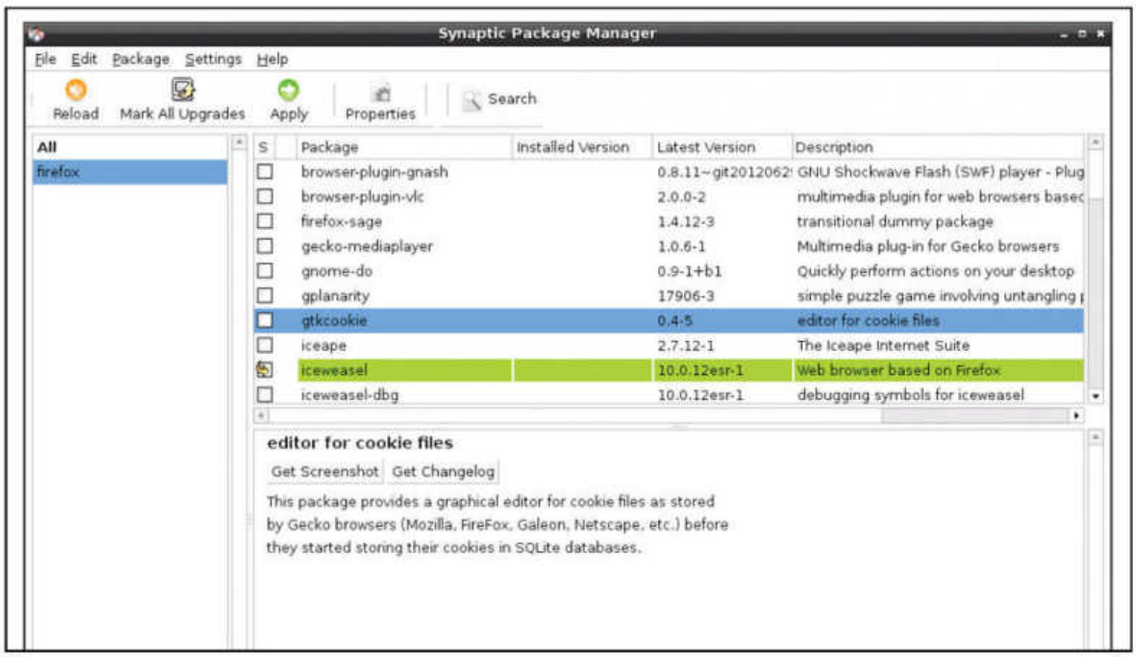
If the filename ends in .tar.bzip2, you need to replace **zxvf** with **xjf**. This should now create a new directory which you need to **cd** into. Hopefully, there'll be a file called **INSTALL**, which you can read with **less INSTALL** This should tell you how to continue with the installation. Usually (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:

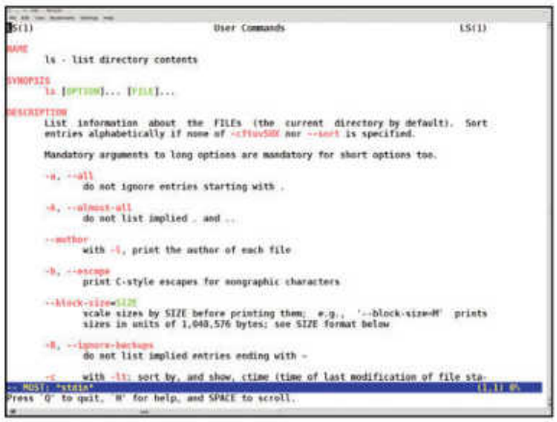
```
firefox --help
```

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



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.

As a normal user, you would normally only use sections 1, 5 and 8 (and possibly 6). If you use the section number with 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
```

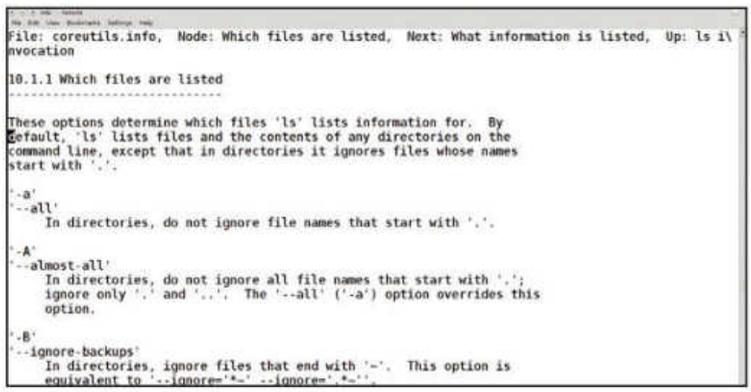
This is a faster way to see what commands do, especially if you want to check more than one:

```
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:



The ls info page goes into more detail and groups options according to their function. Info pages generally provide more detail than man pages.

```
info info
```

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:

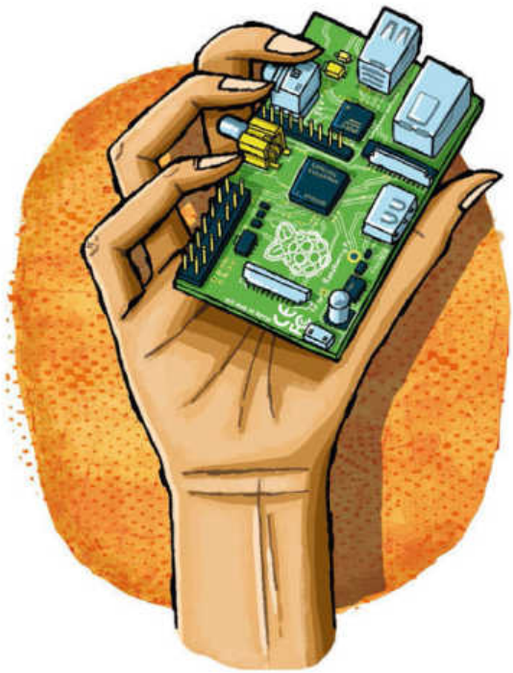
```
man -t somecommand | ps2pdf - somecommand.pdf
```

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:

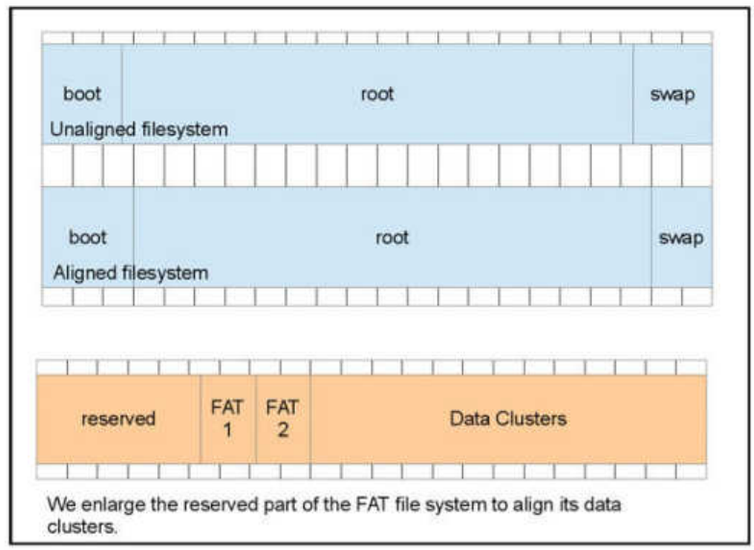
```
# pacman -Syu
# pacman -S base-devel python2 git parted dosfstools
```

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).



Aligning with the erase blocks makes for a happy SD card.

```
$ pacman -Sg base | awk '{print $2}' | grep -v "^\
(linux\|kernel\)" | tr '\n' ' ' > base_packages
$ 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
```

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}
```

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:

```
$ 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).

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**

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.

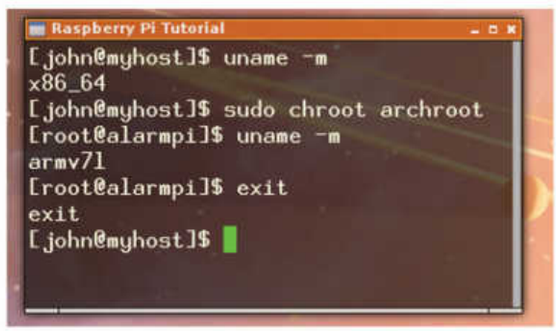
```
$ git clone --depth 1 git://github.com/raspberrypi/linux.git
$ cd linux
$ ssh root@alarmpi zcat /proc/config.gz > .config
$ make -j 8 ARCH=arm CROSS_COMPILE=arm-linux-
gnueabi- menuconfig -k
```

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
```



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.

```
$ 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 **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\x02\x00\x28\x00:\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.

200 BEST-EVER LINUX TIPS

There's always something you don't know. Here are 200 amazing ways to get more from Linux, gleaned from years of navigating its nooks and crannies.

Getting started

Testing and installing Linux distros like a pro.

1 Create a Live distribution with persistent storage
The most popular distros, such as Fedora and Ubuntu, ship with tools that earmark storage space on the live USB disk for saving data that will be available on subsequent reboots.

2 Put multiple live distros on one disk
If you want to test several live distributions you can put them all onto one USB flash drive using either the MultiCD script (which you can find here: **http://multicd.us**) or by using the French MultiBoot LiveUSB tool (**http://liveusb.info/dotclear**).

3 Use an external partitioning tool
While the partitioning tools within the distributions have improved considerably in terms of achieving better control over your disk, it's best to prepare partitions for a Linux installation using third-party tools, such as *Gparted*, which is also installed in the live versions of several distros.

4 Use LVM partitions
One of their biggest advantages to using LVM (Local Volume Manager) is that unlike standard partitions you don't have to estimate partitions at install as you can grow (or shrink) an LVM volume without losing any data.

Get more from the desktop

Be more productive on your favourite desktop.

5 Middle-click to paste
When you highlight some text with your mouse, the text is copied to a special buffer. When you middle-click in a text entry area, a copy of the text that you originally highlighted is pasted into the text entry field.

6 Define keyboard shortcuts
Almost every mainstream desktop allows you to define custom keyboard shortcuts. You'll find the option under the keyboard setting in their respective configuration panels.

7 Touchpad tricks
Move your finger up and down the right side of the touchpad to scroll vertically and tap in the lower-right corner of the touchpad to perform a right-click.

8 Enable workspaces
To enable workspaces in Ubuntu, head to System Settings > Appearance > Behavior and toggle the Enable workspaces option.

9 Install a Dock
On desktops, such as Gnome, cut down the time that it takes to launch your favourite apps by placing them on a Dock, for example the lightweight *Cairo-Dock* which is available in the official repos of most distros.

10 File manager context-menu
The right-click context-menu that is found inside the file manager on most desktop distros are full of useful options that you might have missed, such as the ability to email, compress or restore them to an earlier version.

11 Create Favourites
Place your favourite apps in Ubuntu's Launcher and Gnome's Dash by dragging them from the desktop's respective applications view.

12 Put icons on the desktop
To alter Gnome install the handy *Gnome Tweak Tool* from your distro's repos. Launch the app, head to the Desktop tab and toggle the Icons on Desktop option.

13 Quick Launch menus
Right-click the icons in Ubuntu's Launcher or an app's name in top bar in Gnome to reveal application specific options and actions.

14 Launch commands from the Mint menu
Right-click the Menu applet, choose Configure > Open the menu editor. Then select a sub-menu or create a new one and select 'New Item'. Enter the command in the space that is provided and toggle the launch in the terminal checkbox for CLI apps.

15 Alter power button behaviour
To tweak the setting of the Power button in the GTK-based Cinnamon, head to System Settings > Power Management and use the power button pull-down to select how it responds.

16 Change Panel Layout
To change Cinnamon's default panel layout head to Settings > Panel and use the Panel Layout pull-down menu to select a different style.

17 Add Applets to Panel
Cinnamon ships with several interesting applets that you can add to any panel by right-clicking the panel and selecting 'Add Applets' to the Panel option.

18 Enable compositing
For some bling, enable compositing on Mate by toggling the 'Enable software compositing window manager' option from under Control Center > Windows.

19 Get different widgets on each desktop
To customise the virtual desktops in KDE, right-click the Pager, switch to the Virtual Desktops tab and toggle the option. Now each desktop can have different widgets etc.

20 Run applications as another user
To get an application running as another user (like root) in KDE, right-click the menu icon and select 'Edit Applications', select an existing entry and click 'Copy'. Then navigate to where you want the new entry, click 'New Item', give it any name and click 'Paste'. Switch to the Advanced tab and toggle the 'Run as a different user' option and enter the username of any user.

21 Slideshow wallpaper
Right-click the KDE desktop and click 'Default Desktop Settings'. Use the Wallpaper pull-down menu to select the Slideshow option and point it to a set of images.

22 Enlarge windows horizontally
Xfce users can right-click the Maximise window button to horizontally stretch it across the screen.

Useful keyboard shortcuts

23 Alt+F2 Bring up the Run dialog box.

24 Alt Search through an app's menu via Ubuntu's HUD.

25 Alt+~ Switch between windows on the same app.

26 Alt+Ctrl+ Up/Down/ Right/Left Switch between workspaces.

27 Alt+PrntSc Take a screenshot of the current window.

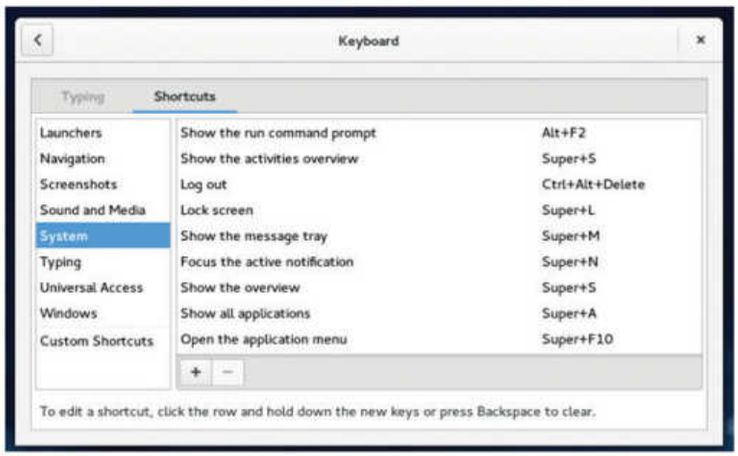
28 Shift+Ctrl+ Alt+r Record a screencast in Gnome.

29 Super+Up Maximise windows in Gnome.

30 Super+Down Minimise windows in Gnome.

31 Super+ Left/Right Snap windows in Gnome.

32 Super+m View any missed notifications in Gnome.



Customise and use keyboard shortcuts to save the time navigating menus.

Tips for your favourite apps

Save time and be more productive with these hidden gems.

LibreOffice

33 Quick change case
Select the words, right-click and head down to Change case menu and select the required option.

34 Enable word completion
Go to Tools > AutoCorrect Options > Word Completion and toggle the 'Enable word completion' and 'Collect words' options.

35 Define Keyboard control
Go to Tools > Customise and click the Keyboard tab to modify any of the shortcuts.

36 Play media files
Head to Insert > Media > Audio or Video and select a media file. Select the media icon in the document to enable media controls.

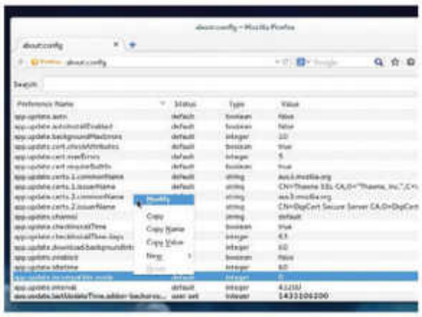
37 Use the Navigator
To swiftly navigate any documents or spreadsheet with the navigator window under View > Navigator.

38 Auto format tables
To auto format them, select some cells and head to Format > Autoformat to choose a different formatting for them.

39 Conditional formatting
Format the cells based on conditions specified under Format > Conditional Formatting > Condition.

40 Protect Sheet
Go to Tools > Protect Document > Sheet to lock access to the sheet with a password.

41 Status bar values
By default the status bar shows the sum of the values in the selected cells. Change the behaviour by right-clicking on the status bar.



Pop up the hood and take a look inside any Mozilla app with the about:config feature.

Evince

42 Autoscroll PDFs
Right-click inside a document and select the 'Autoscroll' option and use the mouse to control the speed.

43 Make text easier to read
Head to View > Inverted Colors to display white text on a black background.

44 Add Annotations
Select the Annotations option from the drop-down menu in the side pane and switch to the Add tab to add annotations.

Internet apps

45 Speed up the browser (Firefox)
Type `about:config` in the address bar. Then type `network.http` in the filter field and set the network.http.pipelining and network.http.proxy.pipelining parameters to True.

46 Limit RAM usage (Firefox)
Go to `about:config`, filter browser.cache and set the browser.cache.disk.capacity parameter to 30000 if you have 2GB of RAM.

47 Repair folders (Thunderbird)
Right-click the damaged folder, head to Properties and click the 'Repair Folder' button.

48 Create a mailing list (Thunderbird)
Head to Tools > Address Book > New List and specify which address book list to add addresses to and start adding addresses.

49 Store less mail locally (Thunderbird)
Head to Edit > Account Settings > Synchronisation & Storage for the desired account. Toggle the Synchronise the most recent option and choose the period.

50 Search all messages (Thunderbird)
To search through all mail, including mail only available in full on the server, head to Edit > Find > Search Messages and toggle the 'Run search on server' option.

51 Insert a background image (Evolution)
Toggle the Format > HTML option, head to Format > Page and click 'Browse' under Background image section and pick an image.

52 Advanced search (Evolution)
Head to Search > Advanced Search to create complex search rules. Use the 'Add Condition' button to define parameters.

Use Pidgin's Autoaccept files plugin to drop files in a folder that you can use with Tip No. 54 to add torrents remotely.

53 Optimise Torrent speed (Transmission)
Use **http://bit.ly/AzureuaUploadCalc** to determine the recommended settings that you can then enter in the Edit > Preferences > Speed and the Network tabs.

54 Monitor directory (Transmission)
Head to Edit > Preferences > Downloading and toggle the 'Automatically add .torrent files from' option and pick a directory.

55 Remote control torrents (Transmission)
Transmission ships with a browser-based interface that can be enabled from Edit > Preferences > Remote.

56 Use a privacy-centric profile (Firefox)
JonDoFox is a *Firefox* profile that automatically integrates with the installed browser and allows you to browse the internet anonymously using a proxy server.

Media players

57 Auto-fetch subtitles (Gnome Videos)
Press Ctrl+Shift+s to open the Movie Subtitles dialog. Now select the language and click 'Find' to look for subtitles on the **www.opensubtitles.org** website.

58 Covert media files (VLC)
Head to Media > Convert/Save, add a file and click 'Convert/Save' button and select the desired codec to convert to.

59 Download online videos (VLC)
Go to Media > Open Network Stream and enter the URL of the video and use the Play pull-down menu and choose 'Convert'. Then select a preset Profile, enter the filename to save and click 'Start'.

60 Record desktop (VLC)
To enable desktop recording, go to Media > Convert / Save > Capture Device. In the Capture mode drop down menu, select

Desktop, then select your frame rate. Finally, click 'Convert/Save', give it a name and click 'Start'.

61 Remote control VLC from a browser (VLC)
Go to Tools > Preference and toggle the 'All' button under Show settings. Now go to Interface > Main Interfaces and toggle the 'Web' option. Then under Main Interface > Lua, set the Lua HTTP Password.

62 Identify a song (Amarok)
Right-click the song you can't recognise, head to Edit Track Details > Tags and click 'Get Tags from MusicBrainz'.

Image editors

63 Move the selection mask (Gimp)
Make a selection, then click the Move tool. Make sure that the Move option is set to 'Selection' in the panel and you can now drag the selection into a new position.

64 Rounded corners (Gimp)
Go to Filters > Decor > Rounded Corners. Then select the 'Edge Radius', which is the amount of curve and optionally customise the other options.

65 Batch process images (Gimp)
Grab and install David's Batch Processor plugin (**http://bit.ly/DavidsBP**) to enable all kinds of tweaks.

66 Automatically write metadata to images (Shotwell)
Head to Edit > Preferences and toggle the Write tags, titles and other metadata to photo files checkbox.

67 Organise photos by events (Shotwell)
By default, *Shotwell* clubs all photos uploaded in one go in a single event. For better organisation you can create new events from a selected group of photos from under Events > New Event.

68 Render RAW files correctly (Shotwell)
To ask *Shotwell* to use the camera's RAW developer, just open an image and toggle the Photo > Developer > Camera option.

KDE apps

69 Bookmarks locations (Konsole)
Use the Bookmarks menu to bookmark any directory. The 'Bookmark Tabs as Folder' option lets you bookmark all open tabs in a single folder.

70 Label tabs (Konsole)
If you've bookmarked a bunch of tabs that you use regularly, you can name them by double-clicking on the tab.

71 Run command on multiple sessions (Konsole)
Use Edit > Copy Input To All Tabs in the Current Window, or Select Tabs if you wish to run the same command, eg on multiple SSH'd hosts.

72 Monitor activity (Konsole)
Enable the View > Monitor for Activity option and KDE will notify you with a popup in the taskbar whenever there's any activity in that Konsole tab.

73 New tab in custom directory (Konsole)
Head to Settings > Edit Current profile and first disable the Start in the same directory as current tab option and then enter the location of the custom start directory in the field above it.

74 Create custom profiles (Konsole)
You can create new profiles with custom fonts and permissions by heading to Settings > Manage Profiles > New Profile. Then customise it by switching to the other tabs such as Appearance.

75 Read-only editor (Kate/Kwrite)
Toggle the Tools > Read Only Mode option to prevent accidentally making changes to an important document.

76 Change highlighting (Kate/Kwrite)
Choose the appropriate highlighting mode for the currently open document by heading to Tools > Highlighting.

VirtualBox

77 Create VM snapshots
To save the current state of a VM, switch to the Snapshot tab in the main interface and click the 'Take Snapshot' button. You can restore snapshots from this interface later.

78 Use USB devices
Head to the Devices > USB Devices and select the USB devices you want to connect which will then be disconnected from the host and made available to the VM.

79 Forward virtual ports
Setup Port Forwarding to ensure any server software inside the VM is accessible from the Internet by heading to Settings > Network > Advanced > Port Forwarding.

80 Enable remote display
If you run *VirtualBox* on a headless server, you can enable the remote display by heading to Settings > Display > Remote Display and toggling the Enable Server checkbox.

81 Manage VirtualBox from a browser
Another useful application for managing *VirtualBox* from a remote computer is *phpVirtualBox*, which recreates the interface inside a web browser.

82 Share clipboard
If you've installed the Guest Editions enable the appropriate option under Devices > Shared Clipboard to copy/paste text between the guest and host.

File manager shortcuts

83 F4 (KDE Dolphin)
Displays the in-line command line.

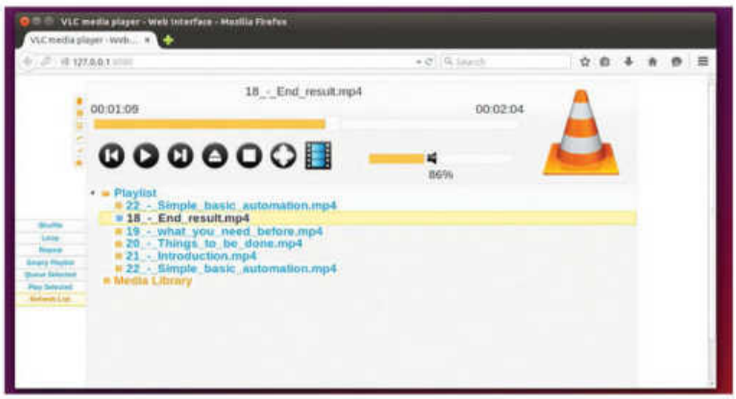
84 F3 (KDE Dolphin)
Splits a single window into two different views.

85 Ctrl+I (KDE Dolphin/ Gnome Nautilus)
View the location bar if hidden (Note: lowercase L).

86 Shift+Enter (Gnome Nautilus) Open the selected folder in a new tab.

87 Ctrl+Shift+ drag the file (Gnome Nautilus)
Creates a soft link to the file.

88 Spacebar (Gnome Nautilus) Preview the selected file if the *Sushi* previewer is currently installed.



Once activated, the *VLC* web interface is available at localhost:8080.

Better manage software

Use the command line to get more from your distro's package manager.

Tips for RPM/Yum/Fedora

89 Install RPMs with Yum
To resolve and fetch dependencies install RPM packages with `yum install <package.rpm>`.

90 Update a particular package
Use `yum check-update <package>` to check for updates for the package which you can install with `yum update <package>`.

91 Search for packages
Use `yum whatprovides <name>` to fetch the name of the package that provides the mentioned file.

92 Install package groups
List all available groups with `yum grouplist` and install any with `yum groupinstall <group-name>`.

93 Rollback updates
Get a list of actions along with their update IDs with `yum history` and undo one with `yum history undo [update-id]`.

94 Speed up Yum
Install the fastestmirror plugin with `yum install yum-plugin-fastestmirror` and always use the closest mirror to install a package.

Tips for Apt/DPKG/Ubuntu/Mint

95 Backup package list
To install the same packages on another machine, create a list of installed packages with `dpkg --get-selections > pkgs.list`.

96 Replicate on another system
On a fresh installation, first import the list of packages with `dpkg --set-selections < pkgs.list` and then install them with `apt-get dselect-upgrade`.

Use Ubuntu's Additional Drivers tool to install proprietary drivers for your graphics card and other hardware.

97 Uninstall apps
To completely uninstall apps along with their configuration files, use `apt-get remove --purge <app>`.

98 Downgrade packages installed from PPAs
Install the PPA purge tool with `apt-get install ppa-purge` and revert upgraded packages with `ppa-purge <ppa-repo>`.

99 Install dev libraries
To compile a newer version of an app fetch the dev libs of the version in your repos with `apt-get build-dep <app-name>`.

Linux Mint has great custom software management tools for easy management of mirrors and PPAs.

100 Remove archives
Use `apt-get autoclean` to remove downloaded archives of packages that have since been upgraded to newer versions. You can also get rid of them all with `apt-get clean`.

101 Remove unnecessary packages
The `apt-get autoremove` command zaps all dependencies no longer in use.

102 Fix broken dependencies
Use `apt-get -f install` if you get an error while trying to install a Deb package without installing its dependencies first.

103 Use fastest mirror
In Ubuntu's *Software & Updates*, select 'Other' from the Download from the menu and click the 'Select best server' button.

Tips for URPMI/Mageia

104 Fetch a list of dependencies
The command `urpmq -d <pkg-name>` will get a list of required package dependencies.

105 Update all media
Use `urpmi --auto-update` to update the list of available packages.

106 Saves the RPMs
Append the `--noclean` option to prevent urpmi from automatically deleting the downloaded rpms after installing an app.

107 Install from a local directory
Drop RPMs inside a directory and then add it as an installation medium with `urpmi.addmedia backup <directory>`.

108 Install from a URL
Instead of first downloading packages you can install them directly from the web with `urpmi <URL-to-the-rpm>`.

Tips for ZYpp/OpenSUSE

109 List installed packages
The `rpmqpack` command displays a list of all installed packages.

110 Update a package
Use `zypper in <app-name>` to update a package. The command will also install the package if it isn't yet installed.

111 Faster zypper
Use `zypper sh` to enter the Zypper shell which installs packages faster as it keeps all relevant data in memory.

112 Simulate an upgrade
Before you upgrade your installation do a dry run with `zypper -v dup -D`.

113 Backup repos
Save all the configured repos with `zypper lr --export ~/backup-repos.repo`.

114 Restore repos
Use `zypper ar ~/backup-repos.repo` to restore repos from the backed up file.

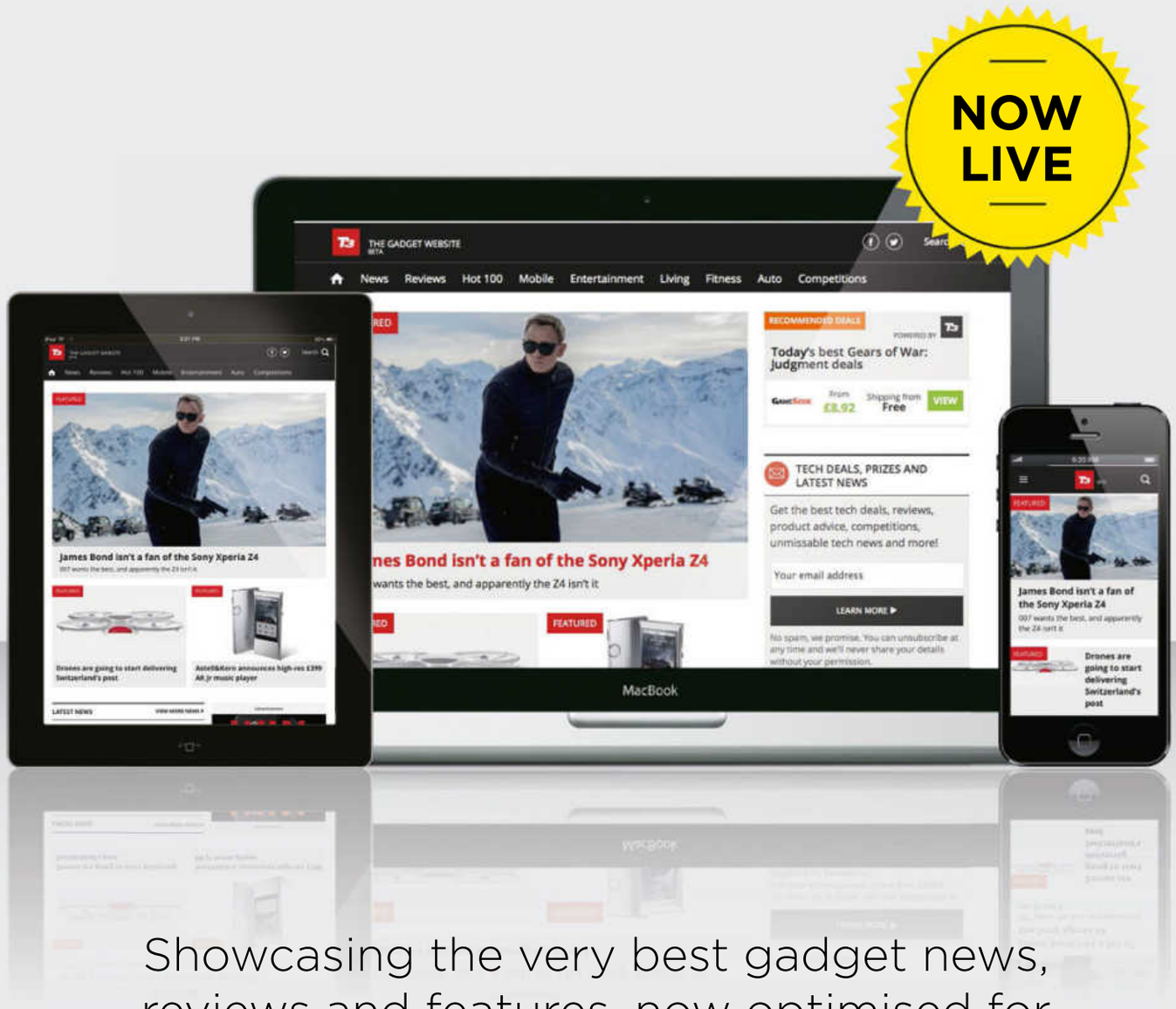
115 View required patches
Fetch a list of required update patches with `zypper lp`.

116 Install patches
Upgrade apps by applying all available patches with `zypper patch`.

T3

THE GADGET WEBSITE

INTRODUCING THE ALL-NEW T3.COM



Showcasing the very best gadget news, reviews and features, now optimised for any screen, anytime, anywhere.

www.T3.com

Future

GIZMODO | UK

Not your average technology website

EXPLORE NEW WORLDS OF TECHNOLOGY GADGETS, SCIENCE, DESIGN AND MORE

- Fascinating reports from the bleeding edge of tech
- Innovations, culture and geek culture explored
- Join the UK's leading online tech community

www.gizmodo.co.uk

twitter.com/GizmodoUK facebook.com/GizmodoUK

Power user tips

Become a master of your domain.

System administration

117 Monitor remote systems
Launch KDE's *KSysGuard* and go to File > New Tab. Then switch to the new tab and head to File > Monitor Remote Machine and enter the target machine's IP address and connection details.

118 Mount ISO files
Use mount `-o loop <path-to-ISO-file> /tmp/iso-file` to explore the contents of an ISO image.

119 Create virtual consoles
With *tmux* you can create multiple sessions, run different tasks in each, and then switch from one session to another without interrupting the task running inside them.

120 Use tar efficiently
The tar archiver can detect compression formats and `tar xf <compress-file>` is all you need to unpack a file.

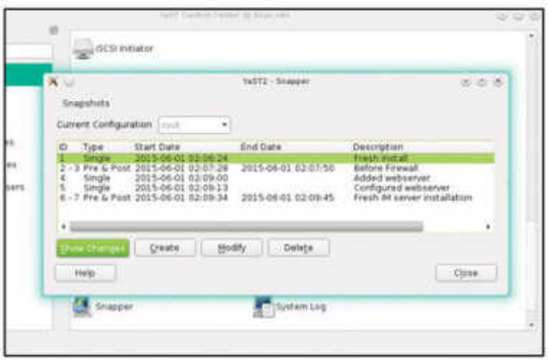
121 Set one-off reminders
You can use `at` with `notify-send` to set short time reminders, such as `echo notify-send "Check on the tea" | at now +4 min`.

122 Schedule a job for multiple times
Use a comma in the crontab file specify multiple times. For example `00 11,16 * * * <task>` executes the task everyday at 11am and again at 4pm (11,16).

123 Run a job within a specific duration
Similarly use a hyphen to specify a range. eg, `00 10-17 * * 1-5 <task>` performs the task from Monday-Friday (1-5) between 10am and 5pm (10-17).

124 Execute a command after every reboot
Use the `@reboot` keyword to run a job whenever the computer starts up.

125 View multiple log files simultaneously
You can install *multitail* from the repos to view multiple file in the following way, eg `multitail /var/log/syslog /var/log/boot.log`.



OpenSUSE's *Snapper* tool helps you manage snapshots of the distro's btrfs filesystem.

Bash tips

126 View commands matching a pattern
Search through previously executed commands that match a pattern using `history | grep -i <first-few-letters-of-command>`.

127 Reuse arguments from an earlier command
You can use the colon (:) key to reuse the options from the previous command, such as `!!:2` points to the second argument in the previous command.

128 Preview a command before executing
Test your complex *Bash* statements by appending `:p` at the end, such as `ls -l !tar:3:p`.

129 Create shortcuts for commands
You can roll often repeated complex commands into custom ones with `alias`, such as `alias sshbox1='sudo ssh bodhi@192.168.3.111'`. To make aliases permanent add them to the **~/.bashrc** file.

130 Autocorrect CLI typos
You can use `shopt` to autocorrect any common *Bash* typos you tend to create. First, enter `shopt` to display all the available patterns and enable any with `shopt -s`. For example, using `shopt -s cdspell` will find nearest match to misspelt directory names.

131 Create files that are tough to delete
A file with a leading or trailing space in it's name or a hypen (-) cannot easily be zapped from the CLI.

132 Delete tough to delete files
Once you've create a tough file to delete, here are several ways to get rid of files with peculiar names. You can wrap the filename in quotes or use double hyphens, such as `rm " example"` or `rm -- -example`.

133 Delete all files except some
Use the ! operator to remove all files except those that match the specified pattern. For example. `rm ~(*.txt)` will remove all files in the directory that don't end with .txt.

Performance

134 Get details about the hardware
The `dmidecode` command will spit out detailed information about your computer's hardware. For example, using `dmidecode -t 16` will list details about the physical memory. Browse the dmidecode man page for a list of supported DMI types.

135 List process in a hierarchy
You can use `ps --forest` to represent the process tree in ASCII art and clearly identify parent and child processes.

CLI shortcuts

136 Ctrl+a
Send the cursor to the start of the command.

137 Ctrl+e
Send the cursor to the end of the command.

138 Ctrl+l (lowercase L)
Clear the screen but retain what's on the current prompt.

139 Ctrl+k
Cut text starting from the command prompt.

140 Ctrl+y
Short for 'yank'. Paste the text in the buffer.

141 Ctrl+Shift+c/v
Copy and Paste text to the CLI.

Bash shortcuts

142 Shift+PgUp/PgDown
Scroll the console.

143 Ctrl+r
Search command history.

144 ! <event-number>
Repeat a command from history.

145 !!
Repeat the last command.

146 Alt+. (dot)
Prints the last argument of the previous command.

147 > <filename>
Empties specified file.

148 Find memory leaks
To figure out which processes are hogging up the RAM, use `ps --sort mem` which arranges processes in ascending order of memory consumption with the heavy consumers at the bottom.

149 Memory of a particular process
View a detailed memory consumption report of a particular process with `pmap -x <PID>` which displays the amount of resident, non-shared anonymous, and locked memory for each mapping.

150 Trace the execution of a binary
If you have an unknown binary, trace its execution with `stace <binary>` to view all the system calls and signals it makes.

151 Track logged in users
Use the `w` command to get a list of logged in users and their processes. Add the `-f` option to include the hostname of remote users in the output.

152 Kill a graphical app
Type `xkill` in the terminal or the run dialog box which changes the pointer into a cross-hair cursor. Now click on any non-responsive window to kill it. Right-click to dismiss `xkill` without killing a process.

153 Decrease use of swap
If you've got ample RAM, optimise swap usage by editing the **/etc/sysctl.conf** file and changing the value of the vm. swappiness parameter to 10.

Backup

154 Backup the boot sector
A boot sector backup comes in handy when you accidentally wipe out your MBR. Make a backup of a healthy boot sector with `dd if=/dev/sda of=disk.mbr count=1 bs=512` and restore it with `dd if=disk.mbr of=/dev/sda`.

Mageia also includes a Parental Controls for time-based and app-based restrictions.

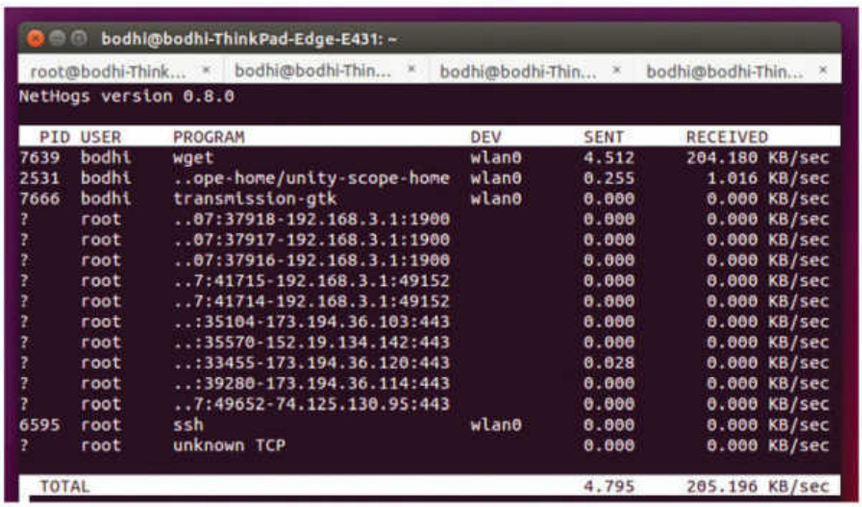
155 Backup partition table
You should also keep a backup of your partition table in the event when a mishap or other zaps this crucial bit of information. Use `sfdisk -d /dev/sda > disk.sf` to backup the table and `sfdisk /dev/sda < disk.sf` to restore the partition table.

156 Monitor the progress of dd
Install the *Pipe Viewer* (pv) tool from your distro's repos and use it to monitor `dd`. For example, `pv -tpreb some-distro.iso | sudo dd of=/dev/sdb bs=4096`.

157 Speed up backups on slower machines
If bandwidth isn't a problem, use `rsync -W` to transfer whole files and save time spent computing the changed blocks and bytes.

158 Track rsync progress
Append the `--progress` option to the `rysnc` command to keep an eye on the data transfer.

159 View changes between source and destination
Use the `-i` option to view the list of items modified by an *rsync* operation, such as `rsync -avzi [source] [destination]`.



Use *Nethogs* to get a real-time view of the bandwidth being consumed by an application.

160 Use rsync over ssh
To transfer *rsync* data over SSH use the `-e ssh` option, such as `rsync -avhze ssh [source] [destination]`.

161 Exclude files
Rsync also lets you skip over certain files that you can specify with the `--exclude` option, like `rsync -avhz --exclude '*.tmp*'` will ignore files with the .tmp extension.

162 Test rsync
First time users should append a `--dry-run` option to all *rsync* operations and scan the output for any unexpected outcomes before running it for real.

163 Limit bandwidth
To make sure the *rsync* operation doesn't hog all the bandwidth restrict its usage with the `--bwlimit option, such as rsync -avhz --bwlimit=50`.

164 Don't backup files on external filesystems
Tar is a popular choice for creating an archive of the disk. Use the `--one-file-system` option with tar to make sure it doesn't backup any mounted partitions (**/media**) or virtual partitions (**/proc**, **/sys**).

Security & Firewall

165 Find which port a program is running on
Use `netstat -ap | grep [app-name]` to see a list of ports that a particular application is communicating from.

166 Disable ping reply
Pings can be used to flood the network and cause network congestion. Disable it temporarily with `echo "1" > /proc/sys/net/ipv4/icmp_echo_ignore_all` or permanently by editing the `/etc/sysctl.conf` file to add `net.ipv4.icmp_echo_ignore_all = 1`.

167 Backup iptables
If you've spent customising the kernel's iptables firewall, make sure you back it up with `iptables-save > ~/iptables.backup`

168 Block a particular domain
First, you need to figure out the domain's IP address with `host -t a www.example.com`. Then use the IP address to get its CIDR with `whois [IP Address] | grep CIDR`. Then use the CIDR to block access to the domain, such as `iptables -A OUTPUT -p tcp -d [CIDR] -j DROP`.

169 Change password for any user
If you've forgotten a password for a user, you can set a new one with `sudo passwd [username]` without being prompted for the current password.

170 **Replicate permissions**
Use the `--reference` option to copy the permissions of one file to another, such as `chmod --reference=[copy-permission-from-this-file] [apply-on-this-file]`.

171 **Securely delete files**
Install and use the *shred* utility to delete files so that they cannot be recovered. For example, `shred [file]` will overwrite the file's block with random data several times.

172 **Enable built-in Firewall**
Some distros such as Ubuntu ship with a simpler front-end to *iptables* firewall, called *UFW*. It's disabled by default but you can enable it with `ufw enable`.

173 **Allow incoming connection**
UFW denies all incoming connections by default. To tweak this policy and allow connections for common servers do `ufw allow ssh, sudo ufw allow www, ftp`.

Network & Internet

174 **Run commands remotely**
You can also use SSH to execute commands on a remote machine, such as `ssh [hostname] [command]`.

175 **Copy SSH keys to another machine**
Use `ssh-copy-id [remote-host]` to securely copy the public key of your default identity to the remote host.

176 **Keep connection open**
If you frequently get disconnected from remote SSH sessions due to lack of activity, you can enable the KeepAlive option by adding the `ServerAliveInterval 60` line in the **/etc/ssh/ssh-config** file.

177 **Browse via SSH tunnel**
First create an SSH tunnel to a remote host with `ssh -f -N -D 1080 user@remotehost`. Then change your web browser's Proxy settings and set the SOCKS host to **127.0.0.1** and the port to **1080**.

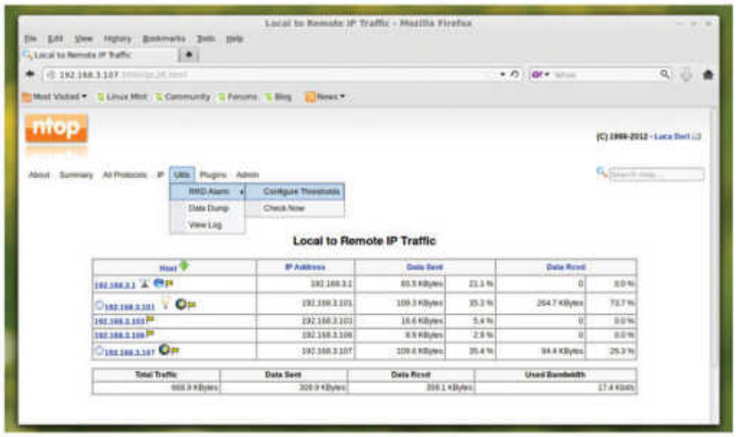
178 **Play music over SSH**
The command `ssh user@remotehost cat ~/Music/audio.ogg | mplayer` will redirect the output of the remote media file to mplayer on the local machine.

179 **Mount partitions over SSH**
Use *sshfs* to mount remote partitions such as `sshfs user@remotehost:/home/bodhi /media/remotefs` to mount the remote home directory under the local filesystem.

180 **Better monitor network traffic**
Ntop is available in the official repos of most distros and gives you detailed analysis of the network traffic via its web-based interface running on port 3000.

181 **View network statistics**
Use `netstat -s` to view statistics for all protocols or `netstat -st` for only the TCP protocol.

182 **Save a webpage**
Use `wget` to properly download a webpage. eg, `wget -r -np -k http://www.tuxradar.com/content/dear-edward-snowden` will download all images and change the links in the HTMl and CSS files to point to local files.



***Ntop* is a versatile tool that can be extended with plugins.**

183 **Save multiple files**
If you have saved links to multiple downloads in a file, use `cat isos.txt | xargs wget -c` to download them all.

184 **Limit data transfer rate**
Prevent *wget* from hogging all the bandwidth by imposing limitations, such as `wget --limit-rate=2m` will limit the transfer rate to two megabytes per second.

185 **Download files based on modification date**
Use curl with the `-z` option to only download files that have been modified after a particular time. For example, `curl -z 29-May-2015 [download-location]`.

186 **Upload files**
You can use *curl* to connect to a FTP server and upload files, such as `curl -u [user:pass] -T upload.txt ftp://ftp.example.com`.

187 **Get definitions**
Curl can fetch the definition of a word from a directory server. List them all with `curl dict://dict.org/show:db` and then query one with `curl dict://dict.org/d:shell:foldoc` which fetches the definition of the word 'shell' from the Foldoc dictionary.

188 **Simple web filtering**
To prevent your computer from accessing a website enter its URL in **/etc/hosts**, such as *127.0.0.1* **www.addictivewebsite.com**.

189 **Mirror entire websites**
Use the graphical *WebHTTrack* tool available in the official repos of most distros to mirror entire websites and automatically modify links.

190 **Regulate bandwidth**
You can use *Trickle*, lightweight userspace bandwidth shaper, to control both upload and download speeds. It can also regulate speeds for package managers such as `trickle -d200 apt-get install`.

191 **Monitor bandwidth**
To monitor bandwidth used by individual network applications use the *nethogs*, a small net top tool that's available in the repos of most distros. Instead of breaking traffic down by protocol it groups bandwith by process.

Top command shortcuts

192 **Shift+m** Sort by RAM utilisation.

193 **k** Kill a task from within *top*.

194 **1** Track all cores individually within *top*.

195 **Shift+w** Save the modified configuration permanently.

less command shortcuts

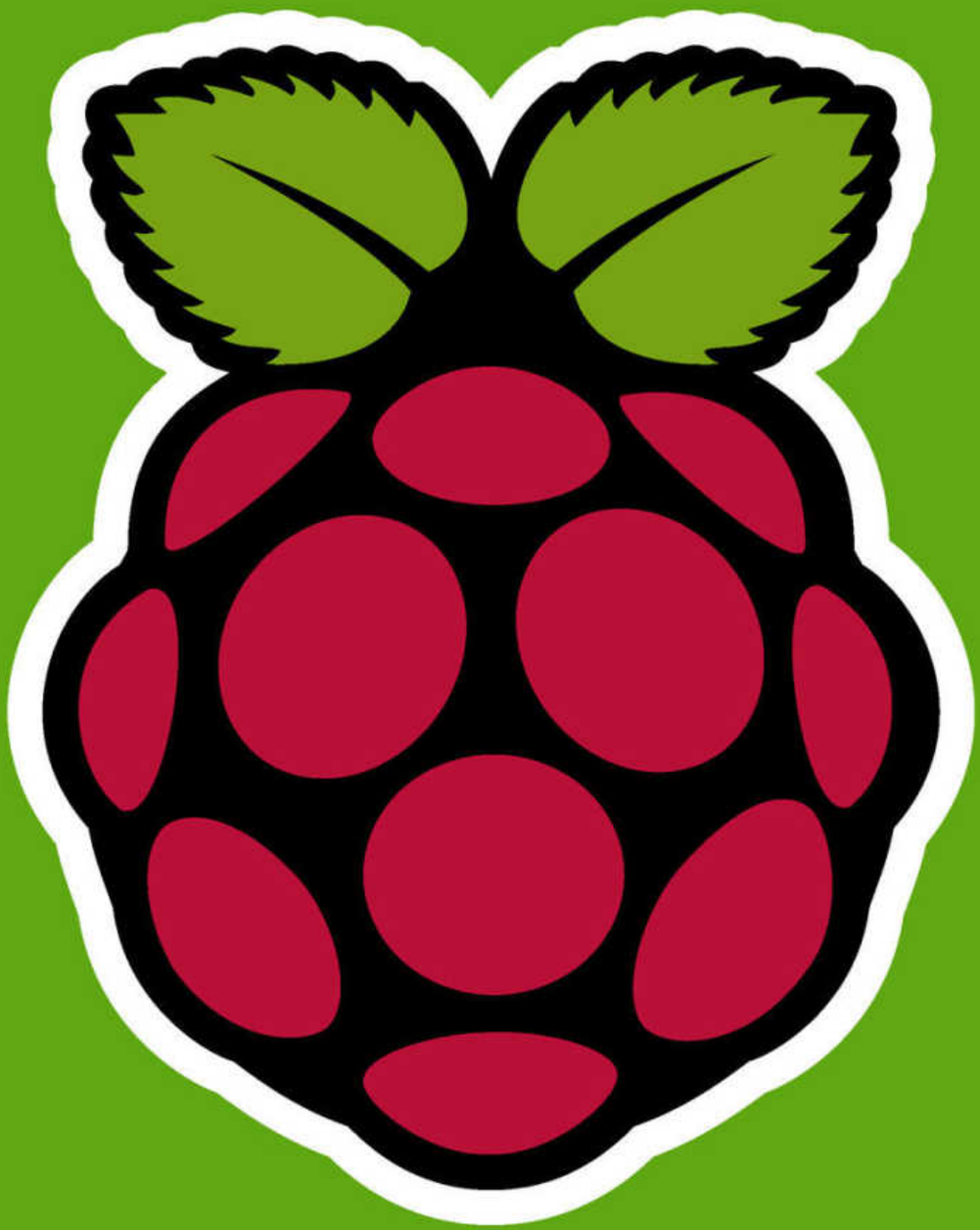
196 **/** Search forward for a pattern.

197 **n** Next match.

198 **Shift+f** Displays new content as it's appended to the file.

199 **v** Edit the file with your system's configured editor.

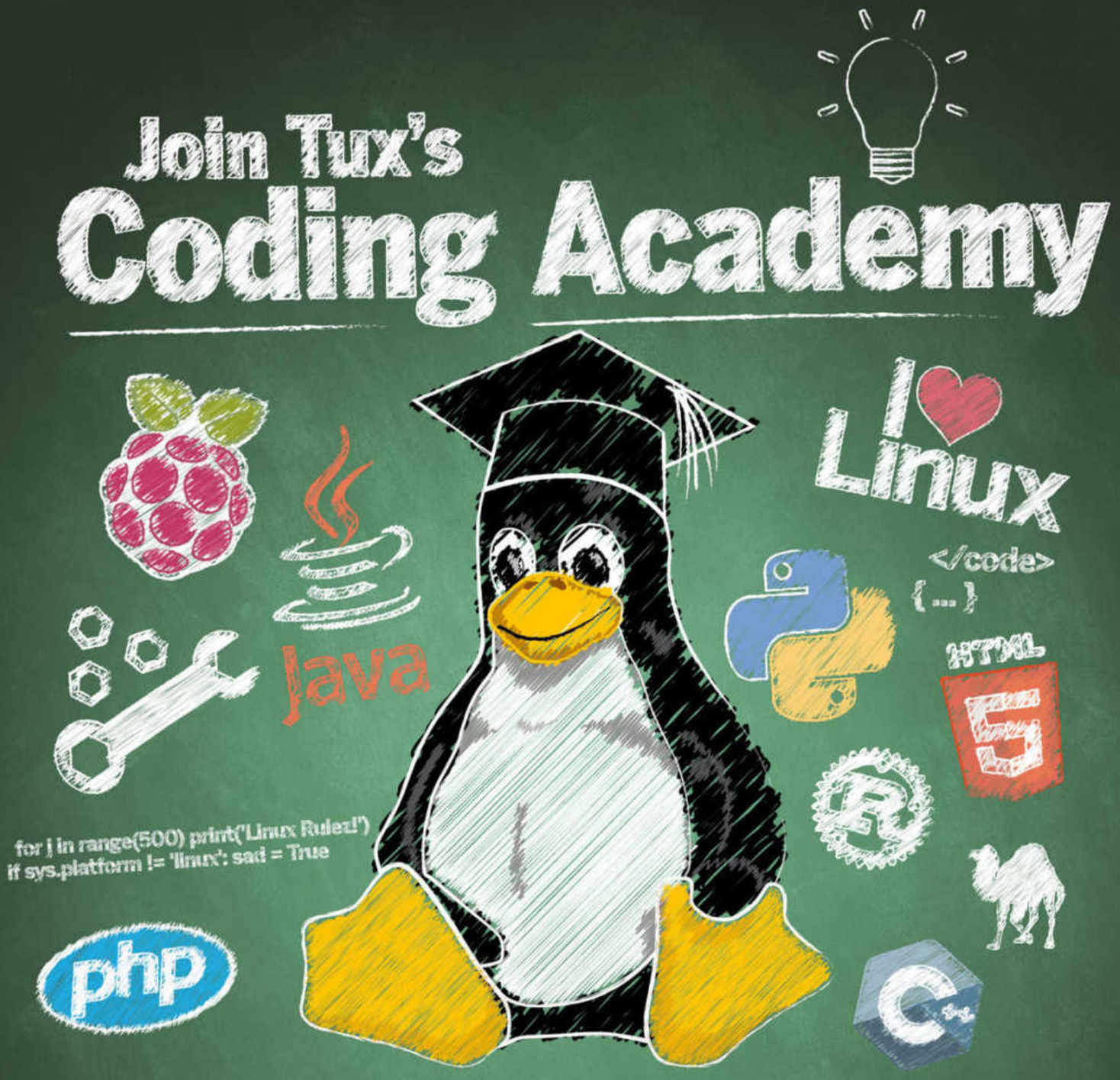
200 **h** View the complete list of shortcuts.



Coding

You too can be a programmer.

Join Tux's Coding Academy



Coding isn't scary. Promise. If you've always wanted to learn, there's no better time to get started.

Coding is the new cool. If you don't know how to Python, all your friends are going to be laughing at you...

We're not being flippant. Knowing how to code is almost an essential for the open source fiend. Be it basic Bash scripting or knowing how to read a bit of PHP, coding knowledge is more than useful, it's an essential skill, particularly now that coding is being taught in schools.

Knowing a bit of code helps with using Linux itself, helps you get more out of the terminal, can open doors to a new career or help you troubleshoot why that webpage won't work correctly. Other than taking a bit of time to pick up, there's also no cost and as a FLOSS user, access to every language under the sun is just waiting an **apt-get** away.

So take our hand and let's take just a few minutes to create a fun game in Python, learn which languages are right for you and your projects, then see how you can tackle the new school curriculum and even web development.

Get with the Program

This is going to be a very gentle introduction to programming in Python in which we'll explore the basics of the language, then use the *Pygame* module to make a simple game. That we are able to do this is testament to the power of Python and *Pygame*: much of the tedium inherent in game work is abstracted away and, once we've covered some elementary programming constructions, for the most part we work with intuitive commands.

First follow the instructions in the box to check which version of Python you have and install *Pygame*. Whether you're on a Raspberry Pi or a larger machine, start at the command line. So open up a terminal (*LXTerminal* in Raspbian) and start Python 2.7 with

```
$ python2
```

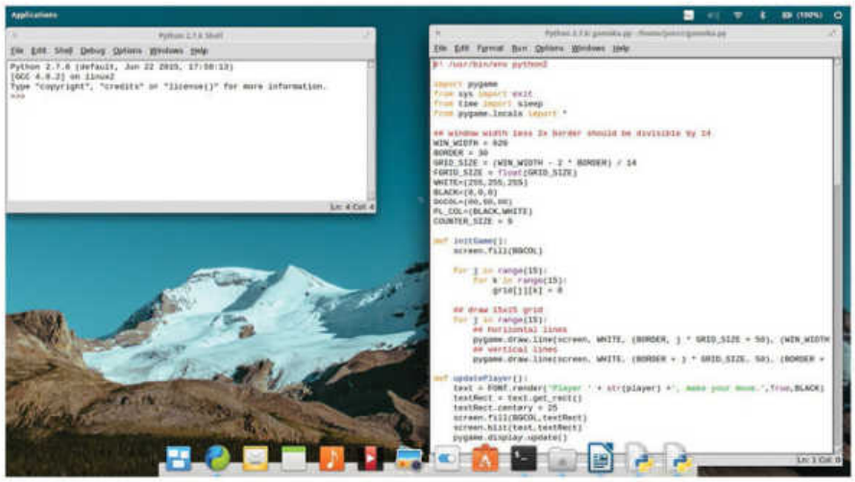
Alternatively, start up *IDLE* and start a command prompt from there by choosing 'Python 2.7 Shell' from the Window menu. Both will start the Python interactive interpreter, wherein Python commands can be typed at the `>>>` prompt and evaluated immediately. It is here that we will forge our first Python program, by carefully typing out the following incantation:

```
>>> print('Hello World')
```

As you might suspect, pressing Enter causes Python to display a global greeting. You can exit the interpreter at any point by pressing Ctrl-D or using the `exit()` command. For larger programs it makes sense to work in a text editor, or an Integrated Development Environment (like IDLE), but the interpreter is a great way to test smaller code fragments. So we'll use it to now to introduce some fundamental coding concepts and constructs. First let's introduce the idea of variables:

```
>>> name = 'Methuselah'
>>> age = 930
>>> print(name, ' is ', age, ' years old.')
```

So we can assign values to variables,



The IDLE development environment is purpose built for Python. You can install it on Ubuntu (or in this case ElementaryOS) with `apt-get install idle`.

change them to our hearts' content, and use `print` statements to see them. Technically, the brackets in the `print` line are only required in Python 3, but they don't do any harm and it's good practice to write, wherever possible, ambidextrous code that will run in both versions. Variables in Python are automatically assigned a type based on their content. So `name` is a string (short for a string of characters) and `age` is an integer (a whole number). You can check this by typing `type(name)` etc.

Some types can be coerced to other types – we can transmute `age` to a floating point number (one with a decimal part) with:

```
>>> age = float(age)
>>> age
```

Just typing the variable name into the interpreter will show you its value, so you can easily see the changes you have enacted. We can convert ints or floats to strings, using the function `str()`. Python can also go the other way, converting a string to a float for example, but this will only work if the original string looks something like its target type: `float('10.0')` will work, but `float('Rumplestiltskin')` will not.

Just as division works differently for floats and ints, so addition works differently for strings. Here the `+` operator stands for concatenation, tacking the latter string onto the end of the former. Thus:

```
>>> 'Hello ' + 'world'
'Hello world'
>>> str(123) + str(456)
'123456'
>>> 'Betelgeuse' * 3
'BetelgeuseBetelgeuseBetelgeuse'
```

The last line shows that we can also multiply strings – division and substraction, however, are not defined for strings.

Data types dictate how data is represented internally, and the effects of this can be quite subtle. For example, in Python 2 the division operator `/` works differently if one of its arguments is a float:

Installing Python and Pygame

If you're using Raspbian on the Raspberry Pi or any flavour of desktop Linux, then the chances are that you already have at least one (probably two) versions of Python installed. While the latest release (1.9.2) is now Python 3 compatible, no distributions are shipping this version yet, so we'll stick to Python 2 (2.7 to be precise) for this tutorial. Check your default Python version by typing:

```
$ python -V
```

If this returns a result that begins with 2, then everything is fine. If, on the other hand, you see 3-point-something, then check Python 2 availability with the command:

```
$ python2 -V
```

Some distros, notably Arch Linux and the recently released Fedora 22, don't ship the 2.7 series by default. Installing *Pygame*, however, will pull it in as a dependency, so let's do that now. Users of Debian-derived distributions (including Raspbian on the Pi) should use the following command to install *Pygame*:

```
$ sudo apt-get install python-pygame
```

Users of other distributions will find a similarly named package (on Arch it's called python2-pygame) and should be able to install it through the appropriate package manager, whether that's *pacman yum*, *zypper* or whatever. Most distributions bundle the IDLE environment with each version of Python installed; if you can't find an icon for it, try running the commands `idle` or `idle2`. If that fails to produce the goods, then go hunting in your distro's repos.

```
>>> 3/2
>>> 1
>>> 3/.
>>> 1.5
```

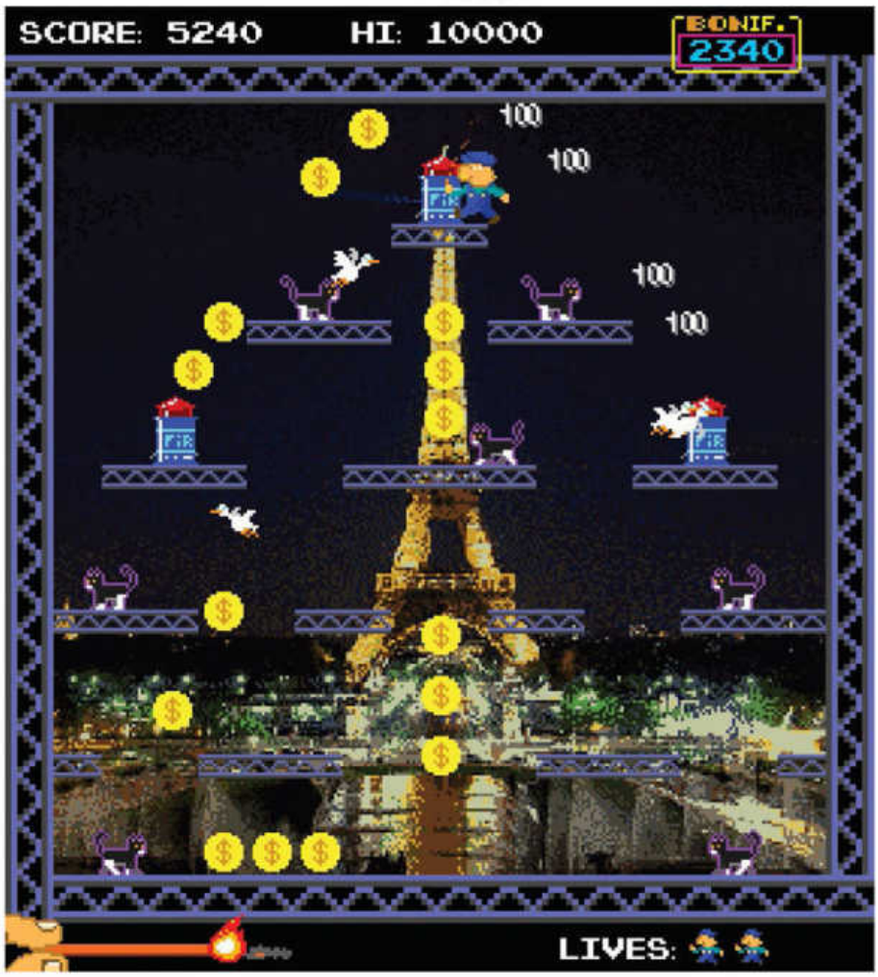
Funny the difference a dot can make. Note that we have been lazy here in typing simply `2.` when we mean `2.0`. Python is all about brevity. (Besides, why make life hard for yourself?) Sooner or later you'll run into rounding errors if you do enough calculations with floats. Check out the following doozy:

```
>>> 0.2 * 3
0.6000000000000001
```

Such quirks arise when fractions have a non-terminating binary decimal expansion. Sometimes these are of no consequence, but it's worth being aware of them. They can be worked around either by coercing floating point variables to ints, or using the `round()` function, which will give you only the number of decimal places you require. We'll see this in practice when we program our Gomoku game later.

Going loopy

Very often programmers desire to do almost the same thing many times over. It could be appending entries to a list, adding up totals for each row in a table, or even subtracting energy from all enemies just smitten by a laser strike. Iterating over each list item, table row or enemy manually would be repetitive and make for lengthy, hard-to-read code. For this reason we have loops, like the humble `for` loop below. When you hit Enter after the first line, the prompt will change to `...` . This is because the interpreter knows that a discrete codeblock is coming and the line(s) following the `for` construct 'belong' to it. Such codeblocks need to be indented, usually using four spaces, though you can use as many as you like so long as you're consistent. If you don't indent the second line, Python will shout at you. Entering a blank line after the `print` statement will end the codeblock and cause our loop to run.



There are all kinds of *Pygame*-powered games. This one, *You Only Get One Match,* features lots of fireworks but limited means of ignition. Check it out at http://bit.ly/LXF202-onematch

```
>>> for count in range(5):
...         print('iteration #', count)
```

There's a few things going on here. We have introduced a new variable, an integer by the name of `count` . Giving sensible names to your variables is a good idea; in this case it implies (even in the absence of any other coding knowledge) that some counting is about to happen. The `range()` function, when used in isolation, returns a list consisting of a range of integers. We'll cover lists in just a moment, but for now we just need to know that `range(5)` looks like [0, 1, 2, 3, 4], which you can verify in the interpreter. So our variable `count` is going to iterate over each of these values, with the `print()` line being issued five times – once for each value in the range.

Another type of loop is the `while` loop.

How to play Gomoku

Gomoku is short for gomokunarabe, which is roughly Japanese for 'five pieces lined up'. The game in fact originated in China some 4,000 years ago. Players take turns to each place a counter on the intersections of a square grid with the goal of forming unbroken lines (horizontal, vertical or diagonal) of length 5. Traditionally the board has 19x19 intersections, but we've gone for the smaller 15x15 board used in some variations. We haven't included an AI (that would be somewhat too complicated for a beginner tutorial) so you'll need to find a friend/other personality with which to play. Alternatively there are plenty of online versions you can play, and KDE users get the similar *Bovu* game with their desktop.

It's easy to figure out some basic strategies, such as always blocking one side of your opponent's 'open three' line or obstructing a 'broken four'. Yet to become a master takes years of practice. The basic rule set as we've implemented it heavily favours the starting player (traditionally black). In fact, work by L. Victor Allis has shown that a good player (actually a perfect player) can force a win if they start. To mitigate against this, big tournaments use a starting strategy called swap2. The first player places two black counters and one white one on the board, and the second player then either chooses a colour or places another black and another white counter on the board and allows player 1 to choose colours. You are free to modify the code to force use of swap2, but it's entirely possible to obey this rule without any code modification: just disobey the first few 'Player x, make your move' prompts.

Rather than iterating over a list, our wiley `while` loop will keep going over its code block until some condition ceases to hold. In the following example, that condition is that the user claims to have been born after 1900 and before 2016.

```
>>> year = 0
>>> while year < 1900 or year >= 2015:
...     year = input("Enter your year of birth: ")
...     year = int(year)
```

Again the loop itself is indented, and again you'll need to input a blank line to set it running. We've used the less than (`<`) and greater than or equal to (`>=`) operators to compare values. Conditions can be combined with the logical operators `and`, `or` and `not`. 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. We've used the `input()` function, which returns whatever string the user provides. This we store in the variable `year`, which we convert to an integer so that the comparisons in the `while` line do not fail. It always pays to be as prudent as possible as far as user input is concerned: a malicious user could craft some weird input that causes breakage, which while not a big deal in this example, is bad news if it's done, say, on a web application that talks to a sensitive database. 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.

The opposite of listless

We mentioned lists earlier and in the exciting project that follows we'll use them extensively, so it would be remiss not to say what they are. Lists in Python are flexible constructions that store a series of indexed items. There are no restrictions on said items: they can be strings, ints, other lists, or any combination of these. Lists are defined by enclosing a comma-separated list of the desired entries in square brackets. For example:

```
>>> myList = ['Apple', 'Banana', 'Chinese Gooseberry']
```

The only gotcha here is that lists are zero-indexed, so we'd access the first item of our list with `myList[0]` . If you think too much like a human, then 1-indexed lists would make more sense. Python doesn't respect this, not even a little, so if you too think like a meatbag, then be prepared for some classic off-by-one errors. We can modify the last item in the list thusly:

```
>>> myList[2] = 'Cthulhu'
```

Lists can be declared less literally – for example, if we wanted to initialise a list with 100 zeroes, we could do:

```
>>> zeroList = [0 for j in range(100)]
```

This is what is known as a list comprehension. Another example is

```
>>> countList = [j for j in range(100)]
```

which results in a list containing the integers 0 up to and including 99, which could equally well be achieved with `range(100)` in Python 2. However, the concept is more powerful – for example we can get a list of squares using the exponentiation (to the power of) operator `**` :

```
>>> squareList = [j ** 2 for j in range(100)]
```

And after that crash course we're ready to program our own game. You'll find all the code you need to use at **http://pastebin.com/FRe7748B**, so it would be silly to reproduce that here. Instead we'll focus on the interesting parts, in some cases providing an easier-to-digest fragment which you can play with and hopefully see how it evolves into the version in the program.

To dive in and see the game in action, copy **gomoku.py** to your home and run it with:

```
$ python2 gomoku.py
```

On the other hand, if you want to see some code, open up that file in *IDLE* or your favourite text editor. Starting at the first line is a reasonable idea… It looks like:

```
#!/usr/bin/env python2
```

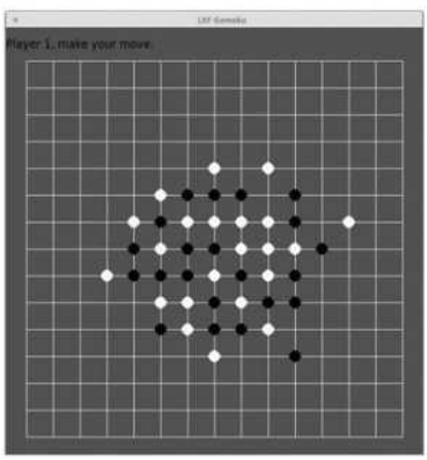
This line is actually ignored by Python (as are all lines that begin with `#`) but it is used by the shell to determine how the file should be executed. In this case we use the *env* utility, which should be present on all platforms, to find and arm the Python 2 executable. For this nifty trick to work, you'll need to make the **gomoku.py** file executable, which is achieved from the command prompt (assuming you've copied the file to your home directory, or anywhere you have write permission) with:

```
$ chmod +x gomoku.py
```

You'll find you can now start the game with a more succinct:

```
$ ./gomoku.py
```

Next we have three `import` statements, two of which (`pygame` and `sys`) are straightforward. The **pygame** module makes easy work of doing game-related things – we're really only scratching the surface with some basic graphics and font rendering. We need a single function, `exit()` , from the `sys` module so that we can cleanly shut down the game when we're done. Rather than importing the whole `sys` module we import only this function. The final import line is just for convenience – we have already imported `pygame`, which gives us access to `pygame.locals`, a bunch of constants and variables. We use only those relating to mouse, keyboard and quitting events. Having this line here means we can access, say, any mouse button events with `MOUSEBUTTONDOWN` without prefixing it with `pygame.locals`.



One of many tense counter-based battles which Jonni ultimately won. [*That's cos you were playing both sides – Ed*]

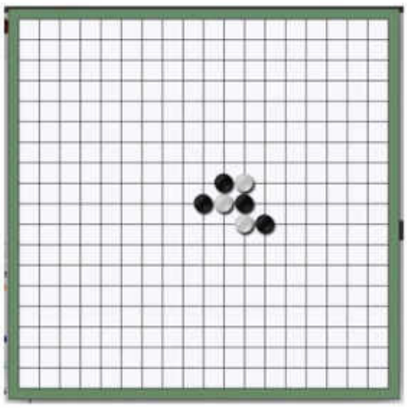
It's all in the (Py)game

Throughout the program you'll notice that some variables are uppercase and some are not. Those in uppercase are either from `pygame.locals` or should be considered constants, things that do not change value over the course of the game. Most of these are declared after the `import` statements and govern things like the size of the game window and counters. If you want to change the counter colours, to red and blue for example, you could replace the values of `WHITE` and `BLACK` with `(255,0,0)` and `(0,0,255)` respectively. These variables are tuples (a similar structure to a list, only it cannot be changed) which dictate the red, green and blue components of colours.

Next you'll see a series of blocks beginning with `def:` – these are function definitions and, as is the case with other codeblocks in Python, they are demarcated by indentation. The `initGame()` function initialises the play area. Here's a simple version that shows what this function does:

```
WIN_WIDTH = 620
GRID_SIZE = (WIN_WIDTH) / 14
WHITE=(255,255,255)
BLACK=(0,0,0)
BGCOL=(80,80,80)
def initGame():
    screen.fill(BGCOL)
    for j in range(15):
        pygame.draw.line(screen, WHITE, (0, j * GRID_SIZE), (WIN_WIDTH, j * GRID_SIZE))
        pygame.draw.line(screen, WHITE, (j * GRID_SIZE, 0), (j * GRID_SIZE, WIN_WIDTH))
pygame.init()
pygame.display.set_caption('LXF Gomoku')
screen = pygame.display.set_mode((WIN_WIDTH,WIN_WIDTH))
initGame()
pygame.display.update()
```

If you add the three import lines to the beginning of this, then this is actually a »



Joel Murielle's graphical *Gomoku* is available from the *Pygame* website. *Pygame* takes all the pain out of working with sprites, notorious troublemakers.

perfectly valid Python program. The `initGame()` function doesn't do anything until it is called at the last line, by which time we've already initialised *Pygame*, set our window title and set the window size to 620 pixels. All variables set up outside of function definitions, hence the five all-caps constants at the beginning and `screen` are accessible inside function definitions; they are known as global variables. Variables defined inside function definitions are called 'local' – they cease to exist when the function exits, even if they have the same name as a global variable – again, something to be aware of in your future coding endeavours. The variable `screen` refers to the 'canvas' on which our game will be drawn, so it will be used extensively later on. The `initGame()` function's first act is to paint this canvas a delightful shade of grey (which you're very welcome to change). Then we use a loop to draw horizontal and then vertical lines, making our 15x15 grid. None of this artwork will appear until we tell *Pygame* to update the display, hence the last line.

Astute readers will notice that the grid overruns ever so slightly at the edges. This is because drawing 15 equispaced parallel lines divides the board into 14, but 620 (our window size) is not divisible by 14. However, when we add in some window borders, since we want to place counters on the edge lines as well, 620 turns out to be a very good number, and we were too lazy to change it. Though rough around the edges, it's still testament to *Pygame's* power and Python's simplicity that we can do all this in just a few lines of code. Still, let's not get ahead of ourselves, our game still doesn't do anything.

Finer points

From here onward, we'll refer to the actual code, so any snippets we quote won't work in isolation – they're just there to highlight things. You'll notice that the `FONT` variable isn't defined with the other constants: this is because we can't use *Pygame's* font support until after the *Pygame's* `init()` method has been called. Let's look at the main game loop right at the end of the code. The introductory clause `while True:` suggests that this loop will go on forever. This is largely correct – we want to keep checking for events, namely mouse clicks or the user clicking the exit button, until the game is done. Obviously we exit the loop when the application quits – clicking the button triggers a `QUIT` event which we react to with the `exit()` function from the `sys` package. Inside the main loop, the first thing we do is call the `updatePlayer()` function, which you'll find on line 32. This updates the text at the top of the screen that says whose go it is, drawing ('blitting') first a solid rectangle so any previous text is erased. Next we loop over the events in the events queue; when the player tries to make a move, the `tryCounterPlace()` function is called, with the mouse co-ordinates passed along.

To keep track of the game, we use a two-dimensional square array (a list of lists) in the variable `grid`. This is initialised as all 0s, and when a player makes a move a 1 or a 2 is entered accordingly. The first job of the `tryCounterPlace()` function is to reconcile the mouse coordinates where the user clicked with a pair of coordinates with which to index the grid variable. Of course, the user may not click exactly on an intersection, so we need to do some cheeky rounding here. If the player clicks outside of the grid (e.g. if they click too far above the grid, so the `y` coordinate will be negative) then the function returns to the main loop. Otherwise we check that the grid position is unoccupied and if so draw a circle there, and update our state array `grid`. A successful move causes our function to return a `True` value, so looking at line 111 in the code we see this causes the next player's turn. But before that is enacted, by the `updatePlayer()` call at the top of the loop, we call the `checkLines()` function to see if the latest move completed a winning line. You'll find details of how this check is carried out in the box.

When a winning counter is detected by our state-of-the-art detection algorithm, the `winner()` function is invoked. This replaces the text at the top of the screen with a message announcing the victor, and the gameover loop is triggered. This waits for a player to push R to restart or rage quit. If a restart is ordered, then the player order is preserved and, since this is updated immediately before `checkLines()` is called, the result is that the loser gets to start the next round.

This is a small project (only about 120 lines, not really a match for the 487 byte *Bootchess* you can read about at **www.bbc.co.uk/news/technology-31028787**), but could be extended in many ways. Graphics could be added, likewise a network play mode and, perhaps most ambitiously, some rudimentary AI could be employed to make a single player mode. This latter has in fact already been done...

Reading between the lines

Part of Key Stage 2 involves learning to understand and program simple algorithms. We've already covered our basic gameflow algorithm – wait for a mouseclick (or for the user to quit), check if that's a valid move, check if there's a line of five, etc. At the heart of that last stage, lies a naïve, but nonetheless relevant, algorithm for detecting whether a move is a winning one.

Consider the simpler case where we're interested only in horizontal lines. Then we would loop over first the rows and then the columns of our grid array. For each element we would check to see that it is non-zero (i.e. there is a counter there) and if the four elements to its right have the same value. In Python it would look like this:

```
for j in range(15):
    for k in range(10):
        pl = grid[j][k]
        if pl > 0:
            idx = k
            while grid[j][idx] == pl and idx < 14:
                idx += 1
            if idx - k >= 5:
                # game winning stuff goes here
```

Note that the inner loop variable `k` reaches a maximum value of only 9. We do not need to check row positions further right than this since our algorithm will reach out to those positions if a potential line exists there. Our variable `idx` effectively measures the length of any line; it is incremented using the `+=` operator, short for `idx = idx + 1`.

The algorithm is easily adapted to cover vertical and diagonal lines. Rather than having four separate functions, though, we've been clever and made a general function `lineCheck()`, which we call four times with the parameters necessary for each type of line checking. Said parameters just change the limits of the `for` loops and how to increment or decrement grid positions for each line direction.

Languages: an overview

One of technology's greatest achievements was IBM's Fortran compiler back in the 1950s. It allowed computers to be programmed using something a bit less awkward than machine code. Fortran is still widely used today and, while some scoff at this dinosaur, it remains highly relevant, particularly for scientific computing. That said, nobody is going to start learning it out of choice, and there are all manner of other languages out there.

Traditionally, you have had the choice between hard and fast languages – such as Java, C and C++ – or easy and slower ones, like Python or PHP. The fast languages tend to be the compiled ones, where code has to be compiled to machine code before it can be run. Dynamic languages are converted to machine code on the fly. However, on some level all programming languages are the same – there are some basic constructs such as loops, conditionals and functions, and what makes a programming language is simply how it dresses these up.

For those just starting coding, it's simply baffling. Opinions are polarised on what is the best language to learn first of all, but the truth is that there isn't one, though for very small people we heartily recommend Scratch. Any language you try will by turns impress and infuriate you. That said, we'd probably not recommend C or Haskell for beginners.

There is a lot of popular opinion that favours Python, which we happily endorse, but then many are put off by the Python 2 versus 3 fragmentation Python has a lot going for it: it's probably one of the most human-readable languages out there. For example, you should, for readability purposes, use indentation in your code, but in Python it's mandatory. By forcing this issue, Python can do away with the curly brackets used by so many other languages for containment purposes. Likewise there's no need to put semicolons at the end of every line. Python has a huge number of extra modules available, too – we've already seen *Pygame* and our favourite, the API for programming *Minecraft* on the Pi.

Beginner-friendly

Other languages suitable for beginners are JavaScript and PHP. The popularity of these comes largely from their use on the web. JavaScript works client-side (all the work is done by the web browser) whereas PHP is server-side. So if you're interested in programming for the web, either of these will serve you well. You'll also want to learn some basic HTML and probably CSS too, so that you can make your program's output look nice, but this is surprisingly easy to pick up as you go along. PHP is cosmetically a little messier than Python, but soon (as in The Matrix) you'll see right through the brackets and dollar signs. It's also worth mentioning Ruby in the accessible languages category. It was born of creator Yukihiro Matsumot's desire to have something as "powerful as Perl, and more object-oriented" than Python.

> **"Although any language will by turns impress and infuriate you, Python has a lot going for it."**

Barely a day goes by without hearing about some sort of buffer overflow or use-after-free issue with some popular piece of software. Just have a look at **https://exploit-db.com**. All of these boil down to coding errors, but some are easier to spot than others. One of the problems with the fast languages is that they are not memory safe. The programmer is required to allocate memory as it is required and free it when it is no longer required. Failure to do this means that programs can be coerced into doing things they should not do. Unfortunately, 40 years of widespread C usage have told us that this is not a task at which humans excel, nor do we seem to be getting any better at it. Informed by our poor record here, a new generation of languages is emerging. We have seen languages contributed from Google (Go), Apple (Swift) and Mozilla (Rust). These languages all aim to be comparable in speed to C, but at the same time guaranteeing the memory safety so needed in this world rife with malicious actors.

Rust recently celebrated its 1.0 release and maybe one day *Firefox* will be written using it, but for now there are a number of quirks and pitfalls that users of traditional languages are likely to find jarring. For one thing, a program that is ultimately fine may simply refuse to compile. Rust's compiler aims for consistency rather than completeness – everything it can compile is largely guaranteed, but it won't compile things where any shadow of doubt exists, even if that shadow is in the computer's imagination. So coders will have to jump through some hoops, but the rewards are there – besides memory safety and type inference, Rust also excels at concurrency (multiple threads and processes), guaranteeing thread safety and freedom from race conditions. »

Programming paradigms and parlance

With imperative programming the order of execution is largely fixed, so that everything happens sequentially. Our Gomoku example is done largely imperative style, but our use of functions makes it more procedural – execution will jump between functions, but there is still a consistent flow. The object oriented (OO) approach extends this even further. OO programs define classes, which can be instantiated many times; each class is a template for an object, which can have its own variables (attributes) and its own code (methods). This makes some things easier, particularly sharing data without resorting to lengthy function calls or messy global variables. It also effects a performance toll, though, and is quite tricky to get your head around. Very few languages are purely OO, although Ruby and Scala are exceptions. C++ and Java support some procedural elements, but these are in the minority.

Functional programming (FP) has its roots in logic, and is not for the faint-hearted. This very abstract style is one in which programs emphasise more what they want to do than how they want to do it. Functional programming is all about being free of side-effects – functions return only new values, there are no global variables. This makes for more consistent languages such as Lisp, Scheme, Haskell and Clojure. For a long time FP was traditonally the preserve of academia, but it's now popular with industry.

Coding in the classrooms

In September 2014, the UK embarked on a trailblazing effort which saw coding instilled in the National Curriculum. When the initiative was announced in 2013, then education secretary Michael Gove acknowledged that the current ICT curriculum was obsolete – "about as much use as teaching children to send a telex or travel in a zeppelin". Far more important was imparting coding wisdom unto the young padewans. Coding skills are much sought-after, as evidenced by industry consistently reporting difficulties in finding suitably qualified applicants for tech jobs n the UK.

The FUZE box gives you everything you need to start fiddling with registers or making GPIO-based mischief.

The 'Year of Code' was launched to much fanfare, though this was slightly quelled as details emerged: a mere pittance was to be added to the existing ICT allocation, and most of this would be spent on training a mere 400 'Master Teachers' who could then pass their Master Skills to lesser teachers around the country. Fans of shadenfreude will enjoy the BBC Newsnight interview with the then Year of Code chief, wherein she openly admits not knowing how to code, despite claiming it is vital for understanding how the world works.

Learning opportunities

Criticism and mockery aside, we're genuinely thrilled that children as young as 5 are, even as we speak, learning the dark arts of syntax, semantics and symbolism. Fear now, ye parents, when your progeny hollers at you (that's what kids do, apparently) "What's the plural of mongoose?" and then follows through seeking clarification on the finer points of recursion and abstraction. Rightly or wrongly, many private firms will benefit from the concerned parents and confused kids resulting from the new computing curriculum. But there are more wholesome directions in which one can seek help.

> **"Without any coding, the ICT curriculum was 'as much use as teaching kids to send a telex'."**

For one thing, you'll always find great tutorials monthly in Linux Format – just look again at the previous five pages, such exceptional erudition. There are also many free resources on the web. Some of them (such as the official Python Documentation) are a little dry for kids, but we'd encourage adults to learn these skills alongside their offspring. Refurbishing an old machine with a clean Linux install will provide a great platform to do just this. All the software you need is free, and distributions like Mint and Ubuntu are easy enough to get to grips with.

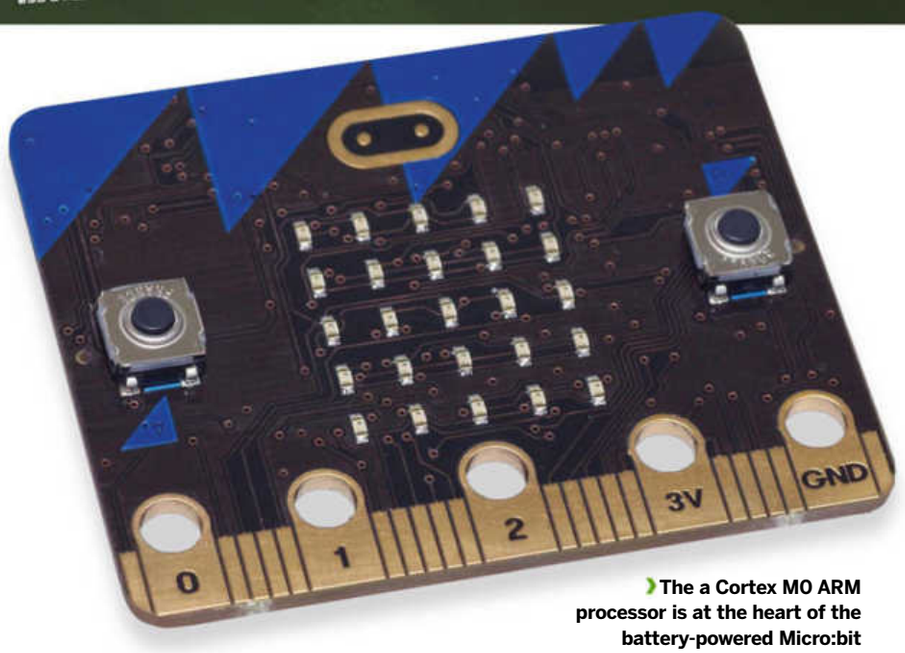
Besides a stray PC, the Raspberry Pi is another great way to provide a learning platform. If you're willing to settle for the older B+ model, then you can get one for about £25, and it'll plug straight into your telly. Of course, you'll need to scavenge a keyboard, mouse, SD card, HDMI cable and possibly a wireless adapter too, if trailing an Ethernet cable to your router is not an option. Mercifully, there are many kits available (for example the Kano, billed as a DIY computer) which will provide these additional peripherals, not to mention the glorious array of Pi cases available to protect it from dust and bumps. We particularly like setups such as the FUZE box, which embed the Pi into a more traditional, albeit chunkier, all-in-one device. Yes, the Pi's tiny form factor is appealing, but with a heavy rotation of USB devices thrown in, it's all too easy to end up in cable-induced tiny hell.

Speaking of tiny things, to coincide with the new learning regimen the BBC will distribute about a million 'Micro: bit' computers to Year 7 pupils. These are even smaller than the Pi, but have no means of output besides a 5x5 LED array. Unlike the Pi, then, they cannot function as standalone computers, requiring instead to be programmed from a more capable device.

Microsoft is generously providing the software and training for this venture, and we are relieved to hear they will not tie the product to their nascent Windows 10 platform, an unfinished beta of which was released at the end of July. The devices have gravity and motion sensors, as well as Bluetooth and some buttons. There are five GPIO rings that could connect further sensors or contraptions. They can be programmed in a number of languages including C++, Python, JavaScript and Blocks – a visual programming language. The Microsoft-provided code editors are all web based, and code entered here must be compiled before being downloaded to the Micro:bit. At present, amidst the pre-launch hush, details are sketchy, but it seems like this compilation all takes place on Microsoft servers. It'll be disappointing if the code editors can't be used offline or no option to compile locally is offered. You can find out more about it at **www.bbc.co.uk/mediacentre/mediapacks/microbit**

Micro:bit spokesbods have been keen to stress that device is in no way intended to compete with the Raspberry Pi, but rather to complement it. The Micro:bit features a 20-pin edge connector which connects it to the Pi or another device so that the machines can work in tandem. Such a pairing will be necessary for the Micro:bit to have

means of communicating with the outside world. The device – indeed the combination of the device and the Pi – has a great deal of potential but there exists some skepticism over whether anything like the excitement generated by the 1982 launch of BBC Micro will be seen again. Back then computers were new and exciting, while these days kids expect them to provide a constant barrage of entertainment in the form of six-second cat videos or 140-character social commentary. Not all of them are going to be thrilled at having to program them. But anything that lowers, if not entirely obliterates, any entry barrier to getting into coding is fine by us. We also look forward to ne'er-do-well students hacking each other's Micro:bits or engaging them in a collaborative DDOS attack on their school's infrastructure.



The a Cortex M0 ARM processor is at the heart of the battery-powered Micro:bit

Get with the program

The syllabus comprises three Key Stages. The first, for 5-6 year olds, covers algorithms in a very general sense. Algorithms will be described in terms of recipes and schedules, to introduce the idea of formalising instructions. The second stage (ages 7-11) introduces core ideas such as loops and variables. Alongside this, candidates will be learning to use web services and how to gather data. The final stage, for secondary students aged 11-14, requires students to learn at least two programming languages and understand binary arithmetic, Boolean algebra, functions and datatypes. Students will also touch on information theory, at least as far as how different types of information can all be represented as a string of bits. They will also gain insights into the relationship between hardware and software.

Throughout the curriculum, students will also learn the vital skills of online privacy and information security – skills the want of which has led to many an embarrassing corporate or governmental blunder. All in all, it's a highly ambitious project, but perhaps such a radical step is necessary to address the skills shortage in this area. With luck the scheme will also lead to much-needed diversification among the coding populace. If it works out, and pupils are learning Python alongside Mandarin or studying Kohonen or Knuth alongside Kant, then we'll be thrilled. »

Code Clubs

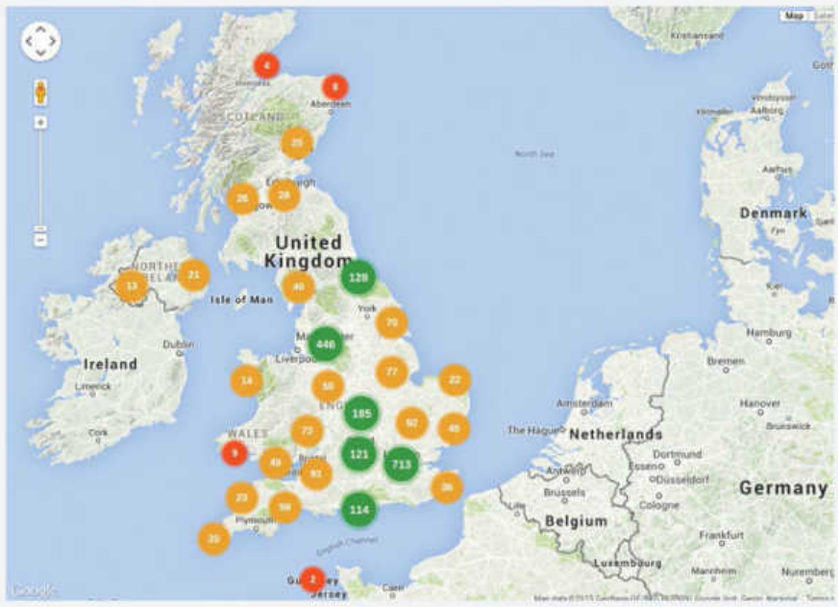
</code>

There are also over 2,000 volunteer-run code clubs across the UK. Code Club, armed with £100,000 courtesy of Google, provides the material, and schools (or other kind venues) provide space and resources for this noble extracurricular activity.

The Code Club syllabus is aimed at 9-11 year olds and consists of two terms of Scratch programming, then a term of web design (HTML and CSS), concluding with a final term of grown-up Python coding. Projects are carefully designed to keep kids interested: they'll be catching ghosts and racing boats in Scratch, and doing the funky Turtle and keeping tabs on the latest Pokémon creatures in Python, to name but a few.

If you have the time and some knowledge, it's well worth volunteering as an instructor. All the tutorials are prepared for you and if you can persuade a local primary school to host you, then you'll get a teacher to maintain order and provide defence against any potential pranksters. You will require a background check, though. Code Club also provide three specialist modules for teachers whose duty it is to teach the new Computing curriculum.

This kind of community thinking is very much in the spirit of open source, providing an access-for-all gateway to coding free from commercial interest and corporate control. The Code Club phenomenon is spreading worldwide too. You'll now find them as far afield as Bahrain and Iceland. The project materials have already been translated into 14 languages with more on the way. You can find out more at **www.codeclub.org.uk**



PHP: Code and calculate

PHP used to stand for Personal Home Page, but these days it's a recursive acronym for PHP: Hypertext Processor. Many of the world's leading web applications, such as WordPress and OwnCloud, are written in PHP. It's used extensively by some of the biggest websites in the world, some of which are so big (Facebook) that they had to write their own engine (HHVM) for it. Rather than using PHP in a traditional setting (running as a module through a webserver), let's install PHP-CLI, the command line edition. This saves us the trouble of setting up Apache or some such (although that's pretty easy nowadays, so you may want to do the tutorial this way and that is fine).

On Debian-based distributions and their derivatives, you'll want to run:

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

Other distros will name their packages differently – on Arch it's called just `php`. Ultimately, if you find your package manager starts trying to install a webserver and all sorts of other gubbins, then you chose the wrong package.

We'll work in the text editor of your choice – you could use `nano` and do the whole thing from the terminal, or you can use something graphical like *IDLE* or *Geany*. Whatever you choose, your first program is going to be short – this short:

```
<?php
echo 'Hello World!';
?>
```

Save this as `hello.php` and then run it with the following command:

```
$ php hello.php
```

You probably knew what was going to happen. Similar to how JavaScript is enclosed in HTML pages via `<script>` tags, PHP code has to be surrounded by the tags on the first and last lines of the snippet there. Codeblocks – which include but are not limited to variable assignments, loops and **if...else..endif** clauses – must be terminated with a semicolon (that most roguish of punctuation marks). If you were to embed the code above in an HTML document on a PHP-enabled webserver, then it would perform its worldly greeting on that webpage. Viewing the page's source through the browser would not show the PHP code, only its output... Spooky.

Let's try a more complicated example. This time we're going to define a function called `fib` which takes an integer `$n` as an argument and returns the `$n`th Fibonacci number. Variables in PHP are prefixed with dollar signs, functions are wrapped in curly brackets and `if` conditions in regular brackets:

```
<?php
function fib($n) {
    if ($n == 0) {
        return 0;
    }
    if ($n == 1) {
        return 1;
    }
    return fib($n - 1) + fib ($n - 2);
}

echo fib(12);
?>
```

Even without any coding experience or exposure to the Fibonacci sequence you'll probably figure out that the 'zeroth' Fibonacci number is 0, and the next one is 1. Subsequent Fibonacci numbers are defined as the sum of their two predecessors, hence the rest of the sequence goes 1,1,2,3,5,8. Amazingly, coding this requires next to no effort – it is perfectly legal to call the function from within itself, and it won't even break. This technique is known as recursion and is vaguely illustrated by the cornflakes packet on which you'll find a smaller picture of the cornflakes packet, which has on it... Maybe not quite like that – thanks to our two base cases the recursion depth is limited, so we won't end up in some infinite depth trap. Be that as it may, this is not the most efficient way to compute Fibonacci numbers. Look at the figure to see the call structure for `fib(5)` – more than half the calls are superfluous. As `n` grows, so too does this duplication of effort: see what happens when you change the function call in the second last line to calculate the 40th Fibonacci number (it's about 100 million, and takes us about 45s to compute). Check out the box opposite for a fix.

> **"Many of the world's leading web apps, like WordPress and OwnCloud, are written in PHP."**

Note that if the user doesn't supply an argument to `fib2.php` (from said box), then PHP will complain since the array `$argv` does not have an entry at index 1. The value of `$argv[0]` is always the name of the program itself, in this case `fib2.php`. Where an argument is provided, we convert it from a string to an int. This converts unusable input to the int 0, which triggers our unhelpful error message. Unfortunately this means our program doesn't work for the zeroth Fibonacci number, but that one's easy to remember. Anything bigger than zero can be found remarkably quickly using our new algorithm. Well, not quite anything – the 93rd Fibonacci number is greater than 2^64, so cannot be stored as a 64-bit integer. PHP rounds this to a floating point number (displayed using exponent-mantissa notation) with limited precision, but eventually (1,477 on our setup) the numbers become too big even for that.

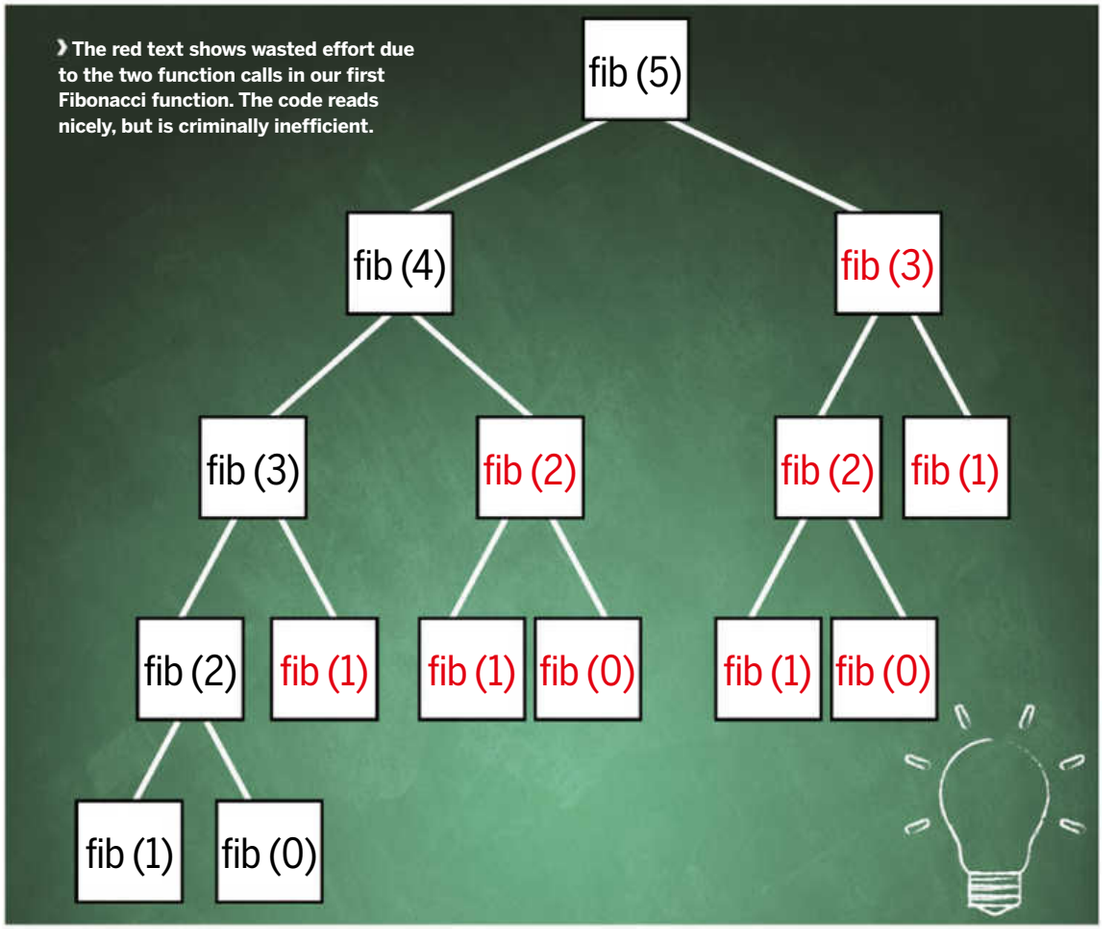
Perhaps it's worth noting that there's an inherent overhead in every layer of recursion and function call you add to your program. So if one really wanted to be Teutonically efficient about it, an iterative solution would be the best one. There's no perceivable speed gains in this case (`fib2 1476` displays an answer almost immediately), so we leave this as an exercise. You'll need to know about **for** loops in PHP, which is good because they feature in the next section.

Caesar cipher

To finish off today's lesson, we're going to use some classic cryptography. Be sure to revisit our glorious treatise on this subject from **LXF189**, but it will suffice to know that Julius Caesar used to encrypt messages by shifting each letter three places down the alphabet, wrapping X, Y and Z to A, B and C respectively. This is slightly tricky to achieve on a computer. One way is to look at the ASCII values of each character of plaintext: uppercase letters A-Z have values 65-90, and lowercase a-z occupy the range 97-122. For simplicity we're going to leave all other characters (e.g. spaces and punctuation) untouched in the ciphertext. So we will loop over `$plaintext` character by character, add the value shift to the ASCII value of each alphabetic character, perform any appropriate wraparounds, get the character represented by this new value and finally append this to our ciphertext.

Once you get that finding the ASCII value is done with the function `$ord` and the character with `$chr` :

```
<?php
function ccipher($plaintext, $shift) {
        $ciphertext = '';
        for ($j = 0; $j < strlen($plaintext);
$j++) {
            $asciicode = ord($plaintext[$j]);
```



The red text shows wasted effort due to the two function calls in our first Fibonacci function. The code reads nicely, but is criminally inefficient.

Quick tip

Change the value of `$shift` to make other equally useless ciphers – a value of 13 will give you the ROT13 code popular for hiding punchlines and spoilers on many forums.

```
                if ($asciicode >= 65 &&
$asciicode <= 90) {
                    $ciphertext .=
chr(($asciicode + $shift + 13) % 26 + 65);
                }
                else if ($asciicode >= 97
&& $asciicode <= 122) {
                    $ciphertext .=
chr(($asciicode + $shift + 7) % 26 + 97);
                }
                else {
                    $ciphertext .=
chr($asciicode);
                }
            }
            return $ciphertext;
}
$plaintext = 'Veni vidi vinci';
$shift = 3;
echo ccipher($plaintext, $shift);
?>
```

For loops are initiated C-style, giving a list of conditions in brackets. Our iterator variable `$j` starts at 0 and goes all the way to the length of our string. The `and` operator is abbreviated to `&&` and we use the `.=` operator to concatenate strings.

All pretty straightforward, except for the wraparound part. Here we use the modulus operator `%`, which gives you the remainder on division, in our case by 26. In the first case we want this to be 0 if our shifted value is 65, so we add on 13, since 65 + 13 = 78, which is divisible by 26. Dually we add on 7 in the second block. There isn't any room left to write the corresponding `cdecipher()` function, but it's just a question of changing two plus signs to minus signs. See if you can recover the original plaintext this way.

Faster Fibonacci function

Programmers are very keen on writing efficient code. This is not just because they like elegance (though as a rule they do); as we've mentioned, inefficient code might be fine to begin with but can start running as slow as treacle when asked to deal with heavier loads. Here we can use a smarter Fibonacci function which passes the two previous numbers in the sequence, so that no effort is wasted. But we still want to be able to call our function with a single argument, so we'll make these extra arguments optional, setting their defaults to 1 and 0:

```
<?php
function fib2($n, $oldfib = 1, $olderfib = 0) {
    if ($n > 1) {
        return fib2($n - 1, $oldfib + $olderfib,
$oldfib);
    }
    else {
        return $oldfib;
    }
}
if (!$argv[1]) {
    echo 'no';
}
else{
    echo fib2((int) $argv[1]);
}
?>
```

Besides defining our new improved function `fib2()` we also use the special variable `$argv` to allow the user to provide input on the command line. If one saved this file as `fib2.php`, then the 54th Fibonacci number could be found thusly:

```
$ php fib2.php 54
86267571272
```

Scripting languages

Let's go beyond Bash to see which scripting languages measure up to the needs and wants of a Linux system administrator.

How we tested...

Comparisons, they say, are invidious. This is certainly true for programming languages, where personality and local support are, at least, of equal import to criteria such as speed, and the level of support for different paradigms. Given this, we're presenting a mixture of facts, collective opinions and our own prejudices, but it's a basis for further investigation. The key to a scripting language's usefulness to the sysadmin lies not just in how easily it helps solve problems, but in how many of the solutions have already been written, and are available to download and adapt, and preferably well-documented.

We tried to work across the range of versions installed on a typical network, but insisted on Python 3. Other than that, we've tried to stay in the context of working with what you're likely to find on your network.

Every admin loves time-saving shortcuts, and carries a selection of scripts from job to job, as well as inheriting new ones when arriving in post. The question any new admin asks is which is the best language to learn? (Followed by, where's the coffee?) Veterans of language wars should know that the best language question rarely has a simple or definitive answer, but we thought it would be well worth comparing the most useful choices to make your Linux life easier.

Most scripting languages have been around longer than you think. For example, NewLISP was started on a Sun-4 workstation in 1991. They've borrowed from each other, and elsewhere, and accumulated a long legacy of obsolete libraries and workarounds. Perl's Regular Expressions, for instance, are now found everywhere, and in some cases better implemented elsewhere. So what matters most? How fast the script runs, or how quickly you can write it? In most cases, the latter. Once up and running, support is needed both from libraries or modules to extend the language into all areas of your work, and from a large enough community to support the language, help it keep up with trends, and even to innovate it. So, which scripting language should you learn to improve your Linux life this year?

> **"The question any new admin asks is which is the best language to learn?"**

The learning curve

Online resources, books and good people.

The key questions are: how easy is the language to pick up? Are the learning resources at least adequate? Even if these two questions are answered in the positive, they still need to be backed up by a helpful community to assist you in quickly producing something useful, and help maintain that initial enthusiasm as you hit inevitable problems.

To produce a backup script and test scripts in each of the languages, we started by browsing Stack Overflow. But downloading random code means no consistency between Posix (pure Bourne Shell) scripts, modern Bash, and legacy code that occasionally fails. Fortunately, **www.shellcheck.net** is a great tool for checking the correctness of scripts, and teaches you best practice as it corrects them. The Linux Document Project's (perhaps overly) comprehensive Advanced Bash Scripting Guide (**www.tldp.org/LDP/abs/html**) is also excellent and will help you quickly gain confidence.

Perl's online and built-in documentation is legendary, but we started by running through an exercise from the classic O'Reilly admin book, *Running Linux*, then leapfrogged the decades to No Starch's recent *Perl One-Liners* by Peteris Krumins. Those who eschew the book form should try **http://perlmonks.org**, a source of cumulative community wisdom.

Recent efforts at getting youngsters learning through Code Club (**www.codingclub.co.uk**) and the rest of us through PyConUK education sprints and open data hackdays have shown Python to be easily picked up by anyone. But out-of-date advice, such as the many ways of running subprocesses which persist for compatibility reasons, means careful reading is needed, and it's yet another good reason for starting with Python 3, not Python 2. Head to **www.python.org/about/gettingstarted** for large list of free guides and resources.

Ruby is also an easy sell to learners, and before Rails, command-line apps were what it did best. David B. Copeland's book, *Build Awesome Command Line Applications in Ruby* will save you hours of wading through online documentation, but we were able to get up and running on our test scripts with a couple of web tutorials.

Last, we come to NewLISP: a challenge to programmers schooled only in non-LISP family languages, but you'll be amazed by what it manages to accomplish with just lists, functions and symbols. We dived right in with the code snippets page on **http://newlisp.org**, adapting to build our backup script, and were rewarded with terse, powerful code, that was easier to read than its equally compact Perl counterpart.

From MOOCs to the bookshop, Python learning resources are everywhere.

Verdict

Bash ★★★★☆
NewLISP ★★★★☆
Perl 5 ★★★★☆
Python ★★★★★
Ruby ★★★★★

Python and Ruby are easier to learn, because of good docs and helpful users.

Version and compatibility

Beating the wrong version blues.

The question here is: have I got the right version? Lets start with *Bash*. Every modern Linux distro ships with a version that will run your scripts and anyone else's. *Bash 4*, with its associative arrays, coproc (two parallel processes communicating), and recursive matching through globbing (using ** to expand filenames) appeared six years ago. *Bash 4.2* added little, and is four years old and *Bash 4.3*'s changes were slight.

```
#------------------------
print "\n${l_hlt}"  . " "x${l_columns}  . "${l_rst}\r"
    . "${l_hlt}  ${l_int}BRANCH:${l_rst}${l_hlt} ${l_git_branch}  (${l_git_sha
}  ${l_rst}\n\$ ";
EOF

}

export COLUMNS
PS1="${PS1}\$(Func_GitCheck)"
Func_GitCheck
richard@luggable:~/work$ bash --version
bash --version
GNU bash, version 4.3.11(1)-release (i686-pc-linux-gnu)
Copyright (C) 2013 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>

This is free software; you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
```

As the Unix shell dates back decades, you will find that recent *Bash* versions contain a few unexpected syntax changes.

Perl is still included in the core of most distros. The latest version is 5.20 (with 5.22 soon to appear), but many stable distros ship with 5.18. No matter, you're only missing out on tiny improvements, and just about every script you'd want to write will be fine.

The switch from Python 2 to 3 still catches out the unwary. Run Python 3 if you can and check the documentation if you come unstuck. Python 3.3 is our baseline for Python 3 installs and Python 3.4 didn't add any new syntax features.

Ruby version changes have caused enough problems that painless solutions have appeared, *rvm* enables you to run multiple versions of Ruby, and *bundle* keeps track of the gems you need for each script.

NewLISP's stability and lack of third-party scripts is an advantage here. We cann't, however, guarantee every script will run on the latest versions.

Verdict

Bash ★★★★★
newLISP ★★★★☆
Perl 5 ★★★★☆
Python ★★★☆☆
Ruby ★★★★☆

Ruby's version workaround is good, but Bash's lack of problems is a better result.

Web native scripts

Get your admin scripts moving over HTTP.

Much of a sysadmin's life has migrated to the web, so you'll need a scripting language that has kept pace. We examined both ease of writing our own code, and finding available solutions for doing anything from web interfaces to system stats.

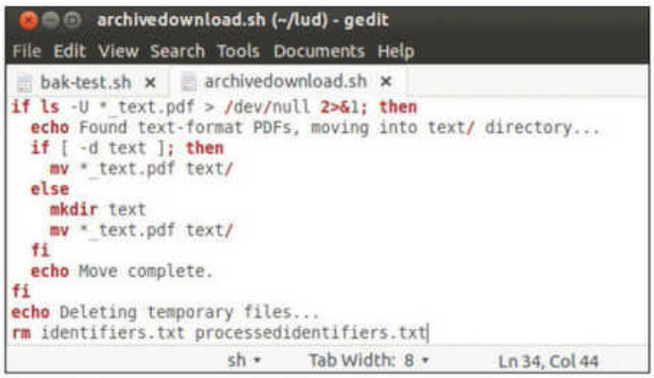
What's noticeable about these languages is the difference in expressiveness and style to produce similar results. However, this is, once again, secondary to personal preference and local support for many admins.

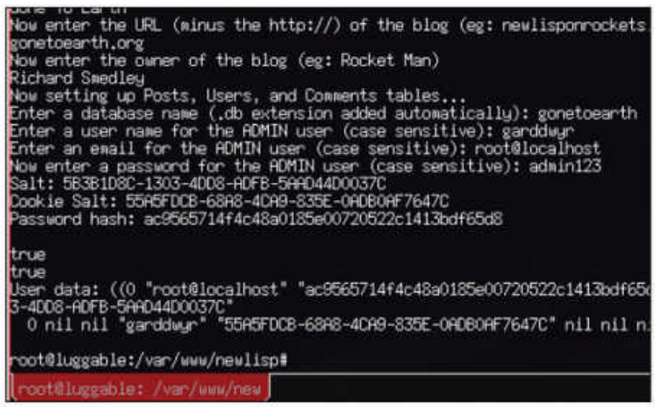
Ruby is quick and enjoyable; Python 'feels right' probably due to it being more human readable; newLISP is astonishingly powerful. But these observations remain partisan clichés without a supportive and maintainable environment to use and develop the code for your own networks.

Bash ★★☆☆☆

While *Bash* will be no one's first choice for a web programming language, it's good to know that when your server doesn't provide for your first choice you can fall back on it thanks to bashlib. This a shell script that makes CGI programming in the *Bash* shell somewhat more tolerable.

Your script will be full of echo statements, interspersed with your commands to produce the desired output. Security considerations mean we wouldn't recommend running this on the open Internet, but it's worth bearing in mind that *Bash* works well as a prototyping language. It's easy to fill a text file with comments describing the broad structure that you want, then fill in the gaps – testing snippets interactively and pasting into **www.shellcheck.net** to check your code as you go. You'll soon be up and running with a proof of concept.





newLISP ★★★★☆

Code Patterns, by NewLISP creator Lutz Mueller, is available on the **www.newlisp.org** website and has chapters on HTTPD and CGI, as well as TCP/IP and UDP communications. If you add in the section on controlling applications, and you'll have everything to get you started.

NewLISP's built-in networking, and simple (or lack of) syntax, makes it surprisingly easy to generate HTML pages of results from, for instance, your monitoring scripts. For a ready built framework, newLISP on Rockets – which uses Bootstrap, jQuery and SQLite – combines rapid application development with good performance.

NewLISP on Rockets provides several functions, from (convert-json-to-list) via (twitter-search) to (display-post-box), which will help you add web functionality. We're impressed but we remain concerned by the small size of the community and the intermittent pace of development.

Community support

Does it have a community large enough to support real work.

DevOps, cloud deployment, test-driven development and continuous integration – the demands on a sysadmin change and evolve, but the requirement to learn something new is constant. Everyone uses *Bash* to some extent but, you'll need to learn *Bash plus* one other.

Perl was the traditional Swiss Army chainsaw of Unix admins through the '80s and '90s, gradually losing ground to Python and then Ruby over the last decade or so. Anyone who started work in the '90s or earlier will be comfortable with it, so finding someone to help with your scripts is often not a problem.

However, the world doesn't stand still, and many tech businesses have standardised on Python, which is used extensively at Google, for example. Much of the software necessary for modern sysadmin work is Python based although the same can be said of Ruby.

Ruby benefits from being the basis of Chef and Puppet, as well as Vagrant and Travis CI, meaning a little familiarity will be helpful anywhere that uses them for deployment. The web frameworks and testing tools written in Ruby have popularised the language at many of the younger web companies.

NewLISP has a much smaller community supporting it, and there aren't many ready made solutions and you may know no-one who uses it. The keenness of the online community goes some way to ameliorate this deficiency, but you have to ask who will maintain your tools when you leave a company?

Verdict

Bash ★★★☆☆

NewLISP ★★☆☆☆

Perl 5 ★★★★☆

Python ★★★★☆

Ruby ★★★★★

» *Ruby has taken mindshare, thanks to some great DevOps software.*

Perl 5 ★★★★☆

Perl was the first web CGI scripting language and has more or less kept pace with the times. It certainly has the libraries, and enough examples to learn from, but with no dominant solution you'll have to pick carefully.

Catalyst, Dancer, and Mojolicious are all good web application frameworks. More likely you'll find everything you need in CPAN. You can glue together a few of the libraries – many of which are already collected together in distros – to handle a pipeline of tasks, such as retrieving XML data, converting the data to PDF files and indexing it on a web page.

Perl's traditional CGI interface is still available, and despite better performing alternatives abstracted through PSGI, you may find that **use CGI;** is all you need to web-enable your script, and remember: 'there's always more than one way to do it'.

```
if ($q->param()) {
    # Parameters are defined, therefore the form has been submitted
    display_results($q);
} else {
    # We're here for the first time, display the form
    output_form($q);
}

# Output footer and end html
output_end($q);

exit 0;

#---------------------------------------------------------------

sub output_top {
    my ($q) = @_;
    print $q->start_html(
        -title => 'Back-up selection',
```

```
                    filename, _, _ = related_attachment
                    content = re.sub(r'(?<!cid:)%s' % re.escape(fil
%s' % filename, content)
                self.alternatives[i] = (content, mimetype)

        return super(EmailMultiRelated, self)._create_alternatives(msg)

    def _create_related_attachments(self, msg):
        encoding = self.encoding v settings.DEFAULT_CHARSET
        if self.related_attachments:
            body_msg = msg
            msg = SafeMIMEMultipart(_subtype=self.related_subtype, enco
            if self.body:
                msg.attach(body_msg)
            for related_attachment ∈ self.related_attachments:
                if isinstance(related_attachment, MIMEBase):
                    msg.attach(related_attachment)
                else:
* backup-reporter.py:61:0 Python
```

Python ★★★★★

Python's Web Server Gateway Interface (WSGI), which was defined in PEP 333, abstracts away the web server interface, while WSGI libraries deal with session management, authentication and almost any other problem you'd wish to be tackled by middleware. Python also has plenty of full-stack web frameworks, such as Django, TurboGears and Pylons. Like Rails, for some purposes you may be better off coding web functionality onto an existing script. But Python's template engines will save you from generating a mess of mixed HTML and Python.

Python has many other advantages, from the Google App Engine cloud with its own Python interpreter, which works with any WSGI-compatible web application framework, for testing of scalable applications to supporting a clean style of metaprogramming.

Ruby ★★★★★

Don't imagine for one moment that Rails is a panacea for most sysadmin problems. It's not. And while Sinatra certainly makes it easy to roll out anything web-based in Ruby, even this is overkill for most purposes. That said, Rails does a good job of getting code up quickly and just doesn't drown in all that magic, generated code.

Ruby is ideal for getting any script web-enabled, thanks to gems that are written by thoughtful people who have made sane decisions. Putting a web interface on our backup script, for example, was fun, but distracting as we played with several gems, eg to export reports to Google spreadsheets. Tools like *nanoc*, which generate static HTML from HAML, and some of the reporting gems complement the language's expressiveness, and make adding any functionality to scripts a breeze.

```
log:
development.log

public:
404.html  422.html  500.html  favicon.ico  robots.txt

test:
controllers  fixtures  helpers  integration  mailers  models  t

tmp:
cache  pids  sessions  sockets

vendor:
assets
richard@luggable:~/work/code/ruby/rails/crash/code/blog$
richard@luggable: ~/work/co
```

Programmability

Managing big scripts requires other programming paradigms.

Before reaching 1,000 lines of code, *Bash* scripts become unmanageable. Despite its procedural nature, there are attempts to make an object-orientated (OO) *Bash*. We don't recommend it, we think it's better to modularise. Functional programming (FP) in *Bash* (**http://bit.ly/BashFunsh**) is also impractical.

Perl's bolted on OO won't be to everyone's taste, but does the job. Perl has fully functional closures, and despite syntactical issues, can be persuaded into FP – just don't expect it to be pretty. For that you should wait for Perl 6.

Python is equally happy with imperative, OO and also manages FP. Functions are first class objects but other features are lacking, even if its list comprehension is very good. Mochi, the FP language (**http://bit.ly/FPMochi**), uses an interpreter written in Python 3.

Ruby is designed as a pure OO language, and is perhaps the best since Smalltalk. It can also be persuaded to support a functional style of programming. But to get FP code out of Ruby, you'll have to go so far from best practices that you should be using another language entirely.

This brings us neatly to NewLISP, an elegant and powerful language with all the functional features at your fingertips. NewLISP uses a pseudo OO implementation in the form of functional-object oriented programming (FOOP), but this doesn't mean, however, that it can cut it for real OO programming.

Verdict

Bash ★★☆☆☆
NewLISP ★★★☆☆
Perl 5 ★★★☆☆
Python ★★★★★
Ruby ★★★★

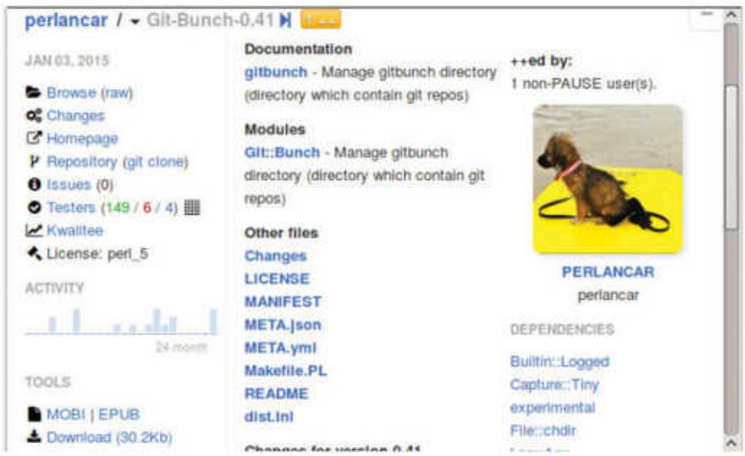
» *Python is a multi-paradigm language and the easiest to maintain.*

Extending the language

Libraries, modules, and getting them working.

None of these scripting languages are as bloated with classes as, for example, Java so that you'll need to use non-core libraries (or modules as they are sometimes called) for writing many scripts. How comprehensive these are, and how easy they are to manage with your script varies greatly.

Perl continues to impress with the mind-boggling choice to be found on CPAN, but its 'there's more than one way to do it' approach can leave you easily overwhelmed. Less obvious, is the magnitude of *Bash* extensions created to solve problems that are perhaps not best suited to any sh implementation.



There's more than one library for that – CPAN is a useful resource for Perl.

Python has excellent library support, with rival choices considered very carefully by the community before being included in the core language. The concern to "do the right thing" is evident in every decision, yet alternate solutions remain within easy reach. At least the full adoption of the *pip* package manager, with Python 3.4, has ensured parity with Ruby.

RubyGems provide the gem distribution format for Ruby libraries and programs, and *Bundler* which manages all of the gems for dependencies and correct versions. Your only problem will be finding the best guide through Ruby's proliferation of libraries. Read around carefully.

NewLisp is not a large language, but it's an expressive one, accomplishing much without the need of add-ons. What modules and libraries that there are address key needs, such as database and web connectivity. There's enough to make NewLISP a useful language for the admin, but not in comparison to the other four choices.

Verdict

Bash ★★★☆☆
NewLISP ★★★☆☆
Perl 5 ★★★★★
Python ★★★★☆
Ruby ★★★★☆

The CPAN's longevity and popularity marries well with good organisation.

Network security

Testing and securing the network – or fixing it afterwards.

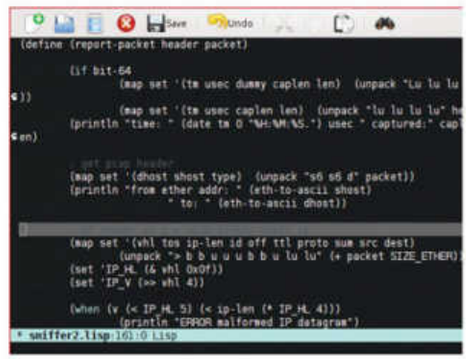
Penetration testing and even forensic examination after an attack will fall under the remit of the hard-pressed sysadmin in smaller organisations. There are enough ready made tools available that you can roll everything you may need into a neat shell script, kept handy for different situations, but writing packet sniffers or tools for a forensic examination of your filesystem in *Bash* isn't a serious option.

Perl has lost some security community mindshare since the early days of Metasploit, but the tools are still there, and are actively maintained by a large user group who aren't about to jump ship to another language. Perl has tools like pWeb – a collection of tools for web application security and vulnerability testing – which is included in distros, such as Kali and Backbox.

Tools such as *Wireshark* are a powerful aide to inspecting packets, but sometimes you'll need to throw together your own packet sniffer. Python not only has Scapy, the packet manipulation library, but provides a socket library for you to easily read and write packets directly.

Ruby's blocks (write functions on-the-fly without naming them) and other features are great for writing asynchronous network code, and its rapid prototyping matches (and even beats) Python. But Ruby's biggest boon is Metasploit, which is the most-used pen-testing software.

In terms of ready rolled tools, you can mix and match as needed, but Perl, Python and Ruby all provide everything you need to quickly examine a network for weaknesses or compromises on-the-fly. Note: Python is featured in more security-related job adverts now.



NewLISP has impressive networking features, even if it lacks the pen-testing tools of the others.

Last, NewLISP isn't well-known among penetration testers and grey hat hackers, but thanks to the networking built in to the language, a function call and a few arguments will create raw packets for pen testing. Once more, NewLISP has clear potential but suffers from its relatively tiny user base.

Verdict

Bash ★☆☆☆☆
NewLISP ★★★☆☆
Perl 5 ★★★★☆
Python ★★★★★
Ruby ★★★★★

Python edges ahead of Ruby and Perl, but all three are friends of the pen tester.

Scripting Languages

The verdict

We admit it's difficult to take the verdict out of a practical context and just declare the best language. For example, *Bash* isn't a strong language, and many time-saving bits of code can be thrown together better with the other four languages, but no-one with tasks to perform at the Linux command line should avoid learning some *Bash* scripting.

Perl is the traditional next step as it's intimately associated with the *nix command line and still found everywhere. It may suffer in comparison with newer languages, but Perl continues to offer not just the Swiss Army Chainsaw of the Linux CLI, but Perl also has a huge and very supportive community.

NewLISP is a pleasant surprise. Yes it has those – Lisp is about lists – but what a compact language for the embedded space as well as the command line. Sadly, the size of the community doesn't match the power of the language, so you'll need to be prepared to back its use with plans to maintain the code yourself.

Python is a powerful, multi-paradigm supporting language. The Python community is large and friendly, and supports everything from education sprints to training teachers. It also backs up community efforts, supporting young learners at Code Clubs, and many other events.

But useful as a community is to the sysadmin, it's often the quick and dirty hacks, readily downloadable and reusable examples, backed by an expressiveness that makes many programming challenges if not trivial, far less of a headache. Rails brought wider attention to Ruby, but Chef, Puppet and Vagrant have helped remind the admin just what can be done with the expressive and eloquent scripting language that was developed by Yukihiro Matsumoto.

```
    private

    def vm_host(vm)
      host_options = {
          user: vm['user'] v 'vagrant',
          hostname: vm['hostname'] v 'localhost',
          port: vm['port'] v '22',
          password: vm['password'] v 'vagrant'
      }

      SSHKit::Host.new(host_options)
    end
  end
end
```

We can't help acknowledging Ruby's power and charms.

Does Ruby edge out Python? Is *Bash* to be ignored? Not for the admin: as they need good knowledge of *Bash* to follow what's going on with the system. And in addition to *Bash*, every sysadmin should know a little Perl, Python *and* Ruby, but have in-depth knowledge of the one that they prefer.

> **"In addition to Bash, every Linux admin should know a little Perl, Python and Ruby."**

1st Ruby ★★★★★

Web: www.ruby-lang.org **Licence:** GPLv2 or 2-clause **Version:** 2.2.0

» Powerful, expressive and very quick to learn.

2nd Python ★★★★★

Web: www.python.org **Licence:** PSFL **Version:** 3.4.2

» Multi-paradigm, encourages good practices and great community.

3rd Perl 5 ★★★★☆

Web: perl.org **Licence:** GPL or Artistic License **Version:** 5.20

» Still a great friend to the sysadmin.

4th newLISP ★★★☆☆

Web: www.newlisp.org **Licence:** GPL **Version:** 10.6.1

» So powerful, it deserves to get more use.

5th Bash ★★★☆☆

Web: www.gnu.org/software/bash **Licence:** GPLv3+ **Version:** 4.3.30

» Doesn't do everything, yet remains essential.

Over to you...

We don't want to start a holy language war, but we would love to hear what you use. Tell Linux Format: **lxf.letters@futurenet.com**.

Also consider...

While *Bash* falls down in some areas, traditional shell scripting could also be represented by Zsh, which has some small but useful differences, such as better access to positional variables, and the ability to extend the shell through widget functions. Nevertheless, it's not a rival to our other choices, and nor is PHP, despite its use in command scripts by some devotees. Instead, our left-field alternative is Rebol (Relative Expression Based Object Language), whose leap towards software freedom two years ago may have come too late to ensure universal popularity. However, Rebol has an elegant design and syntax, and a useful read–eval–print loop (REPL) console.

Rebol's 'dialecting' (using small, efficient, domain languages for code, data and metadata) equips it for just about anything. It's particularly good at dealing with the exchange and interpretation of information between distributed computer systems, but also powerful, terse shell scripts. If you're looking for a new language, give it a try.

Tmux: Terminal multiplexer

Getting script-serious? Do more in the shell and do it all at once...

Tmux is a terminal multiplexer. That term may sound familiar if you have ever used *screen*. A terminal multiplexer enables you to run one or more detachable terminal sessions inside a single terminal or virtual console. Some of the graphical terminals provide tabs, allowing you to run several terminals in the same window, but a multiplexer goes much further. It also doesn't require an X terminal, although it is very happy running inside of one. *Tmux* is probably not installed on your distro by default, but should be in the package repositories, so install it from your distro's software manager in the usual way. Once it's installed, open a terminal or console and run `$ tmux`

Nothing much appears to happen apart from the terminal window clearing and a status bar appearing at the bottom. Run a command, anything that gives some output, say `top`, then press Ctrl+b, then c (that is, press Ctrl+b, release, then press c and release). The window clears, and your previous command appears to be gone. Run another command, then press Ctrl+b, then n and there is your original command back again. Subsequent presses of Ctrl+b, n switch between the two windows, or more if you pressed Ctrl+b, c.

› *Tmux* has a large number of commands available, but thankfully one of them is list-commands.

Much more than tabs

We can hear some muttering that you can do that with tabs in some X terminals, and you can, but *tmux* can do it anywhere you run a shell, and it has some far more powerful tricks up its sleeve. Run something that takes a while to complete – it could be a video transcode with *ffmpeg*, a simple sleep command or the old favourite `$ telnet towel.blinkenlights.nl` then press Ctrl+b, then d and your entire *tmux* session disappears, but it is still running, just detached from your terminal. Open another terminal and run:

```
$ tmux attach
```

and, as if by magic, your previous session reappears with the commands still running.

This is not limited to local terminals either: you can reattach a *tmux* session over an SSH connection. This makes *tmux* especially useful for remote administration. If you want to run a lengthy command from a normal SSH session, you need to stay connected until it completes – especially

Controlling tmux

Tmux is controlled by key commands. Commands are introduced with Ctrl+B followed by the specific command key. You can change this in the configuration file if your muscle memory has been conditioned by *screen*. Common command keys are:

1. **D** disconnect from the session
2. **C** create a new window
3. **N** switch to the next window
4. **P** switch to the previous window
5. **0-9** switch to that numbered window
6. **&** kill the current window
7. **"** split the current pane vertically
8. **%** split the current pane horizontally
9. **o** switch to the next pane
10. **x** kill the current pane
11. Cursor keys move to the pane in the indicated direction
12. **:** enter *tmux* commands manually

problematic if you are mobile. If you run the command inside a *tmux* session, you can detach as soon as it is running and connect again later to see how it is progressing.

If you have a *tmux* session running and you run *tmux* again (without *attach*), you will start a separate session. When you run `tmux attach` it will connect to the first available session, so how do you connect to another? Like so:

```
$ tmux list-sessions
$ tmux attach -t N
```

The first command lists your sessions, each with a number, and the second attaches to session number `N`.

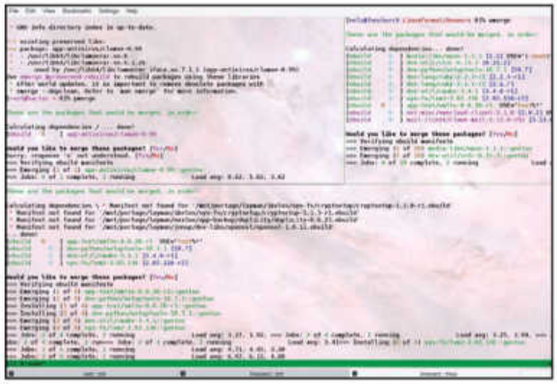
Sessions, Windows and Panes

There is a certain amount of jargon associated with *tmux*. It uses a client/server approach; the first time you start a *tmux* session the server is automatically started, and the *tmux* client then interacts with the server. When run without a following command, it starts a new session. Each *tmux* session is a separate instance and operates independently of any other sessions, although they all run on the same server. Each session contains at least one window, and you create windows with Ctrl+b, c. These windows all exist within the same session, so when you detach and reattach, they are still there. Windows fill the terminal window, so you can see only one at a time – they are listed in the status bar at the bottom of the terminal and you use Ctrl+b, n to switch to the next.

You've seen all of this already, but windows can also be split into panes. Press Ctrl+b followed by the double-quote character, to split a window into two panes, one above the other. Use Ctrl+b, then % for a side-by-side split. Each window contains a single pane when it is created, so you are splitting that pane into two. You can see this in action by pressing the split keys again: only the current pane is split.

If you administer multiple computers, this is a real game-changer. You can have a single terminal window displaying multiple SSH logins at the same time. After a few pane splits, you may find the window gets a little messy with different sized panes. To rearrange them, press Ctrl+b followed by Alt+n where n is any of the keys from 1 to 5, each giving a different layout, you'll use Ctrl+b Alt+5 most often.

Way back we explained how to use *ClusterSSH* to run terminal commands [Tutorials, p74, **LXF179**] and view the output on multiple computers – it opens a small xterm for each host. We can replicate this with *tmux*: press Ctrl+b then the colon (:) to open the *tmux* command line, then type:



One of *tmux's* talents is running a command on multiple computers and viewing the output from all of them.

Tmux for screen users

Moving from *screen* to *tmux* is fairly straightforward, although some of the concepts are approached differently. The most noticeable initial difference is the shortcut key: *screen* uses Ctrl+A. Personally, I think *tmux's* choice is better and it does not conflict with the shell's use of Ctrl+a for "go to the start of the line", so I use Ctrl+b in *screen* too. But if Ctrl+a is ingrained in your muscle memory, put this in **tmux.conf**:

```
unbind C-b
set -g prefix C-a
bind C-a send-prefix
```

If you really don't want to learn any new keystrokes, *tmux* comes with a file called **screen-keys.conf**. Copy this to one of the default locations to get *screen*-compatible key mappings, but bear in mind that it covers only those functions available in both *screen* and *tmux*.

```
setw synchronize-panes
```

Now anything you type goes into each of the panes in that window. This applies to the current window only; any other windows and sessions are unaffected.

Configuring tmux

Tmux looks in two files for configuration settings. Global options are stored in **/etc/tmux.conf** while user settings live in **~/.tmux.conf**. Both files are optional – *tmux* has useful defaults, but if an option is present in both, then the user file takes precedence. These allow you to tweak the behaviour of *tmux*, eg if you want to assign the **synchronize-panes** setting from the above example to a key, you could include this line in one of the config files

```
bind-key S setw synchronize-panes
```

If this key is already defined by default, you will change its behaviour, so it may be best to pick something that is unused. You can see the existing key bindings by entering the command prompt with Ctrl+b, : and running `list-keys`. The full list of commands available is covered in the *tmux* man page; you can test these in the command prompt and then put them in a configuration file to make them permanent.

Direct control

So far we have run *tmux* with no arguments or with `attach` to connect to a session, but there are other options. You can give one or more *tmux* commands some arguments to have them executed when *tmux* starts. Multiple commands are separated by a semi-colon, which must be escaped as the shell also uses it as a command separator, eg:

```
$ tmux new-session \; split-window -h
```

This creates a new session and then splits it into two panes. You can also follow the command by a shell command that is executed in the window or pane, in which case the pane will close once the command exits. Let's go back to the idea of managing multiple SSH sessions in the same window:

```
$ tmux new-session ssh host1 \; split-window ssh host2 \;
split-window ssh host3...\; select-layout tiled \; setw
synchronize-panes
```

That's a fairly long and unwieldy command, but you could add it to a shell alias or a one-line shell script and be able to administer all your computers from a single terminal or console. This highlights one of the strengths of *tmux* for power users: it is well suited to being run and controlled from scripts. There is a lot more to *tmux* – we have barely scratched the surface here – but there is plenty of useful information in the man page and online.

Subscribe to LINUX FORMAT

Get into Linux today!

Choose your LINUX FORMAT package

Print £63 For 12 months

Every issue comes with a 4GB DVD packed full of the hottest distros, apps, games and a lot more.

Digital £45 For 12 months

The cheapest way to get *Linux Format*. Instant access on your iPad, iPhone and Android device.

Bundle £77 For 12 months

- Includes a DVD packed with the best new distros.
- Exclusive access to the *Linux Format* subscribers-only area – with 1,000s of DRM-free tutorials, features and reviews.
- Every new issue in print and on your iOS or Android device.
- Never miss an issue.

LEARN HOW TO BUILD YOUR NEXT PC

OUT NOW!
WITH FREE DIGITAL EDITION

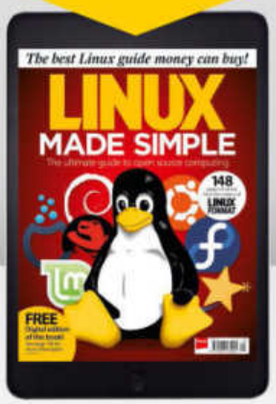
DELIVERED DIRECT TO YOUR DOOR

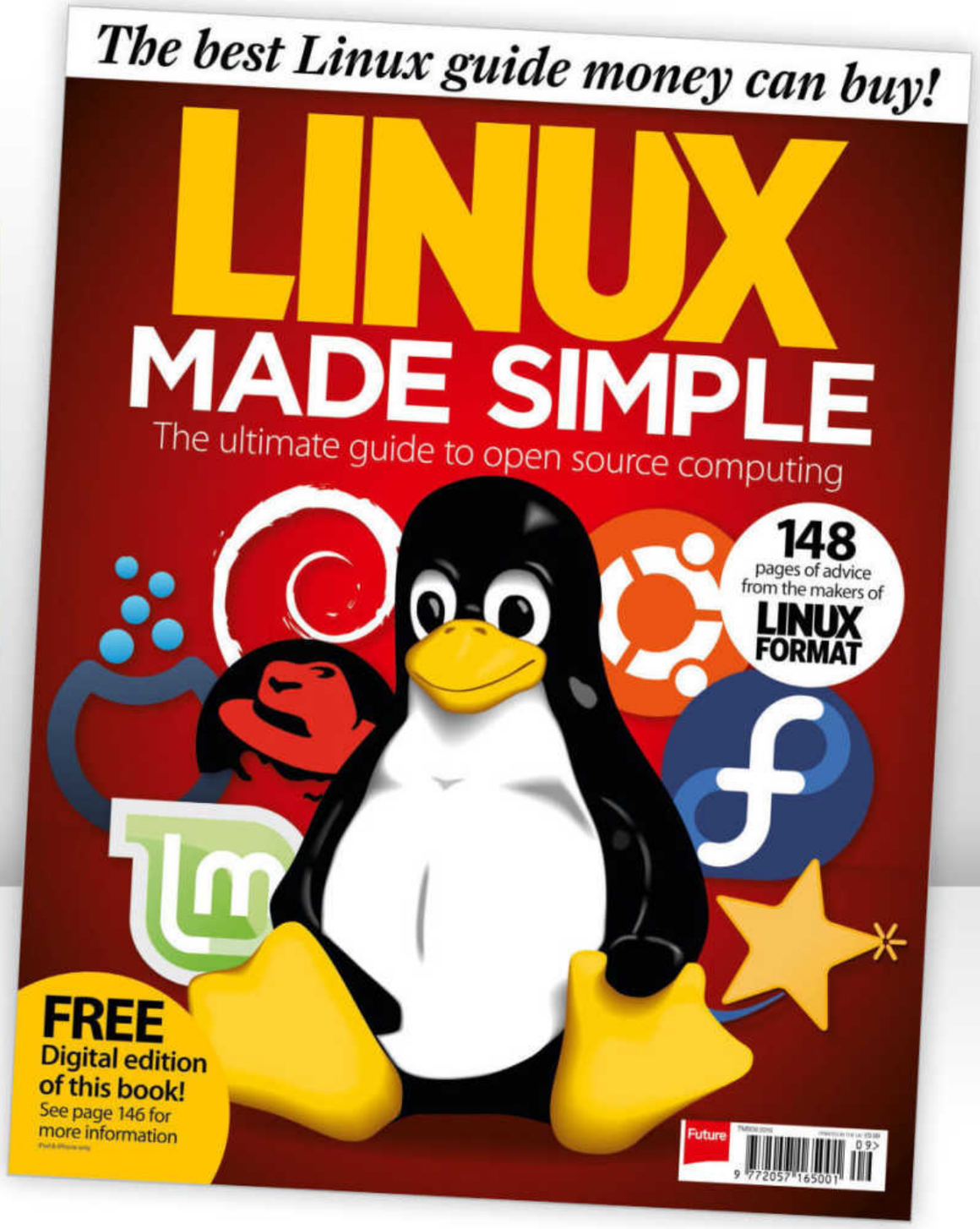
Order online at **https://www.myfavouritemagazines.co.uk/tech-gadgets/the-ultimate-pc-building-handbook**

or find us in your nearest supermarket, newsagent or bookstore!

ALL YOU NEED TO GET INTO LINUX TODAY!

OUT NOW!
WITH FREE DIGITAL EDITION





DELIVERED DIRECT TO YOUR DOOR

Order online at **www.myfavouritemagazines.co.uk**
or find us in your nearest supermarket, newsagent or bookstore!

THE ULTIMATE Raspberry Pi HANDBOOK

The *only* guide you need to get the most from the amazing mini-PC

180 pages of step-by-step guides, ideas, tutorials and more to help you boost your skills and do incredible things with your Pi.

180 pages of Raspberry Pi advice!

Discover how to:

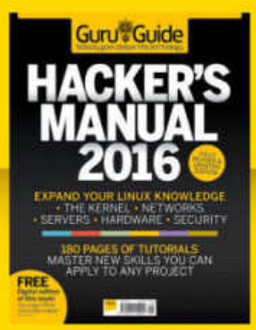
- Upgrade your Raspberry Pi hardware
- Master Linux skills and make the Pi your own
- Go from Python novice to coding master

THE BEST GEAR Expert guides to the Raspberry Pi range and beyond

CONTROL YOUR KIT Hardware hacking made simple with our comprehensive guides

GO BEYOND From first install to your own custom Linux operating system

Like this? Then you'll also love...



Visit **myfavouritemagazines.co.uk** today!