

LINUXVOICE

50% of our profits go back to Free Software • We freely license our content after nine months

**32-PAGE
FREE SAMPLE**
(REGULAR ISSUES
ARE 115 PAGES!)

15 PRIVACY

OPENPGP

Secure your emails from GCHQ and the NSA

21 DESKTOP

LXDE

Save RAM and make your machine faster

24 FILESYSTEM

BTRFS

Discover advanced features coming to your distro



THE BIG SWITCH

3 How the city of Munich switched 15,000 PCs from Windows to Linux – and left Microsoft's boss fuming

FREE SOFTWARE | FREE SPEECH



26 **VIRTUALISATION** KVM, VirtualBox, containers – which is best?

30 **SECURITY** Explore the magic behind key exchange algorithms

28 **SSH MASTERCLASS**

9 **RASPBERRY PI**

MARS ROVER PROJECT

Beat NASA with an ice cream tub



17 **INTERVIEW**

DAMIAN CONWAY

Perl 6 hacker on code and society



7 **REVIEW**

ACER C720 CHROMEBOOK

Not just limited to Google's OS



Welcome to Linux Voice

75,000 of words of awesome each and every month.

LINUX VOICE

Linux Voice is different.
Linux Voice is special.
Here's why...

1 At the end of each financial year we'll give 50% of our profits to a selection of organisations that support free software, decided by a vote among our readers (that's you).

2 No later than nine months after first publication, we will relicence all of our content under the Creative Commons CC-BY-SA licence, so that old content can still be useful, and can live on even after the magazine has come off the shelves.

3 We're a small company, so we don't have a board of directors or a bunch of shareholders in the City of London to keep happy. The only people that matter to us are the readers (you again).

THE LINUX VOICE TEAM

Editor Graham Morrison
graham@linuxvoice.com

Deputy editor Andrew Gregory
andrew@linuxvoice.com

Technical editor Ben Everard
ben@linuxvoice.com

Editor at large Mike Saunders
mike@linuxvoice.com

Games editor Liam Dawe
liam@linuxvoice.com

Creative director Stacey Black
stacey@linuxvoice.com

Malign puppetmaster Nick Veitch
nick@linuxvoice.com

Editorial contributors:
Mark Crutch, Juliet Kemp, John Lane,
Simon Phipps, Les Pounder, Jonathan
Roberts, Mayank Sharma, Valentine
Sinitsyn



GRAHAM MORRISON

A free software advocate and writer since the late 1990s, Graham is a lapsed KDE contributor and author of the Meeq MIDI step sequencer.

We've pooled some of our favourite features, reviews and tutorials from our first few issues to give potential readers and subscribers a better idea of what our magazine contains. We put together 115 pages of content like this each and every month.

Linux Voice is the magazine we launched after a trailblazing \$200,000 Indiegogo campaign that concluded in December 2013. Our success was purely down to the incredible support shown by the community, and wonderful endorsements from the likes of Simon Phipps, Karen Sandler, Eben Upton and Jono Bacon. As old hands in the Linux magazine business, we started Linux Voice because we wanted to create the best magazine we could without any external influences and because we wanted to make a magazine that was more accountable to the community.

By writing the best content, by releasing all our content CC-BY-SA, and by giving a proportion of our profits back to projects selected by our readers, we feel we've succeeded in creating the best magazine we can. So why not join us!

Graham Morrison
Editor, Linux Voice

**SUBSCRIBE
FROM JUST
£38
SEE PAGE 32**

What's been hot in Linux Voice



ANDREW GREGORY

Who couldn't fall in love with Ben's ice cream carton Mars Rover in issue 4? Strawberries and cream robot perfection!



BEN EVERARD

Our series on old school computer scientists like Grace Hopper and Alan Turing makes me want a time machine!



MIKE SAUNDERS

We've been shining some much needed light on groups like the FSFE, Code Club and Young Rewired State.

Free mini sample

www.linuxvoice.com



THE BIG SWITCH

Munich city council has migrated 15,000 workers from Windows to Linux. **Mike Saunders** and **Graham Morrison** visited the city and learned just how upset Steve Ballmer was...

“One of the biggest aims of LiMux was to make the city more independent.”

Hirschgarten, in the west of Munich, is one of Europe's biggest beer gardens, with over 8,000 places to sit. It's a spectacular sight in summer: hundreds of benches as far as the eye can see, trees providing some shelter from the heat, and a vast number of people relaxing and enjoying the city's famous beers.

But while 8,000 is an impressive number, it's not as impressive as 15,000. That's how many people the Munich city council has switched from Windows to Linux over

the last decade. Migrating workers of Germany's third-largest city was no easy task and there were plenty of hurdles along the way, but by and large the project has been a storming success.

We've been following the progress of LiMux (Linux in Munich) for years, and now that the project is effectively complete, we decided to visit the city and talk to the man in charge of it. Read on to discover how it all started, how Microsoft tried to torpedo it, and whether other cities in the world can follow Munich's lead...

Free mini sample

www.linuxvoice.com

Humble beginnings

Cast your mind back to 2001, and the state of Linux at the time. It was well established as a server OS and fairly well known among computing hobbyists, but still a small fish in the desktop pond. Gnome and KDE were still young whippersnappers, while hardware detection needed improvements and top-quality desktop applications were lacking in many areas.

So for an entire city council to even consider moving to a largely unknown platform was a major event. Still, it happened gradually, as Peter Hoffman, the project leader for LiMux, told us in his office:

“Back in 2001, a member of the Munich city council asked: are there any alternatives to using Microsoft software? And based on that question, we put out a tender for a study, which compared five platform options. One was purely Microsoft-based, one was Windows with OpenOffice, one was Linux with OpenOffice, and so forth.”

As the study progressed, two main options emerged as choices for the council: remaining with a purely Microsoft solution, which would involve upgrading existing Windows NT and 2000 systems to XP; and moving to a purely Linux and open source alternative. “If you lay more emphasis on the monetary side, the pure Microsoft alternative would have won, or if you lay the emphasis on the strategic side, the open source alternative was better.”

Doing the maths

That was interesting enough – that staying with Microsoft would have been cheaper. Given the cost of buying licences for Windows and Office, you’d think that sticking with Microsoft would’ve cost far more than switching to Linux. However, the calculations were based on a five-year period, so they mostly covered migration costs (staff, technical support, retraining users etc.) rather than operational costs (buying new hardware, licence fees and so forth). But how did the LiMux team determine that Linux was a better choice strategically?

“With the Linux alternative, we saw that it would be possible to implement the security guidelines we wanted to have. At the time there was a lot of discussion about Windows 2000 and the calling home functionality. If you asked Microsoft at that time, ‘which one of your programs are calling home?’, they said ‘err, yeah, maybe some, or not’. So we didn’t get a clear answer at that time, and we thought there would be a great advantage from a security perspective to using Linux.”

One of the biggest aims of LiMux was to make the city more independent. Germany’s major centre-left political party is the SPD, and its local Munich politicians backed the idea of the city council switching to Linux. They wanted to promote small and medium-sized companies in the area, giving them funding to improve the city’s IT infrastructure, instead of sending the money overseas to a large American corporation. The SPD argued that moving to Linux



Peter Hofmann is the leader of the LiMux project, and explained its ups-and-downs from his office overlooking the Frauenkirche.

would foster the local IT market, as the city would pay local consultants and companies to do the work.

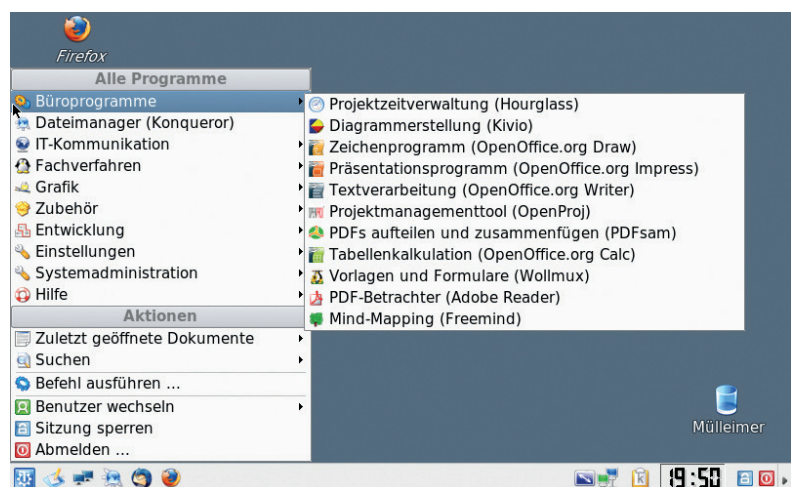
Ballmer marches in

In May 2003, the city council was due to vote on whether to make the big switch to Linux. But Microsoft didn’t stand still: Steve Ballmer, the infamously loud CEO, flew over to speak with Munich’s

What is the LiMux Client?

Put simply, it’s a customised version of Kubuntu. We had a chance to explore it in Peter’s office, and it’s very much what you’d expect from an older Kubuntu release: a Start menu in the bottom-left, various office and productivity applications installed, and a generic theme. There’s a bit of LiMux theming in the wallpaper, but otherwise

it looks rather plain. A new version of the LiMux Client is due this year; it will be based on Kubuntu 12.04, an LTS (Long-Term Support) release. With this, LiMux users across the city will make the transition to KDE 4, and experience something rather more polished than the KDE 3 desktop they’ve been used to.



It’s not pretty or bleeding-edge, but LiMux has done a fine job of replacing old Windows NT and 2000 installations.

Free mini sample

www.linuxvoice.com



This chart shows the migration path in 2012: from 9,000 desktops at the start of the year to 14,000 by the end.

mayor, Christian Ude. But this had an adverse effect, as Peter explains:

"Steve Ballmer tried to convince our mayor that it would be a bad decision to switch to open source, because it's not something an administration can rely on. But some members of the city council said: what are we, if one member of a big company simply comes here, and he thinks he can just switch our opinions?"

And it just got worse for Microsoft's boss. "Our mayor was preparing for a meeting with Steve Ballmer, and because English is not his native language, he asked his interpreter: 'What shall I say if I don't have the right words?' And the interpreter replied:

'Stay calm, think and say: What else can you offer?'

Later on during the meeting, our mayor was quickly at the point where he had nothing to say to Ballmer, except for

'What else can you offer?' several times. Years later, he heard that Ballmer was deeply impressed by how hard he was in negotiations!"

"LiMux has been a success, and has shown how flexible and effective Free Software is."

Alea Iacta Est

So Steve Ballmer flew back to Microsoft HQ, the Munich city council voted, and it voted in favour of Linux. History had been made. GNU/Linux and Free Software users around the world were pleasantly surprised by the decision – especially as it had been made in Munich and Bavaria, one of the more conservative areas of Europe. Something big was going to happen, but it needed time to take root, as Peter explains:

"We could not to start the migration next day, but wanted to do a proof of concept first. In 2004, we started to take preliminary steps for the migration, and one of them was to put out a tender for a Linux-based solution. Ten companies approached us trying to sell their solutions, and a consortium of two small

companies, Gonicus and Softcon, won the tender with a solution based on Debian."

Gonicus provided consultants, and the city council recruited new technicians – eventually there was a team of 13 working on the LiMux project. They started creating a custom version of Debian and by 2006 the roll-out was beginning. But the choice of Debian caused them some minor headaches further down the line:

"In 2008 we saw that Debian was clearly stable, a good thing, but not the best if you want to use new hardware. They are always a few years behind. We also wanted to have a clear timetable for when new versions would be available. In Debian, when it's ready it's ready, so you can't base a release plan on it. Those two things were the basis for switching from Debian to Kubuntu."

From Debian to Kubuntu

Another reason for using Kubuntu was the KDE desktop. It was clear to the LiMux team that some users would fight back against the change – especially if they regarded the current system as good enough, and the new one as something forced on them by politicians. So KDE was chosen as it could provide an interface very similar to that of Windows NT and 2000, as used by the various departments of the city at the time. How did people respond?

"There are different levels of users. Some would say: 'This button was green before, and it isn't green now, so I cannot work like this!' And the others say: 'Just give me something, I have to work, and I'll get used to it'. We had that kind of range of users, but most were the first type."

Peter and his team worked to ease the migration process by organising meetings and roadshows around the city where people could come and see Linux in action. They had Q&A sessions and even a Microsoft-free zone set up with Linux computers to play with. The goal was that users would get a preview of what they'd be using a year or two down the line.

"Some people came to us and said: 'Can I use a mouse? I thought Linux was only command line based'. One person came with a floppy disk and said 'My most important documents are on this. Is it still possible to work with them?' So we showed that it was possible to open them on Linux. We were always trying to give information to the users: what was happening, and why it was happening."

While LiMux was the central project in charge of the operating system, the roll-out and migration was handled by individual departments. There was no specific deadline: departments would choose by themselves when to handle the transition, and the LiMux team would provide the technical know-how to perform the migration.

Not every public sector employee moved to Linux though. Education was one area in which LiMux couldn't get involved, because the decisions about

Who's next?

Surprisingly, the success of LiMux hasn't resulted in a flood of similar projects across Europe, although we all know how slow things move in politics. Peter has been talking to other administrations around Germany – but whether anything will come from them remains to be seen. A similar project, Wienux, aimed to move the city of Vienna over to Linux, but hit stumbling blocks in 2008.

Peter's reasoning for this: Wienux didn't have proper political backing. You need more than just a couple of technically minded councillors to make such a big project a success – you need to know that you have the support of the majority.

It all has to start somewhere, though, so maybe if we all write to our local councillors, point out the success of LiMux and ask them to consider a similar plan, there'll be a lot more FOSS in our towns and cities in 10 years' time...

educational software are made at a national level in Germany. In addition, a few systems with very esoteric requirements are still running Windows, although Peter tried Wine:

"We have a very limited Wine installation, because there's always the need to save the configuration of Wine together with the application. They're deeply dependent. If you change the version of Wine, you have to do something with the application, and vice-versa. We saw that we'd have to use 10 or 15 different configurations of Wine on the same machine in some cases."

Some software vendors won't support their programs if they're running on Wine rather than a native Windows installation, so in the end the LiMux team only deployed two Wine installations.

While the LiMux version of Kubuntu was fairly standardised across the different departments in the city, it took a lot of work to provide the same functionality as the myriad Windows setups previously out there. Peter and his team counted over 50 different configurations of Windows in use, so even when the transition had gone well for one department, the requirements of the next one were often completely different.

Today, the IT infrastructure is a lot more centralised, with the LiMux developers issuing new releases and giving support. It's much easier to fix problems and help people when you have roughly the same operating system on each PC, rather than non-standard custom setups with different service packs, patches and so forth.

Money talks

While the initial aim of the project wasn't to save money, it's still what a lot of people talk about. Today, over a decade down the line, has LiMux been a good idea in terms of finances?

"Yes, it has, depending on the calculation. We did a calculation and we made it publicly available on our information system for the city council. We have the exact same parameters for staying with Windows as



with the migration to the Linux platform. Based on those parameters, Linux has saved us €10m."

A respectable sum indeed – but some companies weren't happy with it. HP compiled a study which concluded that no, actually, switching to Linux had cost the city €60m. Had Munich stayed with Microsoft and moved to Windows XP and Office 2003, it would only have cost €17m. What did Peter and his team make of this?

"We contacted HP and said: 'Nice numbers, how did you calculate them?' And they said 'Uh, um, that was an internal paper and not supposed to be published...' They published a summary, but it was not clear for anyone to see how they calculated."

As a major partner of Microsoft, it's not surprising that HP would try to put a different spin on the project. But the proof is in the pudding: LiMux has been a success, has shown how flexible and effective free software is, and will hopefully inspire many other cities to follow its lead in the future. 🐧

Yes, there are cuddly penguins in the LiMux offices. All is good in the world.

Acer Chromebook C720

There's never enough Google in your life. Says Google. So it's a good job that you (and **Graham Morrison**) can install something else on this great little Chromebook.

DATA

Web

<http://www.acer.co.uk>

Developer

Acer

Price

£199

SUNSPIDER 1.0.2 BENCHMARKS

C720 with Chrome OS:

411.4ms

Firefox/Ubuntu LTS:

315ms

Intel 2.4GHz i5 (4258U):

168.2ms

Not many of us would have thought that a laptop with little more than a web browser would be useful. How can you build things? Or edit anything other than text? Or put together a magazine, or play games?

However, portable computing has become more divergent than we could ever have imagined when we wanted our netbooks to do everything. We're now productive with smartphones and tablets, despite neither being capable of running a virtual machine nor a compiler, and the typical Chromebook is an extension of that idea. It's a low-powered laptop that boots quickly into Google's Chrome OS, a Linux-based operating system that's basically nothing more than the Chrome browser running in fullscreen mode. There's no support for applications other than the web-based ones you can download through Chrome. There's no command line, and there's very little scope for customisation.

What you do get is online and offline access to Google's carefully crafted suite of productivity applications; word processing, spreadsheets and presentation creation alongside YouTube, Maps, Keep and anything else you install through Google's web store. It all works extremely well, and the lack of the usual clutter that comes with an operating system helps to keep you focused – that is, until you install Command & Conquer. There's very little to be said about Chrome OS, which is a good thing for certain



There's a USB 2 port and a USB 3 port, plus an SDCARD reader – but there's only 16GB of usable local storage.

scenarios. There is a small task manager bar at the bottom of the screen, and a launch menu on the right for the Google apps and other apps you may install. Configuration is through Chrome's settings panel, which is augmented for Chrome OS with options to change the background image, network management, and touchpad control – both for sensitivity and for reverse (labelled 'Australian!') scroll settings, which is a mode we're ashamed to admit we now prefer.

Chronos

Acer's C720 is better than the average Chromebook. It has a 1.4GHz Intel Celeron CPU descended from the current Haswell range, which is a massive step up from the ARM CPUs used by many chromebooks. It's a processor that has far more in common with the CPUs found on more general-use laptops and desktops, even if you can't take advantage of its particulars from Chrome OS. It does, however, help Chrome OS feel extremely slick. Wireless internet resumes within seconds of lifting the lid, and it deals with complex websites, WebGL, HD Adobe Flash and HTML playback with ease.

There's very little local storage – around 9GB to play with, but Google obviously wants you to use its own cloud product and Chrome OS plugs directly into your Drive account, as well as offering a deal on 100GB cloud storage for 2 years. Google's video conferencing app, Hangouts, is a good example of whatever hardware acceleration Chrome OS must be leveraging, as we had our best experience on the C720 after trying many devices, from the Nexus 5 and



Acer's Chromebook is surprisingly light, yet despite that it feels like it could withstand the bumps and knocks of travelling to a thousand geek conferences.

Free mini sample

www.linuxvoice.com

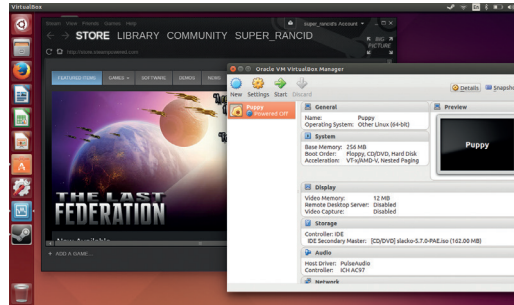
Running Ubuntu

What many Linux users really want to know is how the laptop performs if you install a different flavour of Linux into it. We installed Ubuntu 14.04 with relative ease, but this was mainly thanks to ChrUbuntu, a series of scripts that automatically handle the re-partitioning and downloading of the required packages. All you need to do is go through the slightly convoluted procedure of entering developer mode and making sure that every command you type looks sensible.

The main problem with running Ubuntu rather than Chrome OS is that a default installation requires around 4.5GB of space, leaving you with only a few precious GB for your own data if you keep Chrome OS installed alongside – which we'd recommend. Ubuntu ran brilliantly on the C720, and running a full-fat Linux distribution on the Chromebook felt very liberating. Overall performance, for a 1.4GHz CPU, was fantastic. We even installed Steam to play a few games, and the 3D acceleration in the CPU was more than adequate for many of the less demanding games in our collection. Unfortunately, it's the 2GB of RAM that's the biggest limitation. There's theoretically a 4GB version of the same laptop, but we've not been able to find UK stock, but that would make life a lot easier for other distributions.

Thanks to a VT-x enabled CPU, we even had VirtualBox up and running with a couple of Linux distributions. The biggest

limitation was storage and RAM, but it's still an impressive feat and genuinely useful for running small or server-oriented distributions. Battery performance under Ubuntu was about the same as Chrome OS, but there were a few hiccups that would probably flummox a lot of users. The touchpad stopped working with a kernel upgrade, for example, and we couldn't get Wi-Fi to resume from suspend or the USB 3 port to work without a little script tinkering.



Ubuntu runs brilliantly on the Chromebook – it was possible to run OpenGL Steam games and VirtualBox.


Nexus 10 to quad-core 16GB desktops and laptops. We also had almost two weeks standby time and a typical battery life of 7–8 hours, depending again on whether we launched Command & Conquer.

We were disappointed by the screen, as this hasn't moved with the times. It could be argued that a 1366x768 resolution on a 11.6" screen is acceptable when all you're doing is running a full-screen browser, but we're growing increasingly used to the high DPIs found in phones and tablets. It also lacks punch when compared with IPS displays and has a woeful range of viewing angles. The screen, along with the gun-grey plastic case design helps to make the C720 feel rather more utilitarian than it should. We also wish there were more options for storage. We understand that the tiny SSD is there to keep costs down and perhaps to limit its use outside of Chrome OS, but we'd like the option of adding more. Watching a film or managing anything more than a few albums is going to be impossible without a great data connection. Fortunately, while we've not tried it ourselves, it looks relatively straightforward to replace the SSD with a

larger unit, even if it does invoke the use of a small plastic wedge. We did like the keyboard, however, and the Chrome OS specific keys – backwards, forwards, task switching, and refresh – made us miss their inclusion on many post-Windows 98 keyboards.

You can work and type very effectively from the Acer and we were able to write many words for this very issue from its keyboard. It's also light enough (1.3kg)

and small enough (19mm thick) to throw into almost any bag, and as long as you can offer some kind of networking connectivity, it's the perfect travelling companion for people who need to do some work.

We like the C720 a lot, mainly because of its price, portability and relative power. At just less than £200 in the UK, it could be the perfect laptop to give to family or friends who aren't too confident with computers, where you know the limited options of Chrome OS are going to be all they'll ever need. But it also makes an extremely good low-cost travelling companion, especially if you put another Linux distro onto it. If you can find a compatible SSD, it makes a great machine for hacking about on and for throwing into a rucksack for LUG meetings or hacker sessions. 

"The C720 Chromebook makes an extremely good low-cost travelling companion."



Many games are playable though the Chrome web store, and it's surprising just how powerful the average web browser has become.

LINUX VOICE VERDICT

Excellent value for money and perfect for non-technical relatives, corporate deployment and expert Linux tinkerers.



Free mini sample

www.linuxvoice.com

BEN EVERARD

RASPBERRY PI: BUILD A MARS ROVER

Polish your CV and call NASA: you're about to become a professional-grade robot builder.

WHY DO THIS?

- Get started with robotics
- Learn more about the Raspberry Pi
- Build a robot army and take over the world

"Robotics is a complex area that requires a combination of electronics understanding and the ability to use specialised machinery". That last sentence is a common sentiment, but it's utter balderdash. Modern development boards like the Raspberry Pi (and the host of expansions that do with them) combined with the flexibility of Linux makes robotics incredibly easy.

To prove this, we're going to build a Mars rover-type buggy based on a Raspberry Pi. You'll be able to control it remotely, and it'll stream video back to the controller. To make control really easy, we'll build a smartphone app to use the phone's accelerometer, so you can drive the buggy by turning the phone (much like the controls in many smartphone video games).

There are quite a few parts to this, and we'll be using a few different technologies to control different parts, but thanks to the wide range of development tools on Linux, it's not as difficult as it sounds. For the hardware you'll need:

- Raspberry Pi and SD card (it is possible with a model A, but a model B will be easier to develop on).
- Raspberry Pi camera module (the NoIR module will be able to see in the dark).
- Raspberry Pi-compatible Wi-Fi dongle (see http://elinux.org/RPi_USB_Wi-Fi_Adapters).
- Power supply for Raspberry Pi.
- Power supply for motors.
- Two motors and drive train.
- One or two more wheels.
- Motor controller.
- Chassis.

You'll also need a Linux machine to do some development on, and an Android phone (other smart

phones should work, though you'll need the appropriate development environment).

If you haven't worked with robotics before, the final four might sound a little complex, but don't worry, they needn't be. While you could use almost any motors you can get your hands on, there are some easy, reasonably priced ones that are particularly easy to use from PiBorg (<http://piborg.org/accessories/dc-motor-gearbox-wheel>) and other suppliers. You only need two of these to drive the robot, and the only assembly is pushing the wheel onto the axle.

Reliant Mars Robin

For the final wheel (ours has three, but yours could have four), we used a ball caster (like this one: <http://shop.pimoroni.com/products/pololu-ball-caster-with-3-4-metal-ball>). This allowed the back of the buggy to move freely and follow the front two wheels.

The Raspberry Pi does have General Purpose Input and Output (GPIO) pins that can be used to switch low-power components like LEDs on and off. However, motors draw a much higher current than the GPIO pins can provide. Therefore you need some way of taking a signal from the Pi and converting it into an electrical current powerful enough to drive a motor. For the purposes of this project, we can classify these into two types: on/off controllers and variable speed controllers. The first (such as the PicoBorg or the relays on a PiFace) will work, but the controls won't be as finely-grained as they could be. We used a PicoBorg Reverse (<http://piborg.org/picoborgrev>), which enables us to vary the speed of each motor (other controllers are available with the same features). The most important thing is that the board you use as the brains of the robot should be controllable from Python (almost all are). There should be sample code on the board's website to show you how to do this.

The build

The chassis can be as simple or as complex as you like. Specialised robot chassis are available that are robust and capable of carrying lots of sensors. We don't need this much for a simple buggy though. You can use anything provided you can mount the wheels on it and it will support the electronics.

Finally, we used a USB power pack and a 9V battery to power the Pi and the motors respectively. This is quite a lot of hardware, but all of it could be used on other projects.



An ice cream tub makes a simple and cheap robot chassis – just make sure you wash it out first.

Obviously the build will vary depending on exactly what parts you've chosen. For us, it involved connecting the PicoBorg Reverse according to the instructions on the website (<http://piborg.org/picoborgrev/install>).

To set up the buggy, we glued the motors to either side of one end of the ice cream tub, and bolted the caster to the other end. This created a three-wheeled buggy driven by the two motors at the front. We set the Wi-Fi to automatically connect to our network using the WiFi Config tool on the Raspbian desktop.

All motor controllers should come with some test code so that you can make sure everything is working. The software that installs the PicoBorg Reverse drivers will also put an app on the desktop. If you haven't already, you should run that now. Now is also a good time to make sure that both motors are wired the correct way round. With both motors on forward, the buggy should obviously move forward. With motor 1 on and motor 2 off it should turn left, and with motor 2 on and motor 1 off, it should turn right. If this is different on your buggy, you just need to switch the wires around until it works correctly.

Fire up Python

The PicoBorg Reverse software includes a Python module to control the motors, but it doesn't install it to the global Python directory, so it's not available to scripts that are run from other locations. In order to make this module available, you'll have to copy it across yourself with the following code (you may need to adjust the path depending on where you unzipped the install files):

```
sudo cp /home/pi/picoborgrev/PicoBorgRev.py /usr/lib/cd
pymodules/python2.7/
```

We'll use a simple web server to control the buggy. Web servers work by waiting for requests, and then serving web pages based on the request they get. Normally, the request is given in the URL that the website visitor's browser sends to the web server. For instance, if you visit [www.linuxvoice.com/wp-](http://www.linuxvoice.com/wp-content/uploads/2014/04/turtle.png)

Alternatives to the Pi

The Raspberry Pi is particularly well suited to this project because the camera is well supported and there are plenty of motor control add-ons to provide all the functionality you need. However, it's not the only option. It should be possible to do more or less the same thing on a BeagleBone Black, although you'll have to do a little bit more work to get streaming video set up (there's a guide here: <http://shrkey.com/installing-mjpg-streamer-on-beaglebone-black>). Larger boards such as the Odroid or Udo should work as well, though they'll drain the batteries faster, and their extra processing power isn't really useful for this project.

It should be possible to use a microcontroller such as an Arduino to handle the motor control (though it would be better to use Bluetooth than Wi-Fi in this case). Getting streaming video working with a microcontroller would be challenging, though probably not impossible if you are determined enough. However, you could do this separately using a wireless webcam.



A bit of glue will hold the motors in place, but be careful not to get any on moving parts.

`content/uploads/2014/04/turtle.png` you are requesting the file `/wp-content/uploads/2014/04/turtle.png` from the server www.linuxvoice.com. The server will respond to this request by sending an image from the Python drawing tutorial from Linux Voice issue 2.

Requests don't have to be for files though. The web server can deal with requests however it wants. You can also send bits of data in the URL. These arguments in the URL string come after a question mark and are separated by ampersands. For example, in the URL www.google.co.uk/search?q=linuxvoice, the argument `q` is set to the string "linuxvoice".

We're going to use the Python Tornado web server to use these requests to control the motors on the Pi. You'll need to install this on the Pi with:

```
sudo apt-get install python-tornado
```

The code to control the motors using the PicoBorg Reverse is:

```
import PicoBorgRev
import subprocess
import tornado.ioloop
import tornado.web

maxspeed = 0.3
PBR = PicoBorgRev.PicoBorgRev()
PBR.Init()
PBR.ResetEpo()

class TurnHandler(tornado.web.RequestHandler):
    def get(self):
        PBR.SetMotor1(min([float(self.
get_argument("motor1"))/100, maxspeed]))
        PBR.SetMotor2(min([float(self.
get_argument("motor2"))/100, maxspeed]))
        self.write("Updated")

class HaltHandler(tornado.web.RequestHandler):
    def get(self):
        subprocess.call(["sudo", "halt"])

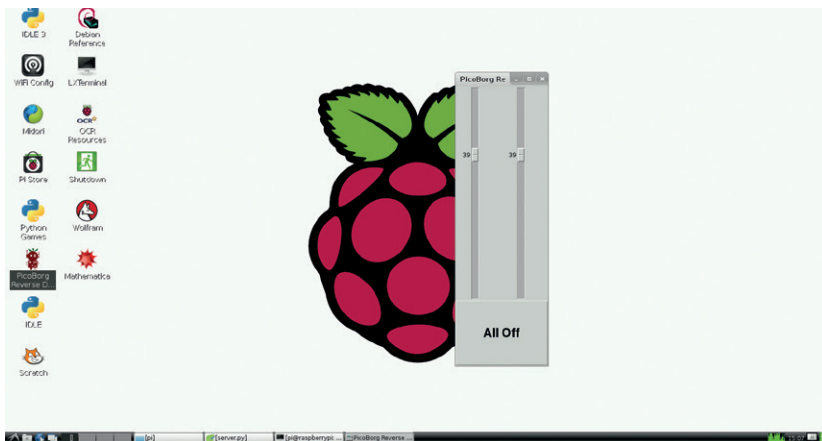
if __name__ == "__main__":
```

LV PRO TIP

Robots are like Lego: once it's built, play with it for while, then take it to bits and build a new one.

Free mini sample

www.linuxvoice.com



The PiBorg Reverse GUI controller is useful for making sure everything's connected correctly.

```
application = tornado.web.Application([
    (r"/turn/", TurnHandler),
    (r"/halt/", HaltHandler)])
application.listen(8000)
tornado.ioloop.IOLoop.instance().start()
```

The final block of this code (which starts with `if __name__`) sets up the web server running on port 8000 (we'll use port 80 – the usual web server port – a bit later). It uses the class **TurnHandler** to handle requests to `/turn/`, and the class **HaltHandler** to deal with calls to `/halt/`. Both of these classes extend **tornado.web.RequestHandler**, which sets them up with almost everything they need. The only thing this code does is add the `get` method that is called whenever a HTTP GET request is sent to the appropriate URL.

You can access the arguments passed in the URL using the `self.get_argument()` method. The two calls in **TurnHandler** are to get the arguments called **motor1** and **motor2**. We then use these values (which

we'll set between -100 and 100) to set the speed of the motor (which is between -1 and 1). We've limited the motor speed using

the global variable **maxspeed** to stop the motors burning out.

The code here works for a PicoBorg Reverse, but it should be fairly trivial to adapt it to other motor boards. If your motor controller only supports on and off, you'll have to include an `if` statement to test the

“You could add an output device to the Pi such as a little LCD screen to display the IP address.”

arguments against a threshold, and if it is, turn the motor on. For example:

```
if float(self.get_argument("motor1")) > 30.0:
```

```
#code to turn motor one on
```

```
else:
```

```
#code to turn motor one off
```

HaltHandler is used to turn the Pi off, since there's no other way to shut it down cleanly when there's not a screen unless you SSH in, which is a little excessive for a simple robot.

We called the file **server.py**, and you can start it running from the LXTerminal command line with:

```
python server.py
```

We'll get it running automatically a bit later on.

You can now control the robot from the Raspberry Pi by opening the web browser and going to **http://localhost:8000/turn/?motor1=20&motor2=20** (be careful not to accidentally drive your robot off your desk when testing this). You can then stop the motors by going to **http://localhost:8000/turn/?motor1=0&motor2=0**.

Control from other machines

You can also access this from other computers on the same network by using the IP address of the Pi. To find out the IP address of the Pi, open LXTerminal and type **ifconfig**. This will output a block of information for each of the network interfaces. The one you need is labelled **wlan0**, and you're looking for the **inet addr**. In the following, the IP address is 192.168.0.33:

```
wlan0    Link encap:Ethernet HWaddr bc:ee:7b:87:7b:38
          inet addr:192.168.0.33 Bcast:192.168.0.255
          Mask:255.255.255.0
          inet6 addr: fe80::beee:7bff:fe87:7b38/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST MTU:1500
          Metric:1
          RX packets:88425 errors:0 dropped:0 overruns:0
          frame:0
          TX packets:81516 errors:0 dropped:0 overruns:0
          carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:76786575 (76.7 MB) TX bytes:14405224
          (14.4 MB)
```

Unfortunately, this isn't fixed and may change from time to time if you reboot the Pi, and it won't be easy to run **ifconfig** if the Pi is mounted inside a robot. There are a few ways around this. Many Wi-Fi routers enable you to assign a static IP address to a device,

Web sockets

The method we've used for controlling the motors is, well, a little hacky. It works, but it doesn't work well. The main problem is that there's a large overhead each time you change the motor speed. The phone app has to negotiate a new TCP connection and send the data, then the Tornado server re-initialises the module to send data to the server. This means there's a noticeable lag between turning the controls and the buggy responding. Part of this is also due to the interval that the app checks the accelerometer, but this has been adjusted to work well with the speed of the server.

A better method would be to create a communications channel through which you can continuously send data. There are a couple of options for this: TCP sockets or Web sockets. Both are supported by Python, and both have plugins for the Cordova framework that we're using for the Android app. Neither should be excessively complex to set up, though they will require some knowledge of both Python and JavaScript. Using one of these methods, you should be able to reduce the latency of the control and increase the frequency with that the app updates the accelerometer readings.

which will enable you to set it so the same IP address will always be assigned to the Pi. You could add some output device such as a little LCD screen to the Pi to display the IP address. The simplest method is to use another Linux computer to scan the address range and find the IP address for the Pi. You can do this using Nmap.

First you'll need to install Nmap from your distro's repositories (on Debian-based systems, this is done with **sudo apt-get install nmap**). Since the above server runs on port 8000, we can use this to detect the Pi. The following command will check all computers in the IP range 192.168.0.0 to 192.168.0.20 to see if that port is open.

```
nmap -sT 192.168.0.0-20 -p 8000
```

The Pi will respond with something like this:

```
Nmap scan report for 192.168.0.33
```

```
Host is up (0.039s latency).
```

```
PORT      STATE SERVICE
```

```
8000/tcp  open  http-alt
```

Usually, the Pi will be the only IP address that returns a state of **OPEN** for this port.

Currently, you also need to start `server.py` manually. We'll set it to start automatically at the end once everything else is set up.

Getting visuals

Installing the Raspberry Pi camera module is simply a case of slotting it into the correct port (the one between the Ethernet and HDMI ports) with the silver coloured bare metal facing towards the HDMI port, then enabling it. Enter **sudo raspi-config** in LXTerminal, then select Enable Camera, then Yes. You'll need to reboot the Pi for the changes to take effect. There's a video guide at www.raspberrypi.org/help/camera-module-setup if you have any problems.

If you don't have a camera mount to attach to the chassis, a blob of Blu-tack also works.

That's the hardware completely set up. There's still a little bit of software to set up on the Pi, but it doesn't involve any more coding. As the saying goes, "good programmers borrow, great programmers steal", and that's exactly what we're going to do. Streaming video from a Raspberry Pi to a website isn't new, and there's no reason to do it yourself.

The easiest setup we've found is at https://github.com/silvanmelchior/RPi_Cam_Web_Interface. Just download the ZIP file and install it with:

```
unzip RPi_Cam_Web_Interface-master.zip
```

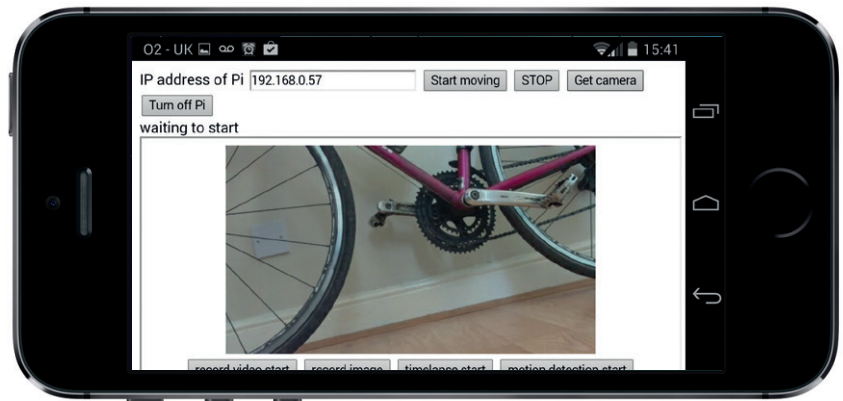
```
cd RPi_Cam_Web_Interface-master
```

```
chmod a+x RPi_Cam_Web_Interface_Installer.sh
```

```
./RPi_Cam_Web_Interface_Installer.sh install
```

Reboot the Pi so it picks up all the new settings. It'll automatically create a web server (on port 80) that starts when you turn on the Pi, and hosts a website with the streaming video as well as some settings so you can control the video stream (and record pictures and video from your buggy).

Once it's up and running, you should be able to open **http://<ip-address-of-pi>** in a web browser on



another computer connected to the same network and see the video stream. You don't need to modify it at all, but it'll fit into the smart phone app we'll create in the next step a bit better if you get rid of the title and resize the image.

To do this, open up the **/var/www/index.html** file on the Pi using a text editor running as `sudo`. For example, to do this in Leafpad, run

```
sudo leafpad /var/www/index.html
```

To get rid of the title, delete the line:

```
<h1>RPi Cam Control</h1>
```

The size you want the image to be will depend on the resolution of your phone screen. We went with a width of 400 pixels, though you can adjust this at the end to make it fit properly on your phone. To do this, change the line:

```
<div><img id="mjpeg_dest"></div>
```

to:

```
<div><img id="mjpeg_dest" width=400px height=auto></div>
```

The only thing left to do set the Python script that runs the motor control server to start automatically (we didn't do this earlier because the setup for the webcam overwrites the file it's done in). Just add the following line (you may have to modify it depending on where you saved **server.py**):

```
python /home/pi/picoborgrev/server.py
```

to the file **/etc/rc.local** directly before the final line (exit 0). Again, you'll need to use a text editor running as superuser, so open Leafpad with `sudo` as you did with **index.html**. That's all the setup for the Pi – now to create the phone app that will control it.

Hands on

The easiest way to create smartphone apps is with Apache Cordova (as seen in Linux Voice issue 2). The idea is that it enables you to use web technologies (mainly HTML and JavaScript) to create apps that can access phone functions that regular web pages cannot. In this case, we'll access the accelerometer.

Accelerometers measure what's known as proper acceleration. This is a little different from what most people know of as acceleration, because it's the acceleration experienced by an object. This means that an accelerometer resting on a surface will experience an acceleration of 9.8 m/s because it's experiencing that acceleration from gravity. On the

The finished app controlling the buggy. It's not much to look at, but the controls are intuitive and fun.

other hand, if you drop the accelerometer, it will read 0 because it's in free fall and not experiencing any acceleration. (Actually, it will read a little higher than 0 because of air resistance.)

As long as you hold the device still, the accelerometer measures gravity. It measures it in three dimensions (x, y and z), which means that you can use it to measure the orientation of the device in three dimensions. In other words, it tells you which way up the device is.

"Cordova's Accelerometer plugin should work on just about every smartphone."

Accelerometer plugin should work on every phone that supports Cordova, which is just about every smartphone (Amazon Fire OS, Android, Blackberry 10, FirefoxOS, iOS, Ubuntu Touch, Windows phone 7 & 8, Windows 8 and Tizen). We'll look at Android here, and there are details of how to get started in the different environments on the Cordova website (http://cordova.apache.org/docs/en/3.4.0/guide_platforms_index.md.html#Platform%20Guides).

Cordova runs on node.js, so you'll need to install **npm** (the node package manager) from your distro's repositories. People using Ubuntu-based systems will need to add a PPA to get the most up-to-date version of node for this.

```
sudo add-apt-repository -y ppa:chris-lea/node.js
sudo apt-get update
sudo apt-get install npm openjdk-6-jdk
sudo npm install -g cordova
```

As well as Cordova, you'll also need the Android Software Development Kit (SDK) from Google (download this from <http://developer.android.com/sdk/index.html>). Once you've downloaded and installed this, you'll need to set up some environmental variables so that Cordova knows where to find

```
export PATH=${PATH}:/home/ben/adt-bundle-
linux-x86-20140321/sdk/platform-tools:/home/ben/adt-bundle-
linux-x86-20140321/sdk/tools
```

First, though, you'll need to set up a Cordova environment on your development machine. According to the Cordova documentation, the

```
export PATH=${PATH}:/home/ben/adt-bundle-
linux-x86-20140321/sdk/platform-tools:/home/ben/adt-bundle-
linux-x86-20140321/sdk/tools
```

You'll need to amend the paths to point to the Android SDK you downloaded and extracted. You can run these commands in the terminal, but it won't remember the settings, so you'll have to re-enter them each time you reboot. In order to add these permanently, add the two lines to the **.bashrc** file in your home directory.

To create a Cordova project for the buggy run:

```
cordova create buggy
cd buggy
cordova platform add android
cordova plugin add org.apache.cordova.device-motion
```

We based our code on the **watchAcceleration** Full Example from http://cordova.apache.org/docs/en/3.3.0/cordova_accelerometer_accelerometer.md.html#Accelerometer. This provides everything to read the acceleration periodically, and the function **onSuccess()** is called when it's successfully read.

Before getting into what we do with the acceleration, let's look at how we'll lay out the screen. This is the code between **<body>** and **</body>**:

```
IP address of Pi <input type="text" name="ip" id="ippi">
<button onclick="startMoving()">Start moving</button>
<button onclick="stopMoving()">STOP</button>
<button onclick="getCamera()">Get camera</button>
<div id="sendingstring">waiting to start</div>
<iframe id="cam" width=100% height=600px></iframe>
<iframe id="turniframe" width=1px height=1px></iframe>
```

As you can see, there will be a text field to enter the IP address of the Raspberry Pi, and three buttons to start controlling the motors, stop controlling the motors, and start the camera feed. **<div id="sendingString"></div>** will hold the URL that's being sent to control the motors. This isn't necessary, but it's useful to see what's going on.

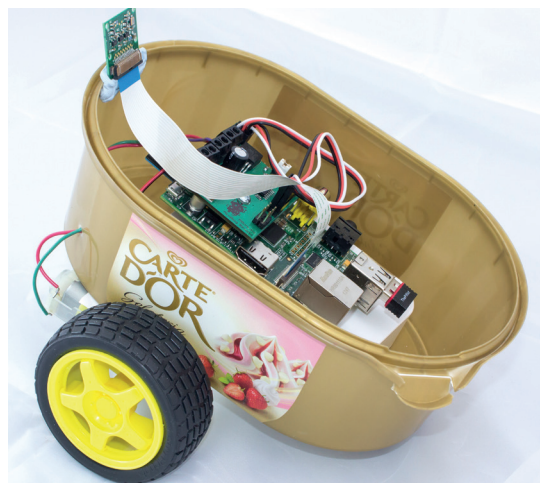
Embed video

Iframes enable you to embed web pages inside of web pages. The first one (with the id **'cam'**) holds the streaming video from the Raspberry Pi camera. The second one (with the id **'turniframe'**) doesn't actually hold anything useful, but by changing its URL, we can use it to create GET requests that control the motors.

To make this work, you need three new JavaScript functions that will run when the buttons are pressed:

```
function getCamera() {
    document.getElementById('cam').src = "http://" + document.
getElementById('ippi').value;
}
function startMoving() {
    window.piMoving=true;
}
function stopMoving() {
    window.piMoving=false;
}
```

The first of these just sets the URL of the **cam** iframe to the address of the streaming webcam



That's all it takes to build a simple robot: Linux on the Raspberry Pi to power the motors, and Linux on a smart phone to handle the controls.

running on the Pi. Remember that we've removed the title and resized the image to make it fit in here. The rest of the controls are still there, so you can tune the streaming image by scrolling down the iframe.

startMoving() and stopMoving()

set the variable **window.piMoving** to **true** or **false**.

This is just a global variable that we'll use to control whether the motor settings are sent to the Pi or not.

You also need to update the **onSuccess()** function (which runs every time it reads the acceleration) to:

```
function onSuccess(acceleration) {
    var element2 = document.getElementById('sendingstring');

    if (window.piMoving) {
        var motor1Prop = (acceleration.y + 10)/20;
        var motor2Prop = 1 - motor1Prop;
        var totalSpeed = acceleration.z * 10;
        var motor1Speed = motor1Prop * totalSpeed;
        var motor2Speed = motor2Prop * totalSpeed;
        sendString = "http://" + document.getElementById('ippi').value + ":8000/turn/?motor1=" + motor1Speed + "&motor2=" + motor2Speed;
        element2.innerHTML = sendString;
        document.getElementById('turniframe').src = sendString;
    }
}
```

Although it's not completely necessary, you can increase the frequency with which the app updates the buggy's speed by altering the frequency setting in the **startWatch** function. In the following example, it updates it once a second, but you could set this to be higher or lower.

```
function startWatch() {
    var options = { frequency: 1000 };
    watchID = navigator.accelerometer.
watchAcceleration(onSuccess, onError, options);
}
```

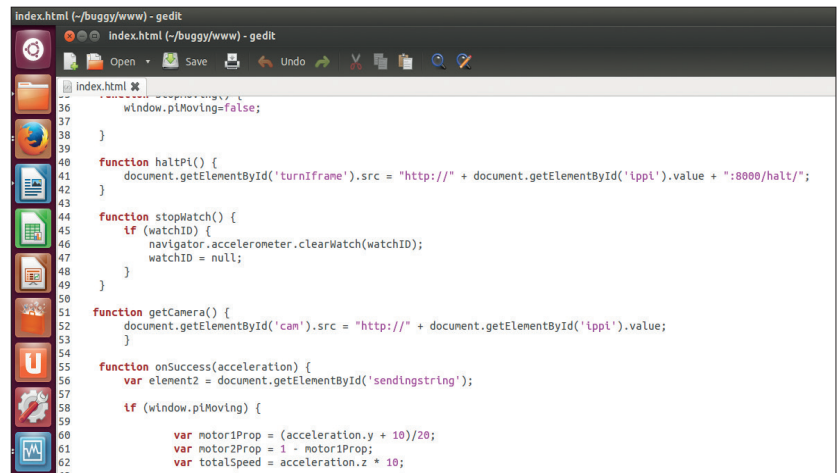
The full code is on the Linux Voice website.

This calculates the speed for the two motors.

acceleration.y is used to change the direction and **acceleration.z** is used to change the speed. This works for holding the phone in landscape. With the screen at right angles to the ground, the buggy will stop, and as you tilt the screen forward (so the screen starts to face upwards), it will start to move. If you tilt the screen back, the buggy will move backwards. Tilting the screen from side to side (as though it were a car steering wheel) will turn the buggy.

Security

This robot is controlled via Wi-Fi with absolutely no security whatsoever. Anyone else on the network could quite easily take over control. Normally this isn't a problem on a local area network, but there may be occasions where you want a bit more privacy. Tornado does handle security quite well, though it's beyond the scope of this tutorial to go into it in detail. Take a look at the documentation on the project's website for guidance on this (www.tornadoweb.org/en/stable). Securing the video stream may be a little trickier, as it's not really designed for it.



The acceleration in each axis is returned as a number between -10 and 10. The formula (acceleration.y+10)/20 returns a number between 0 and 1 depending on how far the phone is rotated. This is then used as a multiplier for the speed of one motor. The multiplier for the speed of the other motor is this value taken away from 1.


The overall speed is the acceleration in the z axis multiplied by 10. This gives it the range -100 to +100 (with negative values being backwards). This is the same range that the motors have. To get the final speed for each motor, we just multiply that motor's proportion by the total speed. This is quite a simplistic method of calculating the speed, and the turn directions will go back to front if the phone's held the wrong way up. However, it works, and it's easy to understand, so it's good enough for our buggy.

With the code ready, you just need to get it on to a phone in order to run it. Unfortunately, this can require a little fiddling with the Udev rules. There's full information on the Android developer site here: <http://developer.android.com/tools/device.html>. You can skip step 1 because Cordova will handle it for you.

Once this is set up, and the phone is plugged into your computer, you can compile and transfer it to the phone. Enter the following in a terminal in the root directory of the app:

```
cordova build android
```

```
cordova run android
```

As you can see, this isn't a particularly elegant solution. Running two web servers is a little over the top. It could have been re-written to do everything in one either by serving the video up from Python or by controlling the motors from PHP. The phone app could be more integrated rather than just serving up an iframe of the webcam controller. However, this project isn't about technical perfection, it's about demonstrating how you can quickly and easily link things together to easily create complex robots by using the tools that are available on Linux. 

We've used Cordova to create a phone app, but you could easily modify the code to create a web interface using sliders or buttons to control the buggy.

Ben Everard is the co-author of the best-selling book on learning Python with the Raspberry Pi, *Learning Python with Raspberry Pi*. He wrestles lions for fun.

Free mini sample

www.linuxvoice.com

KEEP MESSAGES SECURE WITH PGP

The Feds (and GCHQ, and the NSA) are snooping on our communications, but we can fight back with encryption

Normal email is one of the least secure forms of communication available – less secure even than post cards. These mails can typically be read by anyone on the same network as you, anyone at the ISP, anyone at your mail provider, anyone at the recipient's ISP and anyone on the same network as the recipient, as well as anyone with access to the various networks between the two ISPs.

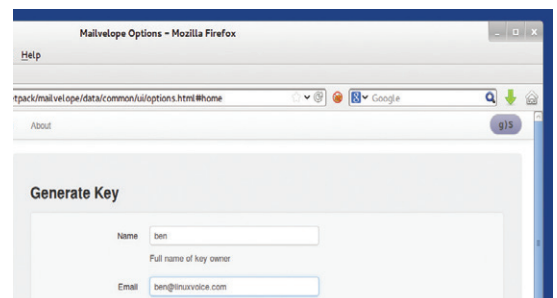
If you use SSL or TLS to connect to your inbox, then it improves things a little, but it's still vulnerable as soon as it leaves your mail provider.

PGP (Pretty Good Privacy) is a program designed to remove these weaknesses. It uses the normal email system, but adds a layer of encryption to protect them in transit. These days, PGP is usually used to refer to the OpenPGP format for these encrypted messages, rather than the PGP program specifically.

The OpenPGP format uses two different types of encryption: symmetric key and public key. In symmetric key encryption the same key (basically just a binary string that's used as a password) is used to encrypt and decrypt the message. In public key encryption, two different keys are used (one to encrypt and one to decrypt). The phrase 'private key' can refer to either the key in symmetric encryption, or the secret key in public key encryption. To avoid this ambiguity, we won't use the phrase in this article, but you may come across it in software.

When encrypting a message with an OpenPGP-compatible program, the software generates a random symmetric key and encrypts the text. This ciphertext forms the bulk of the message.

The problem is that the recipient of the message has to know the key, but it can't be included in the message otherwise anyone who intercepts the message will be able to read it. This is where public



The colour and message in the top-right corner are a random security code so you can distinguish real Mailvelope messages from spoofs.

key encryption comes into play. Everyone who uses PGP first creates a public/secret key pair. The public key is made public while the secret key is known only to the user. However, anything encrypted with the public key can be decrypted only with the secret key and visa versa.

Public and private keys

The solution is to encrypt the key for the message with the recipient's public key. When they receive the message, they can then decrypt the key for the message, and then decrypt the message itself. This is a bit convoluted, but it's much less processor-intensive than encrypting the whole message using public key encryption.

You can use OpenPGP in most mail clients, but we'll look at doing it in webmail. Since OpenPGP is purely a text format, you could generate the encrypted message elsewhere and copy and paste it into your email. That's exactly what we'll do, but instead of copy and paste, we'll use a browser extension to convert the plaintext to encrypted ciphertext.

Mailvelope (www.mailvelope.com) works with Chrome/Chromium and Firefox, and it comes pre-configured to work with some of the most popular webmail providers (Gmail, Yahoo, Outlook.com and GMX). Installing it is no more challenging than downloading the extension from its Releases section (<https://github.com/toberndo/mailvelope/releases>) and opening the file with the appropriate web browser.

The first step is to generate a public/secret key pair. In Chrome/Chromium, you can get to this by clicking on the padlock icon that should have appeared to the right of the address bar. In Firefox, this options menu is a little more hidden. First, you'll need to go to view

USING OTHER MAIL CLIENTS

We've described the process for working with Mailvelope, but the process is almost identical for all OpenPGP-compliant software. You shouldn't have any problems following along using Thunderbird or Evolution, or even AGP and K9 for Android or Cyanogenmod.

Regardless of the software, you'll still have to go through the same process of generating and exchanging keys before you can communicate with someone. As

mentioned in the main text, you should be able to transfer keys between these pieces of software so you can access the same mail account through different programs.

Mailpile is a mail client designed to bring PGP to the masses by making it easier to set up OpenPGP encryption, even for new users. The project raised just over \$163,000 in crowdfunding and is currently in development, and you can track its progress at www.mailpile.is.

DIGITAL SIGNING

OpenPGP encryption ensures that only the intended recipient can read the message; however, it doesn't guarantee that they receive the message, or prove who sent the message. Encryption can't help with the first of these, but there is something you can do about the latter measure.

In many OpenPGP mail clients (and the **gpg** command line tool), you can add a digital signature to a clear-text message. It does this by leaving the message in plain text, but also encrypting a hash of the message with your secret key. This encrypted hash is known as a digital signature. Since it's encrypted with your secret key, it can be decrypted with your public key. Any recipient that knows your public key can then decrypt this hash and check it against the message. If they match, the recipient knows that it really came from you.

> Toolbars > Add-on bar. This will make the Add-on bar appear at the bottom of the screen, and then you should find the padlock icon on the right-hand side of this. This icon will bring up a menu, and you'll need to select Options (see the image, left).

In the Options screen, you can create a new public/secret (private) key pair by selecting Generate Keys. Once you've done this, you can go to the Display Keys screen to see it. This screen will show all the keys that Mailvelope knows, whether they're other people's public keys or your own public/secret key pairs.

Before you can receive emails, you have to send your key to the people you want to communicate with. The key file can be exported from the Display Keys screen (you can also export your public/private key pair here and import them into another mail program).

Getting the public key to the recipient can be a challenge. The best way to do this is to physically transport the key, as you can be completely sure that they got it correctly. The easiest way is just to email them the keyfile. However, it's possible for some malicious attacker to intercept this message and change the keyfile.

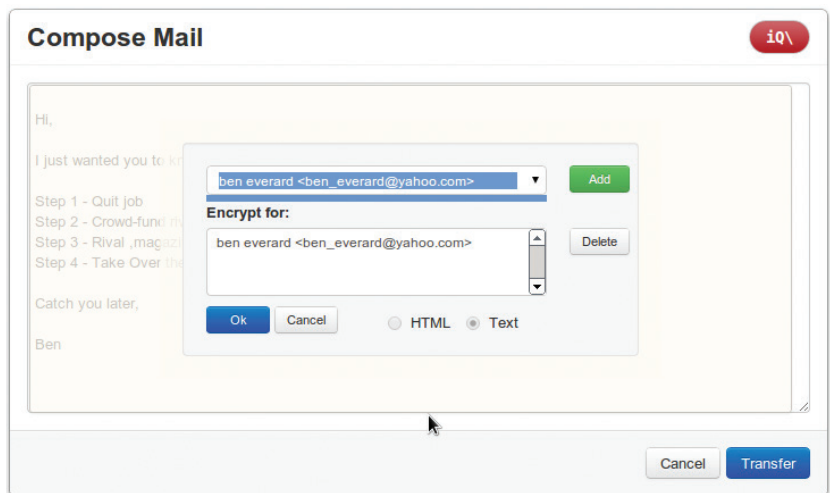
There are two other options: key servers and webs of trust. Key servers are databases of keys that you can add your keys to, and retrieve other people's keys from. For example, try <http://keyserver.pgp.com>

```

1 -----BEGIN PGP SIGNED MESSAGE-----
2 Hash: SHA1
3
4 Hi,
5
6 The eagle flies at midnight.
7 I repeat, the eagle flies at midnight
8
9 Ben
10 -----BEGIN PGP SIGNATURE-----
11 Version: GnuPG v1.4.11 (GNU/Linux)
12
13 iQEcBAEBAgAGBQJSlrx9AAoJEJiddPPe4Nug5XQH/RqP0LBJsbhXhTHhv0v3XxrK
14 tKLaKg7zed/j8Db22vEeW9D4WbgbHDc0w4/C9CbsA2K4hGT2L8UbrxCII1MgIG4
15 hfojFSP17kfqrSPtPv8pRmWd8d/BjGmDK9+uAYWM3fG9b50au96jYaSn+BH4aF
16 6Q3WA+IjTCbR9IPF8fk4MK1unfAAuLYXZDpG7/c4L/1mEWq7hqY4sv2MLfCd4mld
17 Xp0tYEMZM4R0RN8LC2ls6QYz3HHfYogShgJq1lmz9u7D0rknZ+70oKhLILemb09U
18 2BY6eRSWgfwZrDfE10CCVU1VXhG6bWT1bjrHVRLtNQG1uD+7uaV5gXGcUdP2No=
19 =Im6o
20 -----END PGP SIGNATURE-----

```

You can use **gpg** to create signed documents from the command line. Just run **gpg --clear-sign <text-file>** to generate a file containing the plain text and a signature.



or <http://pgp.mit.edu>. Of course, it is possible that some attacker could take control of one or more of these key servers and put fake keys in them. Webs of trust have a decentralised method of verifying keys. It's done by people digitally signing the keys of people they've met and exchanged keys with. If you need to communicate with someone, you can then tap into this web of trust and see who trusts them. Perhaps someone you trust also trusts them. Perhaps someone you trust trusts someone who trusts them. If this chain is short enough, then you can be confident that you can trust the person. Unfortunately, Mailvelope doesn't currently support webs of trust.

Keep it secret, keep it safe

As is so often the case, the decision on which way to distribute your key comes down to security versus convenience. If you're concerned, you could always follow up with another method such as a phone call to confirm the key. Once someone has sent you their key, you just need to load it into Mailvelope using the Import Keys screen in the Options.

Getting set up with keys is the hardest (or at least, most inconvenient) part of using any OpenPGP-based communication. Once you've done this, it's easy. With the Mailvelope extension running, just use your mail provider's web page as normal (if your mail provider isn't already on the Mailvelope watch list, you'll need to add it in the Options). When you get to the compose page, you'll see a floating icon of a pen and paper. Click on this and it will open a new window to let you enter the text for the message. Once you've written the message, click on the padlock, and add one or more people to the list that it's encrypted for, then Transfer to put the ciphertext into the email.

If you receive an encrypted message, Mailvelope will display a decrypt icon; click on this to enter the passphrase you entered when you generated the key. This password gives you some security even if an attacker gets access to your machine.

Provided you exchange keys securely, and keep your keys safe, OpenPGP provides security that is thought to be unbreakable with current technology. 🔒

You can send encrypted messages to several people at once, and Mailvelope encrypts it for each of them.

DAMIAN CONWAY

We meet the creator of a programming language based on Klingon and one of the architects of Perl 6. If only we could tell them apart...

Damian Conway is one of the Guardians of Perl (our term) and one of Perl 6's chief architects. But he's chiefly a computer scientist, a brilliant communicator and an educator. His presentations are often worth crossing continents for. He was the Adjunct Associate Professor in the Faculty of Information Technology at Melbourne's Monash University between 2001 and 2010,

and has run courses on everything from Regular Expressions for Bioinformatics to Presentation Aikido (and of course, lots of Perl). Which is why, when we discovered he was making a keynote at this year's QCon conference in London in March, we braved train delays and the sardine travelling classes of the London Underground to meet him opposite Westminster Abbey.

LV The main reason we wanted to talk to you is that we want to try to simplify people's experience of programming and computers. John Horton Conway said recently that his Game of Life is the blight of his life because he had gone on to do so much more interesting and important work. But what struck us by what he said about the attraction to the game is its simplicity and the fact that that goes on to teach things that you could not possibly imagine. So with that in mind, is there something like that for programming, how does that fit with Perl, and is Perl for people that think like that in the first place?

Damian: That is a huge question! There is almost an industry in making programming seem more difficult than it is. Programming doesn't have to be really complicated. The problems we solve are complicated, and at the scale we have to code things become complicated, but the basic tools of programming are not complicated things. And learning the patterns of use of those tools that work, that scale, that are robust, reliable and maintainable, isn't really that difficult. This is really not rocket science. This is not quantum mechanics. This is not that difficult.

LV But it can be. There was pride in the Perl community when you showed the Turing machine running in this much [gesticulation to show a tiny thing] code.

Damian: Sure, but that's a game. To me, that's just that I make this happen in that kind of way. It's been very interesting for me. I've recently been starting to put together classes on Perl 6, the new language in the Perl family. And the thing about Perl 6 is that it just feels like it's a lot more polished and smooth than Perl 5 ever was.

I mean, I love Perl 5 dearly, I do almost all my work in Perl 5, but Perl 6 has all of the same features but with the rough edges kind of knocked off of them. And what it gives you is the same thing that Perl 5 has always given, which is exactly the right tools to do the job you want to do and not get in your way. What I find when I change to programming in JavaScript or C++ or C is that the language itself gets in the way of my using the language.

I spend all my time coding around either limitations in the language or a particular mindset that makes you do it in one particular way, and that's equally true in Perl 5 on occasion. Perl 5 has got real deficiencies that are only just, in this very year, finally being addressed.

"Programming doesn't have to be complicated. The problems we solve are complicated, but the basic tools of programming are not complicated things."

It's insane, for example, that in Perl 5, until the release that's probably coming out in May, we haven't had parameter lists. Now this is an advanced technology that was pioneered, what, 60 years ago, and we still haven't got them. And so everyone who's writing subroutines in Perl spends most of their time simulating the behaviour necessary for a parameter list. So finally, with Perl 5.20 coming out this year, we have parameter lists.

Every language that I code in, I find these issues. A really good one is, this afternoon I'm talking about regular expressions, and I went through 20 different languages that supply regular expression mechanisms. And in about 18 of them, the regular expression mechanism is bolted on the side, so you can't write a regular expression, you have to write a string, which then gets translated into a regular expression.



And that irritation leads to mistakes too. You don't put the right number of backslashes in, 'cause it's a string, and you've got to backslash all the backslashes to get a single backslash.

LV But, to many of us, Perl looks like a regular expression.

Damian: [laughs] Yeah, but this is kind of the same thing. If I had just gotten up on stage this morning and just shown you Klingon sentences without explaining the structure of them, the syntax of them and how they come together, then it would just look like line noise. Alphabetic line noise, but line

noise. And the thing about Perl is, in the very early design of Perl, a decision was made that there would be lots of syntactic differentiation. In most programming languages, there's only a relatively small amount of syntax. There are identifiers, there are a couple of operators and there's probably a method call mechanism, and then we do everything with that.

In Lisp it's even more extreme. In Lisp there's just comments and atoms, basically. But in Perl the decision was made very early on that we would use as much of the keyboard as possible, so that once you knew what a particular element in the Perl syntax meant, it would stand out for you immediately. So when I read Lisp, and I can read Lisp and write Lisp, and I've taught Lisp, but there's always this mental gear shift that has to go on because the language isn't helping me see what the different

components are. And I find that equally true in Python, which is a lovely language and has many many benefits. But to me, in Python, everything looks like a method call, because everything is a method call. Losing that syntactic distinction makes it really really hard for me to pick up on what's going on.

Now, the problem with that is that it only works if you know the distinction in the syntax. So people coming into Perl get lost in this sea of ampersands and stars and all sorts of other symbols that we use in the language. And until you get past and it sort of goes into your hind brain and it just translates immediately, 'ah yes, that's a scalar variable', 'ah yes, that's a type blah, blah, blah', it doesn't make sense. It looks like line noise, and I fully agree.

LV So do you think it's better for people who want to learn

“In most programming languages there's a relatively small amount of syntax.”

Free mini sample

www.linuxvoice.com



programming to dive into Perl straight away?

Damian: I don't think it is. To be perfectly honest, I think Perl 5 at least is a lousy first language. And the reason I think that is that learning to program isn't just about learning syntax. It's about learning at six or seven different levels at the same time. So the purely lexical level of what character do I type here, the syntactic level of what that means, the semantic level of what does the construct that this represents mean, the algorithmic level of how do I put these things together to make things work... for me it's like when I was learning to juggle or to drive a car or any other complicated multi-level activity. If you think about learning to drive a car, it's not just about how do I steer or how do I push the accelerator pedal, it's also about how aware I am on the road, how I'm aware of what the car is doing, how do I anticipate what's happening next, how do I navigate at the same time and how do I listen to the radio as well. And for me, coding is exactly like that.

For nearly a decade, I taught the introductory programming class at our university, and I was forced to teach it in C and C++ and Java and whatever it

was. But the key is always the same. You have to give them a way of focusing on one level of abstraction at a time. And so the more syntax that the language that they're using has, the harder it is for them to focus on the level of what does this mean, what does it do and how do I make it do what I want. I think from that point of view there have been many CS programs over time that have taught Lisp as their first language. I think, in one sense, that's a really good thing, because I can tell you the syntax of Lisp in three minutes, and from then on it's just trying to understand how the mechanisms work and how the algorithms work.

So I don't think Perl 5 is a good language for that. I think Perl 6 is a better language because Perl 6 doesn't need as much syntax to get the basic stuff done. There's of acres of syntax in the background but you don't need it early on.

LV The UK government has decreed this year as the Year of Code. Its representative said that it was possible to learn some code in an hour. Talking to Robert

Lefkowitz on the subject, he thought that programming is at a similar stage to when spaces were introduced between words in Latin script, which opened up reading to more people. And, similarly, stirrups were fundamental to the feudal system because they enabled riders to wield a sword and shield.

Damian: Or the zero in the number system.

LV Yes, exactly. So is that a relevant question for Perl, or is it better suited to Python or JavaScript, say, and should we just be teaching people concepts before we teach abstraction?

Damian: Wow!

LV Sorry, I've had too much coffee this morning.

Damian: No, these are fantastic and deep and important questions. Let's go back to the very beginning. Anyone who believes you can teach programming in an hour has no idea about what programming is. I think that I finally thought that I was a confident programmer maybe about four or five years ago, so after about a quarter of a century of coding. I felt that I was an ordinary good programmer by that stage. I don't think you can even teach HTML in an hour, to be brutally honest.

LV That's one of the very examples they gave.

Damian: No, no. So there's a fundamental misunderstanding about how complicated a task it is that we do when we do programming and how quickly one ought to be able to do that task. And I think we do a disservice if we try and throw people in at the deep end. And a lot of language choices throw people in the deep end. I would, for example, put JavaScript or Java in that same category.

If you try to teach people Java, just think about the Java 'Hello World' program, you see it online all the time. The Java 'Hello World' program has a class declaration and then it has a method declaration, it has the loading of libraries that make the thing work, it then has the method call chain to actually do that. And in order to even understand the presumably simplest of all programs, you have to understand

Java at about four different levels of abstraction. You have to understand a lot of very sophisticated concepts, including things as simple as what's the difference between static and non-static. Now, a lot of good programmers would not be able to tell you what the difference between static and non-static really is. So, a language like that, which is often touted as being a relatively simple language, actually isn't.

“Anyone who believes you can teach programming in an hour has no idea about programming.”

LV So you can just dive in a change things?

Damian: Yeah, and that's what people do. They don't learn to program, they learn to evolve or mutate existing programs, and that's not the same skill set. And, frankly, a lot of Perl developers are like that as well. Their only exposure to Perl is in existing large-scale scripts on which their entire organisation depends. And all that they're asked to do is go in and make a small change to that. They're not asked to develop, to design, to build, to implement. It's strictly about “let's twiddle”.

When you're looking for a language to actually get people up and running, you need a language that doesn't get in their way, that allows them to think

about the abstractions of how to express this series of instructions clearly and unambiguously. In Perl or Perl 6, Hello World is literally “say ‘Hello World’”. The thing is, I can teach someone to do that in 30 seconds, not an hour, and I can go from there if I'm very very careful about what I introduce them to next. There are other languages where you don't have to be quite as careful because there just aren't that many constructs, and they have pitfalls as well.

What's important is that we do need good programmers, we do need people who can do this stuff, because our entire society will utterly fall apart if we do not have people that can maintain our software. We are not a society that can survive if our software goes down. But to think that we can teach them in an hour, or a day, or a week, or a month or even a year, or the three years of the standard program, is highly optimistic.

LV Does that mean that, in some way, computer science has failed if we still want people to become expert scientists, when the future promised us some pseudo-code that we could just transfer our thoughts to the computer?

Damian: Yes. The future promised us a lot of things, didn't it! I'm still waiting for my flying car.

But should programming become a commodity? Eventually, for a large number of people, it will be. We will find ways whereby people can set up their environments and have the behaviour they want. But there's a fundamental mistake there in thinking things that are that complicated can be reduced down to something so simple.

LV Considering the context of the conversation, what do you think is the ideal path? Is there an ideal language to start with? How would you recommend people get started if you want to take them to Perl nirvana?

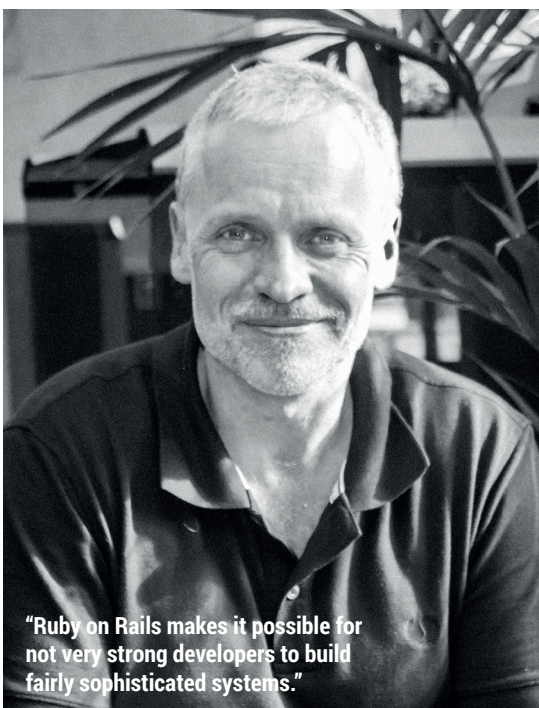
Damian: Perl nirvana! I can probably only go by my own path and by the paths that I've shown to my of students over time. And for me, the most important thing was diversity. Not being stuck thinking this is one way that we code. And I don't care if it's the one way of Python, or the one way of Ruby, or

the one way of JavaScript, or of Java, or C, or C++ or anything. I think the important thing is that if you want to become an experienced programmer, you need to be exposed to an enormously wide range of ways of thinking about coding. You need to be exposed to functional programming systems and imperative programming systems and object-oriented programming systems and declarative programming systems and concurrent programming systems. Because it's only by opening up your mind to these different views on the same reality that you really see.

It's like back in the early days of physics where everyone either just thought of light as particles or just as waves, and there was this enormous fight over which one is it. Well, the answer is both. And is programming a purely functional activity or a purely object-oriented activity or a purely imperative activity or a declarative activity? It's all of them. And what I try to do in all of the syllabuses that I ever put together and what I try to do for myself in my own ongoing learning is find new ways of thinking about what it is that I do. How can I do functional programming in C, for example? How do I do object orientation in C? Well you can do that it. It's not easy, but you can do it. So, for me, it doesn't matter what tool I'm looking at, what I want to know is how can I think of this problem in a way that makes the solution obvious and simple and correct and robust. And often that's just I need to look at it entirely differently. And so what I would encourage every young programmer, and every old programmer as well, is never give up.

Look at the new languages that are coming out. Look at the Clojures, and the Scalas and the Darts and the Gos, and all of the different languages that are constantly coming up. See what they have to give you in the way of insights about what programming actually is. Because the only way you're going to eventually understand what this elephant looks like is if you feel the various parts of it individually and realise that they are simply parts of a greater whole.

LV Brilliant. Thanks Damian.
Damian: My pleasure. LV



“Ruby on Rails makes it possible for not very strong developers to build fairly sophisticated systems.”

Free mini sample

www.linuxvoice.com

LINUX VOICE

TUTORIAL

BEN EVERARD

CUSTOMISE THE LXDE DESKTOP

Get a fantastic desktop environment without overloading your system's hardware.



The Lightweight X11 Desktop Environment – or LXDE as it's more commonly known – is popular for its ease of use and low use of system resources. It's the desktop of choice for the Raspberry Pi, and is an excellent option for replacing Windows XP on older machines. However, in its default form it is a little ugly. Everything works as you expect it to, but it doesn't show off the Linux desktop experience as well as it could. Fortunately, it's quite easy to whip the default configuration into something that looks good and is a little more user friendly.

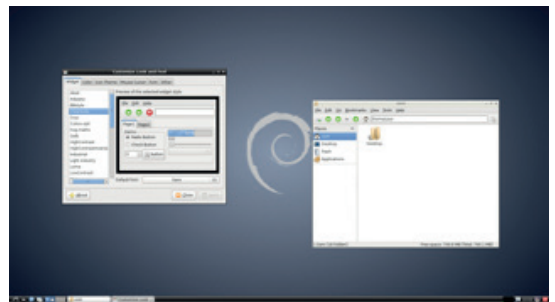
A desktop environment has a large stack of things that are really just images. These are the icons, the bits that make up the widgets (such as buttons), and the desktop background. These can all be easily swapped around provided you have new images to go in their place.

Get new wallpaper

There's no one single place for LXDE themes, but there is for Gnome, and they're mostly compatible. Head to www.gnome-look.org to see a fantastic range of user-submitted work. There are some great-looking things on there, and there are some truly terrible ones too, so take a little time to find ones you like. By default, the website shows the most recently added items, and the quality is variable. You usually need to switch to Highest Rated or Most Downloaded to find the good choices.

To switch desktop wallpapers, just save the image file that you want to use, then right-click on the desktop and choose Desktop Preferences in the menu. This will then give you the option to browse to the image file you want.

This is our LXDE desktop after tweaking. You may notice we've also changed the menu icon. This is done by right-clicking on the old icon and selecting Menu Settings.



The standard LXDE desktop: it's functional and easy to use, but with a little effort we can do much better.

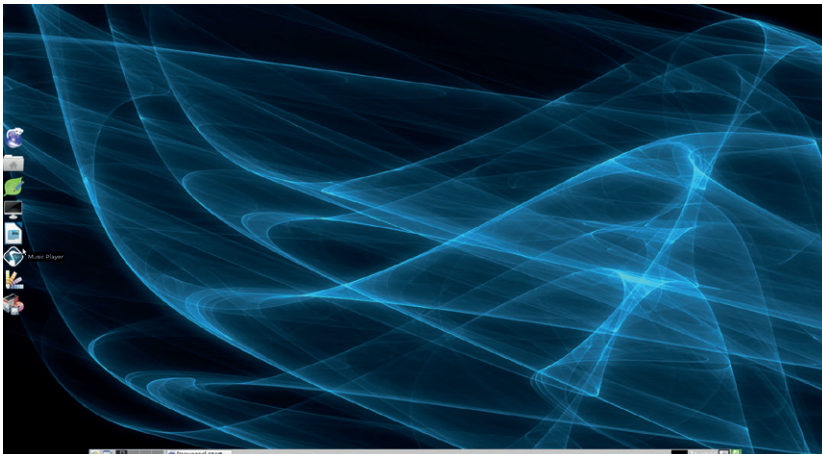
Icons and themes take a little more to change, but are still quite straightforward, since there's a tool called LXAppearance to help. First you need to download the theme. We started with the Elementary icons at www.gnome-look.org/content/show.php/elementary+icons?content=73439, though most icon themes should work.

Follow the download link to DeviantArt, then download the Zip file. In principal, it is possible to install the icon theme with LXAppearance, but in practice it's a little awkward since it only supports **tar.gz** and **tar.bz** files. We found it quite unstable when installing anything. All installing does, though, is place the files in the appropriate directories, so it's quite easy to do it without an automatic installer.

Install new icons

Icon themes should be placed in a folder called **.icons** in the user's home folder. The easiest way to do this is with the PCManFM file manager that comes with LXDE. Just open up your home folder and make sure hidden folders are displayed (you should tick the box in View > Show Hidden). If there isn't already a folder called **.icons**, you need to create it (right-click > Create New > Folder). Then just unzip the icon theme that you've downloaded (right-click it in the file manager, then select Extract To and in the folder path enter **/home/ben/.icons** – with your username instead of **ben**).

To activate the icons, you'll need to use LXAppearance. Depending on your setup, you might find this in the Applications menu under Preferences > Customise Look And Feel. If it's not there, you'll have to run it by typing **lxappearance** in the terminal. In the Icon Theme tab, you should now find the Elementary theme (or whichever Icon theme you installed).



The same basic method can also be used to add new widget themes. In **gnome-look.org**, these are under the GTK 2.x menu in the left-hand column of the screen. We went for BSM Simple (www.gnome-look.org/content/show.php/BSM+Simple?content=121685) These have to be downloaded and extracted into the folder **.themes**, and then they'll appear in the Widget tab in LXAppearance.

The eagle-eyed of you may notice that after installing, it looks a little different to how the theme looks on the main website. We'll come back to that in a minute, but for now, we'll go on with adding a dock.

Building a dock

LXDE comes with a panel along the bottom that holds most of the basic desktop utilities, such as the applications menu, window list and system tray. It can get a little cluttered, so we like to have an application launcher on the side of the desktop to provide quick access to the programs we use most frequently.

This is really just another panel, but we'll use a few tricks to make it function better for our needs. First, right-click on the bottom panel and select Create New Panel. This will add the new panel and open the Panel Preferences window. The first thing to do is get it in position on the left side. We put ours in the middle of the left-hand edge of the screen, taking up 40% of the edge, 54 pixels wide with icons 50 pixels big.

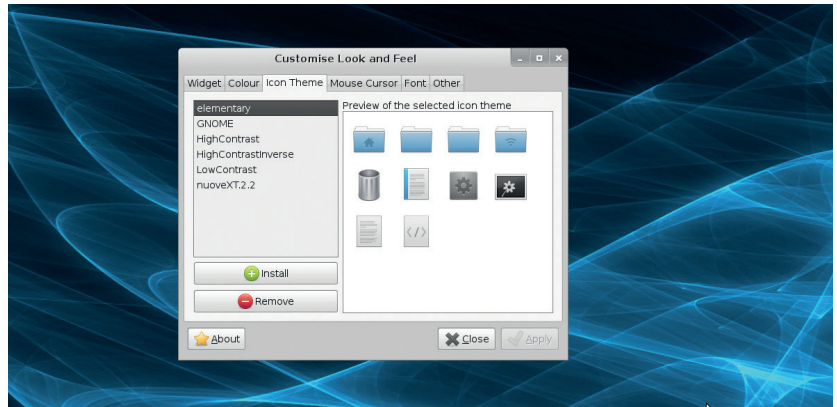
In the Panel Applets tab, add an Application Launcher Bar, then double-click on the entry in the list to open Add Applications To The Launcher. Once you've selected your favourites, you can set the appearance. In Appearance, select Solid Colour (with Opacity), then click on the colour and scroll the opacity down to 0. The final thing to keep it out of the way is to select Minimise Panel When Not In Use in the Advanced tab.

This means it won't take up any screen space normally, but you can just move the mouse to the left edge of the screen when you want to open up an application, enabling you to have nice big application launcher buttons without spoiling the look of the bottom panel with loads of clutter.

Configuration files

Almost all of the configuration we've done has been either by installing work other people have done, or via point-and-click settings. This is a simple way of getting access to a huge range of settings, and you can create wonderful desktops doing just this. However, the ultrageeks among you may be itching to exert ultimate control over everything on your desktop. Fortunately, you can.

If you want to change the appearance of the windows, you'll need to dive into the theme. Creating a new theme from scratch is a daunting task, but it's pretty straightforward to modify an existing one. The Gnome wiki has details of what the various bits are (<https://wiki.gnome.org/Artic/GnomeArt/Tutorials/GtkThemes>), and you'll find everything in text files in the folder that you extracted into the **.themes** directory.



On **gnome-look.org**, you'll see that most themes have rounded corners on the windows, but when you install them, you get square corners. This isn't a huge deal, but you'll also find a few other things that don't quite look as well as they could. The reason for this is the window manager.

Under new management


By default, LXDE uses the Openbox window manager. This is lightweight, and serves most purposes quite well. Openbox looks its best with very minimal windows, and a very clean design. A lot of people like this, but there's also a place for slightly more substance to the windows. For this, a better look can be achieved with other window managers.

Our favourite is Metacity. This is the Gnome 2 window manager. Of course, there's a trade off to this. Metacity will use a little more screen space than Openbox, and a little more CPU and memory. The difference shouldn't be much though: we tested both, and Openbox used about 0.5–1 % of the CPU time, and 1% of the memory, while Metacity used 2–3% of the CPU and 2% of the memory. By comparison, in both cases, the underlying X Windows System used 10–15% of the CPU and 6% of the memory, so while Metacity does increase the window management overhead, in most cases it won't be significant.

To switch to Metacity, first make sure it's installed. On Debian-based systems, this is done by typing the following at the terminal:

```
sudo apt-get install metacity
```

You can then make the change. Go to the Applications Menu > Preferences > Desktop Session Settings, and in the Advanced tab, change Window Manager to Metacity. You'll need to log out and back in again (or reboot) for the changes to take effect.

As you've seen, there are loads of things you can do to improve the default look of LXDE. None of these things really change the way you use the system, but they can make it a little more pleasant. We've shown you how we like it, but with a bit of experimentation, you should find a setup that works well for you. 

We think that the nice-looking Metacity windows are well worth the extra few clock cycles they take to render.

Ben Everard is a Pi enthusiast and the co-author of the best-selling *Learning Python With Raspberry Pi*.

Free mini sample

www.linuxvoice.com

Udoo

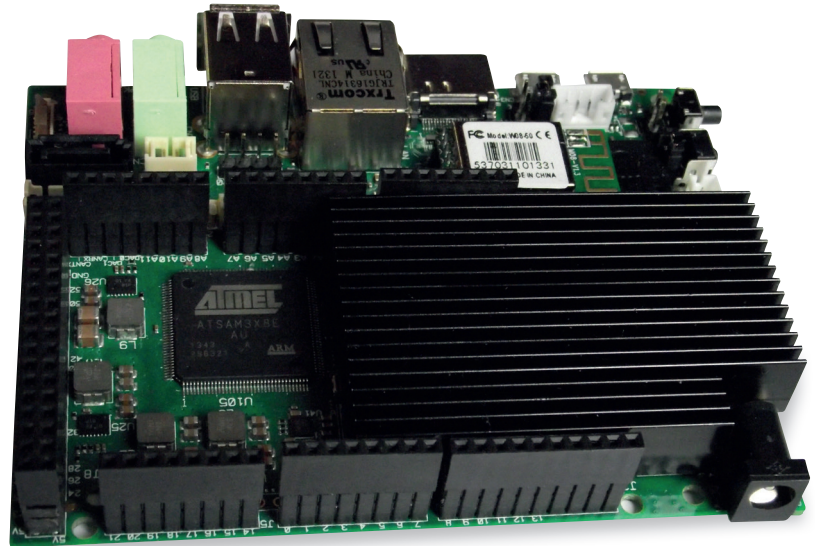
You do like small ARM computers? So does everyone these days, it seems, including Ben Everard.

The Udoo is a small ARM-based machine that runs Linux (both traditional desktop Linux and Android), and has some programmable input/output pins exposed. If that sounds familiar, it's because that's exactly what the Raspberry Pi is. The Pi has proven extremely popular, but for all its uses, it's a little lacking in hardware grunt. That's the niche that the Udoo is aimed at: simple, accessible Linux-based hacking, on a board that packs a little more punch than its fruit-based counterpart.

The CPU is a quad-core 1GHz ARM v7 (a dual-core version is also available). See the box below for benchmarks – these show that each of the Udoo's cores is more powerful than the Raspberry Pi on its own. While benchmarks provide quantitative data, qualitative data about computer performance is harder to capture. The Udoo has enough power to make the desktop feel snappy, and tasks that swamp the Pi (like browsing JavaScript-heavy websites or unzipping packages) are handled with relative ease. Put simply, it feels an order of magnitude quicker than the Pi. However, it's still no match for most x86 machines.

The Udoo uses a separate microcontroller to handle the inputs and outputs. In fact, the microcontroller and pin layout is identical to the Arduino Due, with 76 IOs including 12 analogue inputs and two analogue outputs. However, unlike most Arduino boards, the Due (and Udoo) use 3.3 volts rather than 5, so hardware designed for 5V boards won't work.

Connectivity doesn't just come in the form of IO pins: the Udoo also has a SATA connector (quad-core version only) to allow regular hard drives to connect; an LVDS connector for touchscreens (especially good if you want to build your own tablet – Udoo sells 7- and 15-inch screens); and a USB OTG connection. It also has a camera connection (camera module sold




separately). Network access is accounted for with Wi-Fi and Gigabit Ethernet on the Quad-core version.

The extra power of the Udoo comes at a cost. It's more expensive, bigger and draws more power than the Pi. All of these make it a significantly worse option for projects where the board will be included into the project physically.

An Udoo is almost exactly twice the size of a Raspberry Pi, and it packs many more connectors into that space.

Desktop replacement?

The comparison to the Pi, though, is a bit unfair. The Udoo is more than three times the cost, and while it is still cheap compared to a PC at around £110 (including taxes and shipping to the UK), that takes it out of the impulse buy range for many people.

Boards like the Udoo live or die based on whether they get enough mindshare. If there are plenty of tutorials and books available, it becomes easy to work around the limitations and compromises that are essential to all small board computers. If they don't, using them becomes more hassle than not. A quick Google search brings up about 40,000 results for 'Udoo tutorial', compared with 180,000 for 'Beaglebone tutorial' and over seven million for 'Raspberry Pi tutorial'. That's a lot less, but then the Udoo is the youngest of the three. The Udoo website explicitly pitches it as a competitor to the Raspberry Pi, and it's hard to ignore that, but we can definitely see a useful future for this device on its own merits. 

DATA

Web
www.udoo.org
Developer
SECO USA Ltd and Adilab
Price
£73–99

Udoo vs RPi performance

Benchmark	Udoo	RPi normal	RPi max overclocked
Blowfish	47.43	99.00	68.58
Cryptohash	22.39	9.07	13.28 *
Fibonacci	11.49	26.07	18.16
N Queens	41.64	84.97	69.07
FFT	48.94	149.16	101.19
Raytracing	49.02	130.38	89.84

Hardinfo benchmarks. This compares a single core of the four-core Udoo against the only core on the Raspberry Pi.

* More is better. For all others, less is better

LINUX VOICE VERDICT

The Udoo is good value for money if you're ready for a home hacking board with more power than a Pi.

★★★★★

Free mini sample

www.linuxvoice.com

SYSADMIN

System administration technologies brought to you from the coalface of Linux.



Jonathan Roberts dropped out of an MA in Theology to work with Linux. A Fedora advocate and systems administrator, we hear his calming tones whenever we're stuck with something hard.

Among developers, 'test driven development' has become trendy once more. It's certainly not a new idea, as similar practices are described in *The Mythical Man Month*, which was originally published in 1975, but it is as popular now as it has ever been.

The idea is simple. Developers write automated tests to check that functions and features they've implemented work. Usually, these tests take the form of simple functions that call the code to be tested, and compare the output to an expected value, using an assert statement or similar. If the expected and actual value match, the test passes; if they're different, the test fails.

In TDD, this is taken one step further, and advocates argue that the developer should first write a failing test for the function they're about to implement, and then they can keep working until it passes.

What's this got to do with system administration? Well, I would argue that operations teams should take a similar approach when building out the infrastructure for a new product.

Tests can take the form of checks in monitoring software such as Check_MK or Nagios. Ensure that, after your provisioning servers, your monitoring server is the first thing you install. Then, for each subsequent server to be installed, first add it and all necessary checks (process checks for Apache, MySQL server status checks etc) to your monitoring software.

Then you can begin building it. At first, all the checks will be red. But as you boot it, and then run your configuration management recipes, you'll see check after check turn green. If any stay red by the time Puppet etc has finished, you know you need to tweak your recipes.

Btrfs

The filesystem that's better (or butter) in so many ways.

In our third and final look at new technologies making their way to Linux, we're going to explore Btrfs (which in my head I'm pronouncing butter-eff-ess). Btrfs is a new copy-on-write filesystem for Linux, which aims to deliver advanced features such as volume management, snapshots, checksums and send/receive of subvolumes.

If you're not already a filesystem expert, many of those terms might sound alien to you, but continue reading and we'll do our best to explain what these features do, why you want them, and when you'll get them.

Stability

Let's start with that final question, as any one who's paid even a little attention to news about Btrfs has heard horror stories about it destroying data and may well think it's a long way from production.

As it stands now, OpenSUSE plans to be the first major distribution to use it by default in its November 13.2 release, indicating that they believe it's stable enough for daily use. Facebook, too, which has recently hired many Btrfs developers, has announced plans to begin using Btrfs in its production web tier, where it can test

performance and stability on real, albeit easily recoverable, workloads.

This anecdotal support from distributions and large production environments, along with the official wiki claiming that the on-disk format is now stable, suggests that if you want to start testing, now is the time to do so. It might not be making its way in to the next round of enterprise distribution releases as default, but it will be there for those users who really need it.

So, if you want to try some of the features we'll describe in the rest of this article, make sure you have automatic and tested backups running. Do that, and even if the 'experimental' status of Btrfs does lead to data loss, you won't be left cursing.

Getting started

With that word on stability out of the way, let's get to work and create a new Btrfs filesystem. Once we have the filesystem in place, we'll start working through some of its core features, showing what they do, why they're great and how to use them.

For our simple experiments, we're going to use some plain files mounted as loop devices. So, to start, first create an empty 3GB file, use **losetup** to create a new loop

```

Terminal - jon@localhost:~
File Edit View Terminal Tabs Help
jon@localhost:~/Projects/puppet
jon@localhost ~]$ sudo losetup /dev/loop0 /home/jon/btrfs1
[sudo] password for jon:
jon@localhost ~]$ mkfs.btrfs /dev/loop0
probe of /dev/loop0 failed, cannot detect existing filesystem.
Error: Use the -f option to force overwrite.
jon@localhost ~]$ sudo mkfs.btrfs -f /dev/loop0

WARNING! - Btrfs v3.12 IS EXPERIMENTAL
WARNING! - see http://btrfs.wiki.kernel.org before using

Performing full device TRIM (2.93GiB) ...
Turning ON incompat feature 'extref': increased hardlink limit per file to 65536
fs created label (null) on /dev/loop0
        nodesize 16384 leafsize 16384 sectorsize 4096 size 2.93GiB
Btrfs v3.12
jon@localhost ~]$

```

Creating a btrfs filesystem is just like any other – easy. In this tutorial, we've used loopback mounts to experiment, but this would all work just as well on a real disk.

Free mini sample

www.linuxvoice.com

device, and then create a new Btrfs filesystem:

```
dd if=/dev/zero of=/home/jon/btrfs1 bs=1024
count=3072000
losetup /dev/loop0 /home/jon/btrfs1
mkfs.btrfs /dev/loop0
```

That's all there is to it. You can then mount **/dev/loop0** as you would any other filesystem, examine it with tools like **df** etc.

As with any filesystem, there are a host of options you can specify at mount time to change the way that it works. With Btrfs, one useful option is **compress**, which enables you to turn on compression using either **zlib** or **lzo**:

```
mount -o compress=lzo /dev/loop0 /mnt/btrfs
```

While compression brings the obvious advantage of letting you store more data on disk, in some circumstances it can also bring a performance benefit too. On most systems without solid state storage, there are often CPU cycles to spare, while disk I/O can be a real bottleneck. By asking the disks to pull back less data, but asking the CPU to do some more work uncompressing that data, you can improve your performance.

Subvolumes

Now that you have a Btrfs filesystem available, let's look at the second (after transparent compression) feature of interest: subvolumes. A Btrfs filesystem can be divided into multiple roots that can each be treated as a filesystem in its own right (unlike logical volumes, these independent roots are not separate block devices):

```
btrfs subvolume create /mnt/btrfs/images
```

If you inspect the mounted filesystem at this point, you'll see what appears to be a new directory. You can **cd** in to it, you can create files within it etc. What happens, however, if you try to create a hard link between a file in this subvolume and the parent **btrfs** file volume?

```
ln /mnt/btrfs/images/screen1.png /mnt/btrfs/
screen1.png
```

That operation fails, just as if you'd tried to create a hard link between two different mount points.

OK, so what can you do with subvolumes? Well, when creating a subvolume, you can make it a snapshot of another Btrfs volume:

```
btrfs subvolume snapshot /mnt/btrfs/images /mnt/
btrfs/ss-images
```

Because Btrfs is a copy-on-write filesystem, this snapshot could be an exact replica of a 300GB filesystem and it would still have been created instantly. Btrfs only needs to copy data when information in the snapshot or the original volume actually

```
Terminal - jon@localhost:
jon@localhost:~/Projects/puppet
[jon@localhost ~]$ btrfs filesystem df /mnt
Data, single: total=8.00MiB, used=64.00KiB
System, DUP: total=8.00MiB, used=16.00KiB
System, single: total=4.00MiB, used=0.00
Metadata, DUP: total=150.00MiB, used=112.00KiB
Metadata, single: total=8.00MiB, used=0.00
[jon@localhost ~]$
```

Some of the tools you've used in the past, such as **df**, won't take into account metadata and other features of Btrfs, so it has its own tools, such as **btrfs filesystem df /path** (the substitute for **df**).

changes, making them fast to create and remove as well as extremely space efficient.

What really makes subvolumes useful, however, is that you can mount individual subvolumes without mounting their parent. First, list all of the subvolumes in your Btrfs volumes to find out their 'subvolume IDs':

```
btrfs subvolume list /mnt/btrfs
```

Then, assuming the **ss-images** snapshot created above has volume id 258, **umount** the Btrfs filesystem before remounting with the following options:

```
mount -o subvolumeid=258 /dev/loop0 /mnt/btrfs
```

When you list the contents of **/mnt/btrfs**, you'll only see the contents of that subvolume. This feature is particularly important because it means, for example, you can snapshot your root volume before

striped across a pair of drives, which is in turn mirrored to another pair of drives.

Aims to give the benefits of RAID 0 and 1.

- **RAID 5 and 6** Stripe data, as in RAID 0, but sacrifice some space for 'parity' information. This parity information allows the array to lose one disk in RAID 5 or two disks in RAID 6.

To set up a multi-device Btrfs filesystem like this, first create a second loop device:

```
dd if=/dev/zero of=/home/jon/btrfs2 bs=1024
count=3072000
```

```
losetup /dev/loop1 /home/jon/btrfs2
```

Then use the **mkfs.btrfs** command again, but with the following options:

```
mkfs.btrfs -d raid0 /dev/loop0 /dev/loop1
```

You can check the man page for **mkfs.btrfs** to see other options for the **-d** switch.

"A Btrfs filesystem can be divided into roots that can each be treated as a filesystem in its own right."

an upgrade, and if things go awry, remount the snapshot as your root and get back to a working state straight away.

Multiple volumes

As well as having these LVM-like features, Btrfs also shares features with traditional RAID, too. A Btrfs filesystem can be spread across multiple devices, and you can configure it to distribute the data across the devices according to one of several common RAID levels:

- **RAID 0** Striping, in which data is striped across disks, leading to improved read and write speeds. Btrfs also supports an extension of RAID 0 in which disks do not have to be the same size, known as 'single'.
- **RAID 1** Mirroring, in which data is mirrored across two or more disks of the same size. Can be faster for reads, but will slow down writes, as data has to be written twice.
- **RAID 10** Mirrored striped, in which data is

Self healing

We're close to the end of this month's overview, and there's so much we haven't touched on – file cloning, filesystem mirroring with send/receive, online rebalancing (aka changing RAID levels) and much more. Before finishing this month's section, there's one other aspect of Btrfs that I'd like to draw to your attention: Btrfs aims to be self healing.

Btrfs records checksums for each block that it writes. When it reads the data, it compares the data to its checksum, and if there's a difference, it automatically tries to re-read the data from one of your redundant copies or parity information – eg if you're using RAID1/10, 5 or 6 and Btrfs reads a bad block, you'd never know it happened unless you were to take a look in the logs.

That's all we have space for, but I hope you'll start thinking about all you could do with Btrfs when it does eventually hit your favourite distribution. 📀

CLOUDADMIN

Running make-believe boxes is virtually compulsory, suggests Nick Veitch.

Virtualisation: the options

There are many shades of hypervisor.

Virtualisation is a key technology that helped give birth to the modern cloud as we understand it. It helps run the services on the cloud and often helps build clouds too. But virtualisation is also important to developing tools to run clouds. As our foray into the 'dev' part of devops has already led us to look at how continuous integration is used (see LV002) we should also take a look at the virtualisation technologies commonly in use, their alternatives, and how they may differ from your experience on the desktop.

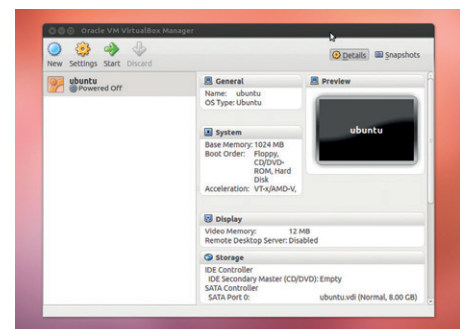
KVM

KVM works on top of Qemu, so for the purists, when we talk about KVM here we mean KVM/Qemu. KVM is a Linux-only virtualisation technology, parts of which are

included in the mainline kernel. The software relies on kernel modules to interface with the host CPU's virtualisation extensions – as such it will only run on CPUs that support (for example) Intel VT or AMD-V extensions (there is also an ARM port).

Popularity of KVM has not been driven by the desktop – it still lacks a lot of the snazzy configuration tools of VirtualBox – but it is very very popular for 'serious' use due to factors such as kernel integration and the unambiguous open source nature of the code (not to mention that it works very well).

Although it lacks somewhat in terms of graphical tools, VMs are controllable from the command line (and therefore, also easily by scripts and other software which uses the libvirt API – see our tutorial on page 94) to a greater degree than pretty much anyone



A popular choice for desktop users, Oracle's VirtualBox has some good things going for it in the developer space too.

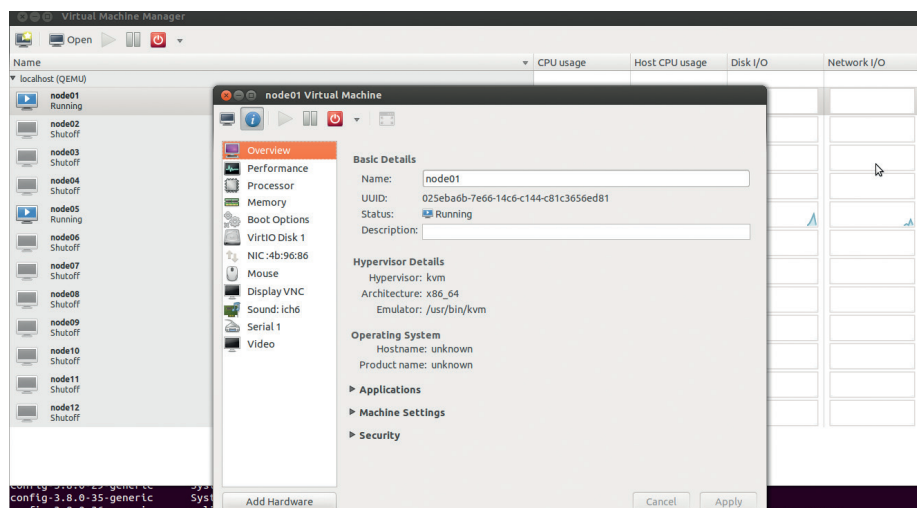
could ever want. As well as being a great tool for development, it is also widely used for spinning up VMs within clouds (eg OpenStack).

VirtualBox

VirtualBox came to prominence by virtue of it being a very featureful, well performing VM hypervisor that worked cross-platform and had an easy to understand management interface. The software has a colourful history – the original company that created it, Innotek, was acquired by Sun Microsystems, before many parts of the disintegrating Sun empire were snapped up by Oracle. As a largely open source project (there is a non-open source version, which makes use of proprietary device drivers for

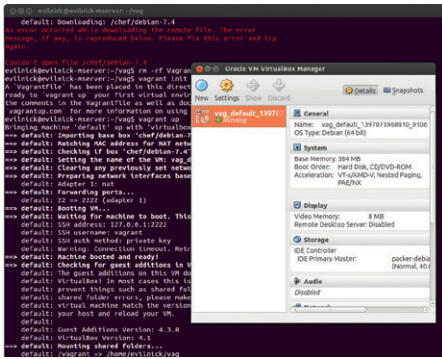
VMWare

No, we didn't forget about VMWare. Although it has been in the vanguard of virtualisation technologies for some time, VMWare is not open source. Although that doesn't exclude it from consideration in the world at large, it does tend to make it less relevant to the emerging cloud platforms, and certainly a little out of the scope of this FLOSS-loving publication.



Virt-manager is a useful graphical front-end for KVM, but you should really familiarise yourself with the virsh commandline tools, especially if you need tricky network setups.

“Vagrant was originally developed to work with VirtualBox, but a system of plugins enable it to work with numerous hypervisors.”



Vagrant is an effective tool for provisioning VMs and working collaboratively.

graphics, and the open source version seems to be stuck on an LGPLv2 licence) it sits a little ill at ease in the Oracle stable.

Nevertheless it is a mature and competent environment for running VMs. It relies a lot on paravirtualisation – special drivers that allow a more efficient throughput of data to and from the host OS. These do bring performance benefits, but rely somewhat on the co-operation of the guest OS, so if you are running custom kernels on strange distros you may not reap the full benefits.

It does have the huge advantage of also running on Mac OSX and Windows (and even Solaris), which can be beneficial in some collaborative environments.

Vagrant

Vagrant isn't a virtualisation engine, but it is most definitely worth talking about. The idea behind vagrant is that it becomes a sort of meta-manager for virtualised instances. Vagrant was originally developed to work with VirtualBox, but a system of plugins enable it to work with numerous hypervisors. Once you have installed Vagrant, you can fetch virtual machine images (which in Vagrant terms are known as 'boxes') and use them to bring up virtual machines.

You may ask yourself "What on earth is the difference between this and just creating a machine in VirtualBox?". The answer, at least at the system level, is "not much". Start up your box, and it behaves pretty much like any other VM you have initiated with

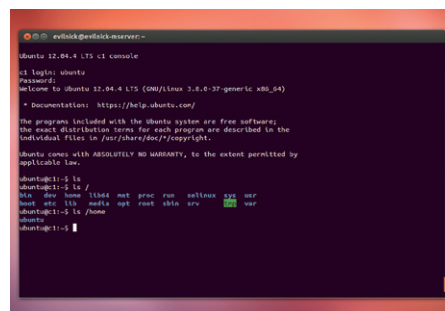
VirtualBox (or Qemu/KVM if you use that as the back-end).

The real difference is in provisioning. If you spend your life testing software, bringing up a clean VM is only part of the day-to-day grind. You then have to prepare that system for use. This ranges from the mundane installation of dependency packages to the more annoying repeatedly setting up options like host configuration or adding SSH keys so you can access the VM you created.

Yes, you can do this once in something like VirtualBox and create a snapshot image. Before you know it though you have half a dozen different snapshots that all differ in subtle ways, and aside from taking up loads of disk space, it can get pretty confusing. By using Vagrant to provision systems from a common set of boxes, you can reduce changes to your install to just changing some options in the Vagrant file that the software uses to bring up the VM. Of course, you can still create your own boxes, and there is a new service specifically for sharing those images in the cloud, so collaboration is much easier than trying to shift gigabytes of VM filestorage around.

Linux containers (LXC)

LXC is not a hypervisor for virtual machines. It is better than that; well, at a lot of things anyhow. LXC uses some very useful user-mode kernel features to containerise an implementation of a Linux OS – think chroot, but taken to extremes. Like a virtual machine, the container is able to carry out its business independently of the host OS, even to the point of running a different OS entirely. What you get is a self-contained running instance that is separate from the host OS, but which can dynamically share resources – there is no need to pre-allocate RAM and disk space for example, because the LXC container will simply consume what it needs just like any other process. LXC also plays nicely with libvirt, so you can use the same tools to



Linux containers are not a VM, and that is the whole point!

Further reading

- Why Vagrant? <https://docs.vagrantup.com/v2/why-vagrant/index.html>
- Stephane Graber's LXC primer <https://www.stgraber.org/2013/12/20/lxc-1-0-blog-post-series>
- Using KVM with Ubuntu <https://help.ubuntu.com/community/KVM>
- VirtualBox homepage <https://www.virtualbox.org>


control containers as you may use with VMs (though to be fair, there are some peculiarities of LXC that aren't adequately addressed by libvirt, but you can also use the comprehensive LXC command line tools).

As there is no CPU or hardware virtualisation, there is a much lower overhead to running containers than VMs – there are no virtualisation layers to go through, so things like file access are much faster, and the scalable resources also mean that more efficient use can be made of hardware.

There are of course, disadvantages to using containers. For a start, you can only run Linux-based containers. It can even sometimes be tricky to run completely different distributions without additional tinkering. Added to that, the lack of virtualisation also means no virtual hardware – which can be a pain when it comes to configuring networking. For simple networks, LXC makes use of a bridged driver, which means the container can access an external network through the host's network setup, but complicated VLAN topologies become more troublesome. There can also be some nagging suspicions that what may work in a container could behave differently on real hardware. Gosh, not that anyone has real hardware!

LXC arguably has the most mature support on Debian and derivative distros, and is well worth experimenting with.

The virtual future

It sometimes seem mad that we run an OS on virtual machines through cloud software, which itself can be running on virtual machines, which themselves can be running on the very same OS. Don't think about it too much, it hurts. The point is that VMs (and containerisation) provide the essential flexibility of cloud implementations, and as the overhead associated with them gets smaller, they become more and more important enablers of future technologies. 

Docker

Missing from this VM get-together is Docker. Like LXC, Docker is a containerisation solution, and we have left it out because we will be having a very detailed look at it next issue!

AN INTRODUCTION TO SSH

Get to grips with SSH and lord it over remote machines...

If you use more than one computer then you will probably, at some point, want to do something on one that isn't in front of you.

SSH (the Secure Shell) isn't another command-line shell like Bash – it's a networking protocol that you can use to connect securely to a remote computer across an insecure network like the internet. It establishes an encrypted connection to a remote computer, executes a command there and redirects its input and output across the connection. In SSH, the shell is like a wrapper surrounding a path through an insecure network that encrypts everything sent through it.

The way most people use SSH is as a command-line to enter commands on a remote machine, which you can then work on as if it were there in front of you. This remote login is what SSH does if it isn't given a specific command.

Using SSH requires a client on the local computer and a server on the remote one. The implementation found on most Linux distributions is called OpenSSH, and both the client and server packages should be in your distro's repository; they may even be installed by default. On Debian-based systems, the client (**openssh-client**) is installed by default, but you may need to install the server on machines that you want to connect to:

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

This installs and starts the SSH server. Once you have set up a remote host you can connect like this:

```
ssh remotehost
```

where **remotehost** is the hostname of the remote computer (or you can use its IP address). You will be prompted for your password on that remote machine. SSH assumes your local and remote user names are the same unless you tell it to use a different one

```
ssh user@remotehost
```

You will see a serious-looking warning when you connect to a remote host for the first time, but assuming you're happy that what you've connected to

is what you asked for, you can enter **yes** to continue the connection.

```
The authenticity of host 'remotehost (192.168.1.15)' can't be established.
```

```
ECDSA key fingerprint is 6d:c4:cf:43:75:a5:79:e0:74:a0:b7:22:b7:da:e1:25.
```

```
Are you sure you want to continue connecting (yes/no)?
```

This happens because SSH uses public-key cryptography to authenticate any server you connect to. The server responds with its public key as confirmation of its identity but your SSH client doesn't recognise it. Your client tries to compare keys it receives with copies kept in a file at **~/.ssh/known_hosts**. When you connect for the first time, it has no copy to compare against, so it displays the warning message instead. When you respond **yes** to continue connecting, you tell your client to trust the server and it adds the received key to the **known_hosts** file.

On subsequent connections, the client compares the key sent by the server with the one previously recorded and aborts the connection if they don't match. It provides some useful information to help you investigate and resolve the problem. The connection is authenticated if the client and server keys match.

The keymaster

SSH requires that each server has a unique key that consists of the public key it sends to connecting clients and a corresponding private key that it keeps secret. Most distributions automatically generate these server host keys when SSH is installed or started for the first time.

What has happened so far is that the client and server have established a secure communications channel but you have yet to authenticate yourself as a user. The simplest way to do this is by entering a password but you can (and, arguably, should) use a key exchange instead. Before you can do this you need to generate your own key:

```
ssh-keygen -t rsa
```

which, by default, saves keys in **~/.ssh** with the private key in a file called **id_rsa** and the public key in **id_rsa.pub**. You can choose whether to enter a passphrase to protect your private key. If you do, you will need to enter the passphrase whenever you use the private key. An unprotected private key is only as secure as the file it's in.

You need to copy your public key (not your private key) to any remote server that you want to connect to, and there is an easy way to do this:

```
ssh-copy-id remotehost
```

You will need to authenticate before the key can be copied, which will require that you enter your

SSH gives a dark and foreboding warning when host keys don't match.

```

$ ssh remotehost

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@   WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!   @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that a host key has just been changed.
The fingerprint for the ECDSA key sent by the remote host is
33:a7:51:e9:a6:1d:21:71:3e:08:69:3a:8e:8a:00:19.
Please contact your system administrator.
Add correct host key in /home/myuser/.ssh/known_hosts to get rid of this message.
Offending ECDSA key in /home/myuser/.ssh/known_hosts:6
ECDSA host key for remotehost has changed and you have requested strict checking.
Host key verification failed.

$

```


password for the remote machine. If the remote SSH server has disabled password authentication then this will not work and you will need to ask the remote server's administrator to copy your public key onto the server for you. Your public key is stored on the remote server in `~/.ssh/authorized_keys`.

Beyond the command-line

Many people use SSH just to get a command prompt on a remote server, but you can do more than that. If you enable X forwarding on the server then you can launch GUI applications and have them display on your local desktop. The `-X` parameter enables this mode. If you wanted, for example, to run a Firefox browser on the remote host, you could do:

```
ssh -X remotehost firefox
```

Forwarding the X protocol over SSH is an example of 'tunnelling', and you can use SSH as a tunnel for any network traffic. The example below forwards email SMTP traffic to a mail server on the remote network:

```
ssh -N -f remotehost -L 25:remotemailhost:25
```

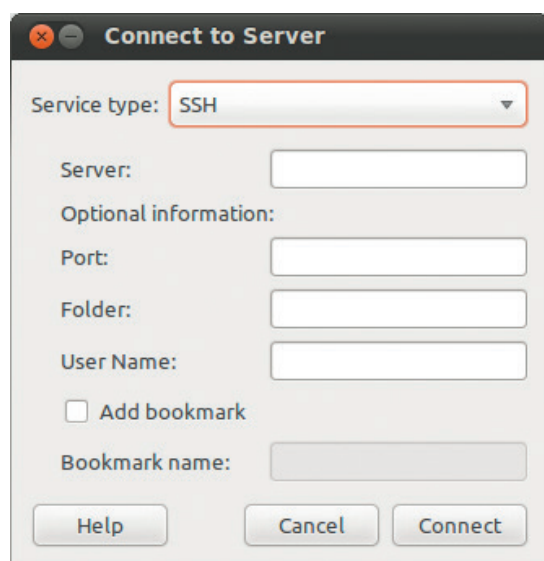
The parameters run SSH as a daemon (`-f`) without executing a remote command (`-N`) because we're tunnelling instead. The `-L 25:remotehost:25` tells SSH to listen on a local port 25 and forward any traffic it receives across the SSH connection to **remotehost** and onwards to **remotemailhost port 25**.

Another use for SSH is to copy files, and there are a few ways to do this. The **scp** (secure copy) command enables you to copy files between your local filesystem and a remote SSH server. You use it similarly to the regular **cp** command, except that the remote path is prefixed with the remote host name (and, optionally, username):

```
scp /local/path user@remotehost:/remote/path
```

You can also copy files with **sftp**, which looks and feels like a basic FTP client. It's also handled by the SSH server, so nothing additional is required to use it.

```
sftp remotehost
```



Gnome has the Gnome Virtual Filesystem (GVFS), which natively supports SSH mounts.

Server configuration

The SSH server has a configuration file, usually located at `/etc/ssh/sshd_config`. Some settings you should review are listed below.

- Set **PasswordAuthentication** to **No** to disable password authentication. Users will need to supply their public key before they can connect.
- Set **PermitRootLogin** to **No** to prevent remote logins as the root user.
- Set **X11Forwarding** to **Yes** to allow use of X applications over SSH.
- Use **AllowUsers** or **DenyUsers** if you need to restrict the users who can connect.
- Use **Banner** to display a message when a connection is established. Specify a file, usually `/etc/issue` containing the message text.

After changing the server configuration, reload it with

```
sudo killall -HUP sshd
```

If you ever need to re-generate a server's keys:

```
rm /etc/ssh/ssh_host_* && ssh-keygen -A
```

If you need constant access to many remote files, you may prefer to mount a remote filesystem and use it as if it were a local one. You can use **sshfs** to do this – it's a userspace filesystem built on top of Fuse.

```
sudo apt-get install sshfs
```

```
sudo adduser myuser fuse
```

This installs it and adds your user ID to the **fuse** group, enabling you to mount a directory on a remote SSH server:

```
mkdir ~/.mountpoint
```

```
sshfs remotehost:/remote/path ~/.mountpoint
```

You can then interact with the remote files as if they were local. When you are ready to unmount the file system, use this command:


```
fusermount -u ~/.mountpoint
```

Securing SSH

SSH is a secure protocol, but there are steps that you can take to increase its security. The first thing most admins will do is insist that users supply their public key and disable password authentication.

You may also restrict what connecting users can do. This is another use for the `~/.ssh/authorized_keys` file. By prefixing a user's key with a command, any connection will run that command regardless of what the user requested.

```
command="/usr/bin/something args..." ssh-rsa AAAAB3NzaC1yc...
```

You might expose an SSH server on the internet so that you can connect to a server in a distant location. If you do this, you can change the SSH port from its default of 22 to a more obscure port of your choice. 

John Lane is a technology consultant with a penchant for Linux. He helps new business start-ups make the most of open source.

Free mini sample

www.linuxvoice.com

PRO TIP

With the 'ssh agent' you only need to enter your passphrase the first time it's needed.

BEN EVERARD

KEY EXCHANGE: THE SCIENCE OF SECURITY

Maths and physics – two ways of keeping your data safe. Read on, bold explorer...

WHY DO THIS?

- Understand advanced cryptography techniques.
- Beat GCHQ (and the CIA as well).
- Gives you an excuse to use the laser beams that you're stuck to your sharks' heads.

We're used to thinking of secure communications in terms of encryption. If an attacker can't crack the encryption then they can't get into the data, right? Wrong. The encryption method is only one of the many parts that make up a secure exchange.

One chink in the electronic armour is the key exchange. That is, the process by which the two parties decide on which key to use for the symmetric encryption. They both have to know the key, so this has to travel between them in order to communicate. However, an attacker may be listening in, and the key has to be sent in such a way as to stop them being able to find it out.

The simplest way to agree on a key is to use public key cryptography. In this, one party can simply generate a symmetric key, encrypt it with the other's public key, and send it. Then both parties can communicate using the symmetric key.

It's a very simple method, and it works fine. Anyone who intercepts the message won't be able to read it because they don't have the private key to decrypt the message. The only flaw is the fact that the symmetric keys are used over a long period of time, and if one is compromised once, all previous messages can be decrypted. This is a particular problem when organisations like GCHQ and the NSA are intercepting and storing huge amounts of data.

Stopping this is known as Perfect Forward Secrecy (or PFS), and is possible using the Diffie-Hellman key

exchange algorithm, which doesn't use public keys. There's no long-term data that could be compromised that could be used to decrypt past data wholesale.

Perfect security

The Diffie-Hellman algorithm involves three parts that are combined to make the key. It also needs a method of combining them that is a form of encryption, which has the basic property that $(s + a) + b$ is the same as $(s + b) + a$. The '+' symbol is used here generically to mean any secure form of combination. By secure, we mean that if you know s and $(s+a)$, you can't use that to work out a .

If Alice wants to communicate with Bob, and GCHQ are trying to listen in, Alice starts by sending Bob a random number that we'll call s . This is sent in plain text, so everyone can read it. Then, both Alice and Bob make up their own random numbers. We'll call these a and b respectively. They don't send them, but combine them with s first. Alice then sends $(s + a)$ to Bob, and Bob replies with $(s + b)$.

Now they have these, Alice can calculate $(s + a) + b$, while Bob can calculate $(s + b) + a$. As we've said, these are equal, so they're used as the symmetric key.

GCHQ knows a , $(s + a)$ and $(s + b)$, but has no way of calculating $(s + a) + b$ provided we have picked a suitably secure method for combining the numbers. As with all encryption methods, if a suitably powerful computer could be found, then it would be possible to subject this to a brute-force to find the key. However, this would have to be done separately for each run of Diffie-Hellman rather than just once, making it a far less attractive proposition.

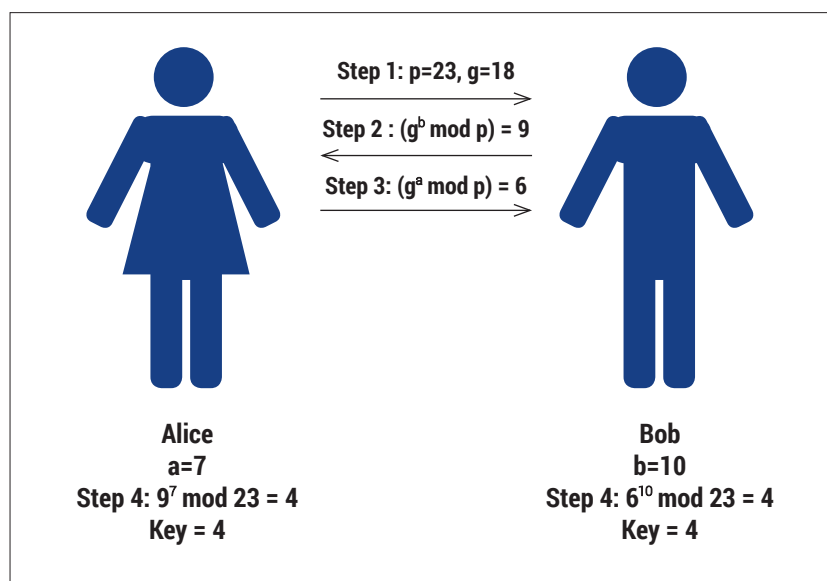
The method that Diffie and Hellman used to combine s , a and b is the fact that for a prime number (p), and a primitive root (g), $(g^a \bmod p)^b \bmod p = (g^b \bmod p)^a \bmod p$. Therefore, both p and g are sent in plain text first, then $(s + a)$ was $g^a \bmod p$ and $(s + b)$ was $g^b \bmod p$.

A quantum of security

Diffie-Hellman requires us to have a method of combining the numbers that can't be broken. While there are several options with no known weaknesses, it's possible that a way will be found to decompose the messages $(s + a)$ and $(s + b)$, and this would allow an attacker to break the encryption.

Instead of relying on mathematical properties to allow you to transmit the data securely, you could rely

GCHQ can listen in on steps 1, 2 and 3, but they won't be able to find out the key.



on physical properties. If our key exchange is protected by the laws of physics, we can be far more confident that GCHQ isn't listening in.

This is possible using the Quantum Key Distribution algorithm. In this method, each bit of information is encoded as a single photon sent from Alice to Bob. The data is in the polarisation. If you think of the light wave travelling through space, the wave could be moving up and down, side to side, or at any other orientation. This orientation of the wave is its polarisation

It is possible to measure the polarisation, but not precisely. Because of quantum indeterminacy, you can only measure it against two perpendicular axes. For example, if you set your axes as vertical and horizontal (0 and 90 degrees), and a photon is polarised at 0 degrees, you'll get a reading as vertical, and likewise for a horizontal photon. However, if a photon has a polarisation of 45 degrees, there's a 50% chance you'll get a reading of vertical and a 50% chance of horizontal.

What's more, by reading the state of the photon, you destroy it.

Using these two properties, Charles Bennet and Gilles Brassard developed a system to send a key so that it can't be intercepted. Again, we'll look at an example where Alice sends a key to Bob and GCHQ tries to listen in.

Alice has a photon transmitter that can send polarised photons in four different orientations: 0 degrees, 45 degrees, 90 degrees and 135 degrees. These are in two groups: 0 and 90 are vertical, while 45 and 135 are diagonal. For each photon, she randomly selects to use either vertical or diagonal. In vertical, 0 degrees represents a binary 0 and 90 degrees represents a binary 1. In diagonal, 45 is 0 and

Public and symmetric key encryption

There are two different types of encryption: public key and symmetric key. In public key encryption, everyone has two keys, one public, one private. The public key is made public, while the private key is known only to that user. When someone wants to send data to the user, they can encrypt it with the public key, and then it's only decryptable with the private key.

In symmetric key encryption (sometimes known as private key

encryption), there is just one key to encrypt and decrypt the message.

Symmetric key encryption is much faster the public key, and so is used for almost all purposes except authentication. SSH, for example, will use public key encryption to make sure that the server you're communicating with is really who it says it is, and once that's done it will negotiate a symmetric key using one of these key distribution algorithms.

90 is 1. She then sends a series of 0s and 1s with photons and switches between vertical and diagonal at random.

Bob has receiving equipment that he can set up at vertical and diagonal orientations as well. However, if he is set up vertical while Alice is set up diagonal, he will receive the photon incorrectly, and likewise if he is diagonal and Alice is vertical. As Alice sends her stream of ones and zeros, Bob also randomly changes between vertical and diagonal.

Keep GCHQ in the dark


After Alice has sent a long enough string of bits, she sends Bob a list of what orientations she was using for which bits. Bob compares this to how he had his receiving equipment set up. On average, they should have had the same

orientation for half of the bits, so Bob replies by saying which bits he was correctly set up for. Both of these messages can go unencrypted since they are

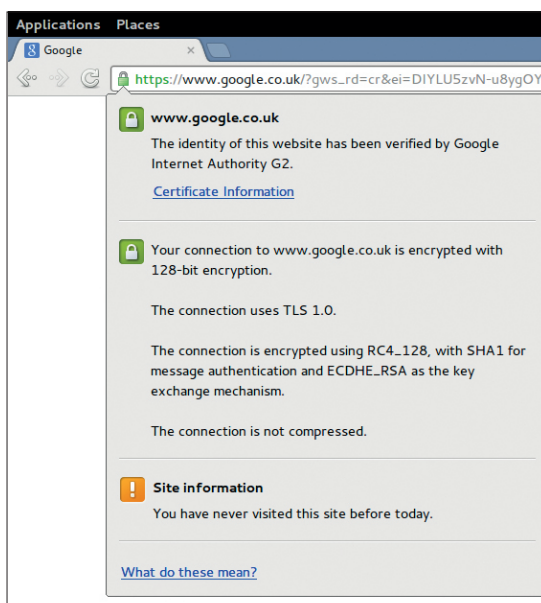
no use to an attacker. Alice and Bob can then use the values of bits that Bob received correctly as the key.

GCHQ can't intercept the photons since they don't know what orientation Alice is as she sends them.

For example, If Alice is vertical, and GCHQ intercept the photon, they have to guess between vertical and diagonal. If they are diagonal, then there's a 50% chance that they will read it incorrectly. In reading in, they destroy the original photon, so they have to create a new one to send to Bob. They have no way of knowing if they read the original one correctly, so they can't be sure either what value it is or what orientation Alice was in. They may get lucky on a few photons, but if they're building up a key of 512 bits, then the errors will quickly mount up.

This might sound fanciful, but there are already some implementations, and April 2014 marks the tenth anniversary of the first bank transfer protected by Quantum Key Distribution (see www.secoqc.net/downloads/pressrelease/Banktransfer_english.pdf for details). 

“Quantum Key Distribution sounds fanciful, but there are already some implementations.”



To see if you're using PFS, look at the technical details of the certificate. If you see ECDHE (Eliptic Curve Diffie Hellman Ephemeral) or DHE (Diffie Hellman Ephemeral), then you have perfect forward security.

Free mini sample

www.linuxvoice.com

SUBSCRIBE

shop.linuxvoice.com

Not all Linux magazines are the same



Introducing **Linux Voice**, the magazine that:

LV Gives 50% of its profits back to Free Software

LV Licenses its content CC-BY-SA within 9 months

12-month subs prices

UK – £55

Europe – £85

US/Canada – £95

ROW – £99

7-month subs prices

UK – £38

Europe – £53

US/Canada – £55

ROW – £60

**DIGITAL
SUBSCRIPTION
ONLY £38**

Each month **Linux Voice** includes 114 pages of in-depth tutorials, features, interviews and reviews, all written by the most experienced journalists in the business.

Payment is in Pounds Sterling. 12-month subscribers will receive 12 issues of Linux Voice a year. 7-month subscribers will receive 7 issue of Linux Voice. If you are dissatisfied in any way you can write to us to cancel your subscription at subscriptions@linuxvoice.com and we will refund you for all unmailed issues.

Free mini sample

www.linuxvoice.com