



RASPBERRY PI MODEL B+: UNLOCK THE SECRETS OF THE NEW PI

LINUXVOICE

**115 PAGES
OF LINUX
LEARNING**

October 2014

FREE SOFTWARE | FREE SPEECH

OVERCLOCKING
RASPBERRY PI

Push the limits of your Pi's performance

BIG DATA
PYTHON

Analyse huge data sets and draw pretty graphs

DESKTOP ENVIRONMENT
KDE 5

Take an early look at the desktop of tomorrow



BEST DISTRO

Which Linux flavour is right for you? Explore the cream of the crop to find your next favourite distro!

2014

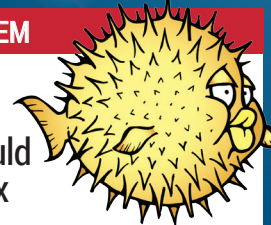
34+ PAGES OF TUTORIALS

- UDP Use the protocol that underpins the internet
- PYUSB Reverse engineer a driver with the power of Python
- BASH SCRIPTING Program your machine with the humble terminal

OPERATING SYSTEM

BSD

The OS that could have been Linux



OLD CODE

KONRAD ZUSE

Creator of the Z3 and writer of Plankalkül



ISSN 2054-3778

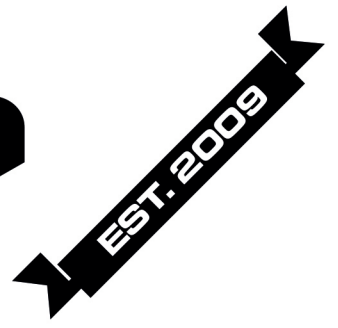
9 772054 377001

October 2014 £5.99 Printed in the UK



109

OGGCAMP



LEARN / TEACH / PLAY

A Free Culture Unconference

◀ October ▶ 2014 ▶

Mon	Tue	Wed	Thu	Fri	Sat	Sun
29	30	1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31	1	2
3	4	5	6	7	8	9



Presentations from attendees welcome,
bring your idea or project!

FREE ENTRY

Tickets and info at oggcamp.org

Supported by:

The
OGGCAMP

Community

OggCamp is seeking sponsors!
Tweet @oggcamp or check the
website for details



From openness comes literacy

The **October** issue

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 relicense 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.

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:
Chris Brown, Russell Barnes, Chris Brown, Mark Crutch, Marco Fioretti, Josette Garcia, Juliet Kemp, John Lane, Vincent Mealing, Simon Phipps, Les Pounder, 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.

In a recent interview with Edward Snowden, there's a point at which the editor of the *Guardian* newspaper, Alan Rusbridger, is explaining about when he first sat down with Edward's leaked documents. He initially gave them to his most distinguished journalists – people with years of experience evaluating stories. At some points, they literally didn't understand what they were looking at. Alan then asks Edward, from that context, how are MPs supposed to understand the technical ramifications of the decisions they're taking, when the technical concepts are so complex.

Edward replies by saying this is probably the single most important factor to explain the failures and oversight that we've seen in almost every Western government, "We need to think of it in terms of literacy," he says. And I think he's absolutely right. This is why the UK government's decision to go with ODF as its documentation format is such a monumental decision. It's the correct decision taken from a literate perspective, and while we've still got a long, long, long way to go, this is one important step in the right direction.

Graham Morrison
Editor, Linux Voice

**SUBSCRIBE
ON PAGE 60**

What's hot in LV#007



ANDREW GREGORY

We've got a scoop on the team behind the hottest new distro around, Elementary OS, due to be released any time now **p32**



BEN EVERARD

Our interview with Mir developer Thomas Voß answers many of the questions we had about why Ubuntu didn't use Wayland **p40**



MIKE SAUNDERS

Ben grabbed the entire list of UK house sales and used his statistical genius to pull loads of ace facts from the data **p82**



CONTENTS

October LV007

Celebrate summer by staying in out of the rain and messing with your Linux machine

**SUBSCRIBE
ON PAGE 60**

BEST DISTRO 2014

One dream, one soul. One prize, one goal.
There can be only one best distro 2014!

RASPBERRY PI B+

The Raspberry Pi takes its next giant leap out of the primordial soup. Now examine its DNA!



32 ELEMENTARY OS
What once was just a set of icons is now the darling of DistroWatch. Find out how it got so good.



38 FAQ: THE BSDs
But for a quirk of fate you might have been reading BSD Voice. But why did BSD end up the nearly man?

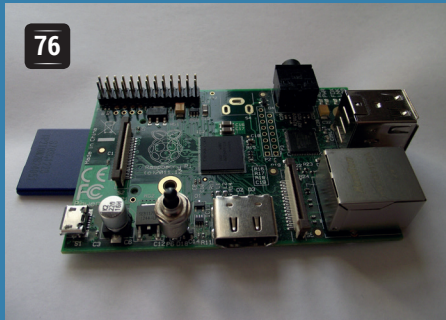


18 OSCON
Hear from some of the brightest and best in Free Software about how the future is going to look.

REGULARS

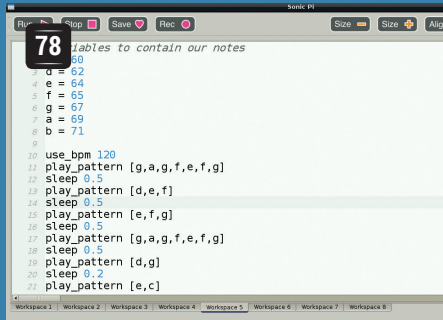
- 06 News**
Governments are saving money all over the place by adopting Free Software.
- 08 Distrohopper**
Cast a covetous eye over GhostBSD, Siduction, Zorin and OpenElec.
- 10 Gaming**
One of the true masterpieces of computer games comes to Linux: Civilisation V.
- 12 Speak your brains**
Send us your modest proposals to share with the world (and give us ideas for the mag).
- 16 LV on tour**
The kids are alright – they're busy hacking Minecraft in a field in North Yorkshire.
- 40 Interview**
Canonical's Thomas Voß tells us why Mir is the best X replacement know to man.
- 54 Group test**
Forget Facebook – we're chatting like it's 1999 on IRC. Find the best client for you!
- 60 Subscribe!**
Never miss another issue – and get access to our archive of Linux Learning.
- 64 Core technologies**
Get to grips with the glue of the internet – the essential UDP protocol.
- 68 FOSSPicks**
Free Software that's fresher and free-er than an unlaidd free-range egg.
- 110 Masterclass**
A brace of systems for sharing files with Windows machines: Samba and SWAT.
- 114 My Linux desktop**
You may know Thomas Voß as the Mir engineer we interviewed on page 40. Here's an insight into his development den.

TUTORIALS



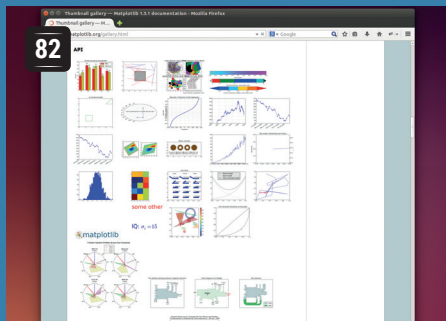
76 Raspberry Pi model B: Void your warranty

Add bits, hack bits, then overclock it and fry it. It's fun to be a geek.



78 Sonic Pi: program electronic music

Code bleeps and beats in a wonderfully simple syntax.



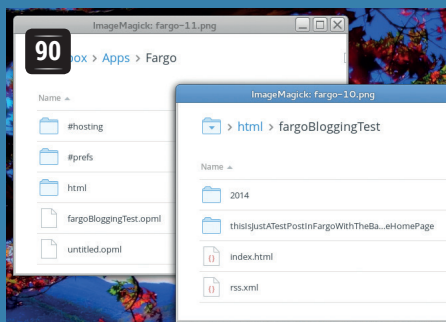
82 Python and MySQL: Big data analysis

Don't trust the official statistics – take the data and make your own.



86 Linux 101: Power up your shell

Customise the stock Bash command line and feel epic.



90 Fargo: write and publish outlines in open formats

Turn the web upside-down with a simple way to publish content.



94 Write a device driver with PyUSB

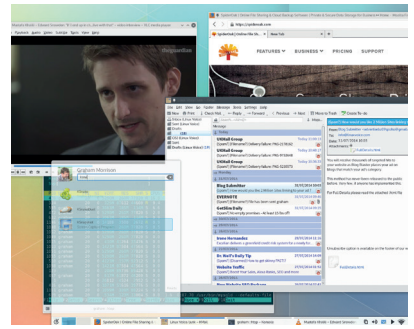
Reverse engineer the software to control a USB toy car.

100 Bash: Beyond the command prompt
Automate tasks for more control.

102 Code Ninja: Programmers' golf
Show off your coding skills.

106 Konrad Zuse: The German Turing
Computing in 1943 Berlin.

REVIEWS



46 KDE 5
Desktop, eye candy, and incubator for some of the finest software around. KDE is back.



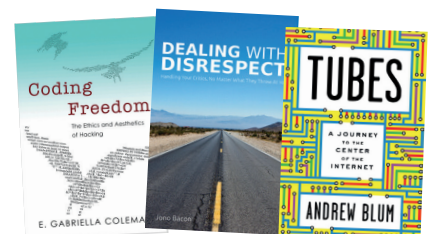
48 CamJam EduKit
Learn to build and program circuits for pennies. Next stop: robot sharks with lasers.

49 Mathematica 10
A hugely powerful data analysis tool for professional users and deep-pocketed individuals.

50 LibreOffice 4.3
The flagship office suite that's saving millions of pounds in unspent MS Office licence fees.

51 Stellarium 0.13
Explore the night sky with this absolutely superb Free Software observatory.

52 Books The ethics and aesthetics of hacking, a Quixotic search for the internet and more.



NEWS ANALYSIS

The Linux Voice view on what's going on in the world of Free Software.

Opinion

ODF comes of age

The UK Government's decision to standardise on ODF means Microsoft has lost.



Simon Phipps is president of the Open Source Initiative and a board member of the Open Rights Group and of Open Source for America.

At the start of the new millennium, a team of us at Sun Microsystems decided we had to do something about an obvious problem. A decade later, we're seeing the fruits of our labour in the decision of the UK government to prefer openly-created, openly-maintained open standards for document formats.

It was clear to us back then that open source software was being severely limited by the near-monopoly of *Microsoft Office* on the desktop. There were several pretty good alternatives available, including the *StarOffice* product we had just acquired with StarDivision and then open sourced as *OpenOffice.org*. All the same, whenever any business tried to adopt our product, subtle incompatibilities with the way it handled documents would emerge and, as the newcomer, *StarOffice* would be blamed whether it was at fault or not. The need for *Office* was transmitted not by superior functions or performance, but by the need for an interoperable document format.

Worse, Microsoft kept releasing new versions with slightly different document formats, forcing unwanted upgrades on their customers in order to remain compatible

and ratcheting up the interoperability issues for competing code. Format lock-in was what was killing the market. Even worse, it was one of the factors stifling the open source desktop, since businesses were loath to adopt a desktop solution that had no interoperable document software.

Open standards

We decided that what was needed was an open standard. If an open document format existed, every product – including *MS Office* – could compete on its merits alone, without the distortion of format lock-in. We decided to donate the work we had been doing on a new, XML-based document format for *StarOffice* to a standards body and then invite everyone else in the industry to use that as a base to collaboratively evolve a truly open standard. We selected a standards body called OASIS, both for its focus on XML and for the fact that other large vendors – notably including Microsoft – were top-level members.

The initiative was well received and a large group of contributors came together to work in the new OASIS Open Document Formats for Office Applications Working Group – OpenDocument to its friends. Microsoft were directly invited to participate, but chose not to. The rest, as they say, is history. Today, Open Document Format (ODF) is an ISO standard and is supported in every serious document application on every platform.

That history is the reason I was so delighted in July to heard that the UK Government has chosen to set ODF as the standard for all documents intended for

further collaboration. They have also chosen PDF/A and HTML as standard formats for final-form documents.

Which begs the question: has Microsoft lost? The answer to that is both yes and no. Yes, its initial refusal to join the ODF TC at OASIS and its costly and reputationally damaging foray into standards gerrymandering with its own XML-based OOXML format were definitely a “lose” of their own making. Even today, despite having got OOXML accepted as an ISO standard, Microsoft don't support the actual standard itself in its product – only a non-standard variant. It has even had to implement full ODF 1.2 support.

We're not there yet...

But that final compromise is what prevents the UK Government's standardisation on ODF being another “lose” for them. Unlike Google Docs, Microsoft has good support for ODF in both *Office* and 365 that interoperates well with other software as long as you avoid proprietary fonts and marginal features in each product. Moreover, Microsoft has other layers of lock-in to fall back on – proprietary support for *Sharepoint*, for example. Competing solutions like *LibreOffice* – the successor to our original *StarOffice* work – don't have a truly level playing field, but at least get to enter a team in the league.

All the same, our original vision of a truly open document format – royalty free, with no platform dependencies, created and maintained in the open – has finally fruited. Arguing against ODF as the preferred format for citizen collaboration will hopefully be seen as self-harming by vendors. That's one less barrier to the open source desktop and one more foundation stone for the digital freedoms of the meshed society. Let's keep going – we can do this!

“Our vision of a truly open document format with no platform dependencies... has finally fruited.”

CATCHUP

Summarised: the biggest news stories from the last month

1 British government adopts open formats

By Jove! This is jolly good news, eh chaps? Yes, the UK government has chosen ODF (as used in *Libre/OpenOffice*) and PDF/A as standard formats for sharing and viewing documents. No longer do users need to buy proprietary software to work with files from government websites, but the use of open formats will also make it easier to access data in the future. It's a slap in the face for Microsoft and its OOXML format as well. Time for a cup of tea to celebrate!

2 Valencia saves €36m by switching to Linux

Another success story: the autonomous region of Valencia in Spain has finished the next version of its customised Linux distribution, as used on over 110,000 PCs in schools. The local government claims that using Linux has saved them €36m over the last nine years – and more savings are to come. Linux has also been thriving in another region of Spain, Extremadura, where Linux is installed on 70,000 PCs and laptops in schools. <http://tinyurl.com/lv3pcwe>

3 OwnCloud 7 released

OwnCloud keeps going from strength to strength, and version 7 brings improved sharing features, a faster interface, and support for Microsoft Word files in the document editing tool. See the full list of new features here: www.owncloud.org/seven



4 Linus Torvalds releases shiny new kernel 3.16

Kernel releases aren't as exciting as they were in the last decade, largely because Linux has matured and stabilised a lot. Still, 3.16 brings a bundle of updates, including better ACPI and power management on Intel CPUs, improved Radeon graphics support, and Btrfs fixes galore. If you're a Dell Latitude user, your hard drive will now stop if you drop the machine thanks to a new Freefall driver. And the best news? 3.16 is due to be included in Debian 8, codenamed Jessie.

5 Geneva class rooms switching to FOSS

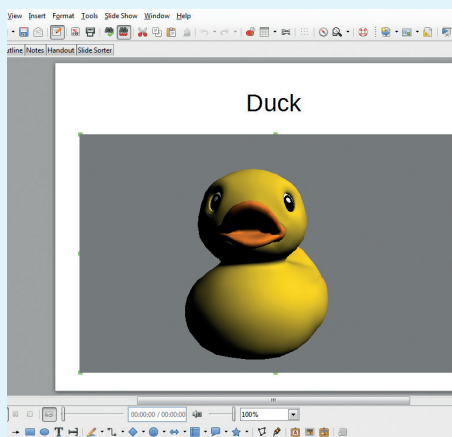
Not to be outdone by Spain, the Swiss canton of Geneva plans to switch its entire school system over to GNU/Linux. Specifically, the move to Ubuntu in 170 primary schools has already been completed, and the transition in secondary schools is planned for later in the year. The canton's IT department claims that Linux is easier to maintain, faster, safer and more stable than the proprietary software it was using before. <http://tinyurl.com/o4m9b4w>

6 Microsoft concedes: Windows has just 14% market share

This can't be right – surely? Windows is still dominant on desktop PCs, isn't it? Well yes, but the desktop is just one part of the wider computing world today, and Microsoft's very own COO Kevin Turner has recognised this. As more people are doing work on tablets and large-screen smartphones, Windows on the desktop is looking less relevant, and when you add up the whole desktop and mobile market, Windows has a paltry 14% share. While Android grows and grows...

7 LibreOffice 4.3 released: "you can't own a better office suite", apparently

That's quite a bold statement from The Document Foundation, but we can attest that *LibreOffice 4.3* is packed to the brim with new features. There's 3D models in *Impress* (the presentation tool), much better support for Microsoft's OOXML formats, and improved commenting facilities (useful in collaborative projects). Tons of fixes have been made as well, and the suite will be heading to a distro near you very soon. www.libreoffice.org



8 Toulouse saves €1m by switching to LibreOffice

And another success story to end with. The French city of Toulouse has saved €1m by switching its PCs from Microsoft software to the *LibreOffice* suite. Sure, it isn't a full Linux transition and the savings aren't as huge as in Valencia, but it's a great step in the right direction, especially with the European economy still in dodgy times. Plus, the city's money goes back to local tech support companies, and not giant megacorps overseas. <http://tinyurl.com/ovgpj2l>

DISTROHOPPER

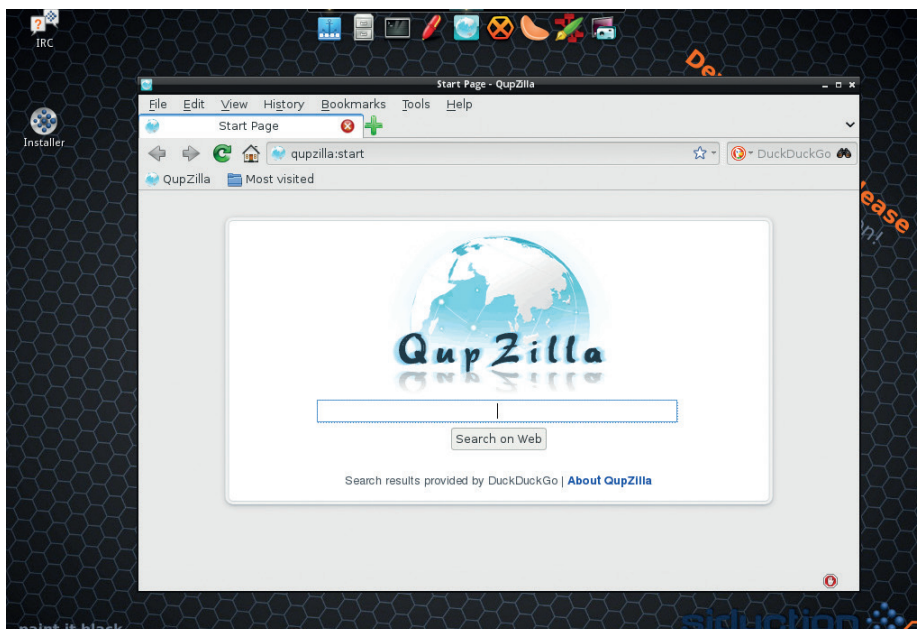
We've tapped GCHQ's communications to find out what's going on in distro land.

Siduction LXQT

Showcasing a new desktop.

Many, many words have been written about the shift from Gnome 2 to Gnome 3, and we won't add more here. However, as well as the desktop shifting, the *GTK* toolkit also shifted, and a lot of other desktop environments relied on that *GTK 2*. LXDE was one of those environments, and they weren't happy with the direction *GTK 3* was taking, so the developers have decided to make a clean break and switch to the *Qt* toolkit that's most famously used in KDE.

This new desktop environment, known as LXQT, is still in development, so not many distros have included it yet, but Siduction has. At first it feels a little strange, because some parts are reminiscent of KDE, such as the control centre and the notifications area. However, on the whole, it's a very different desktop environment. There's far less eye candy, no glow behind the window, no



Siduction was also one of the first distros to feature *Razor-qt*, another *Qt*-based desktop.

cashew, and none of the KDE apps are included. *PCManFM* is still the file manager (like the desktop, it's made the transition to *QT*), and *Qupzilla* is chosen as the web

browser. Despite the new toolkit, it still retains the no-nonsense feel of LXDE, and we suspect it will retain its popularity in low-end desktops.

Zorin 9

Can Zorin 9 help Linux attract new users from Windows?

Zorin pitches itself as the gateway to Linux, a tagline that sums up its goal of being a distro for non-technical users new to Linux. As far as we can see, this claim is mostly made based on the fact that it's themed to look like Windows.

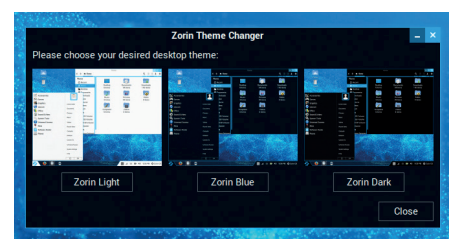
Zorin gives the user three themes to choose from to customise its look and feel (though we can't tell the difference between Dark and Blue), and three looks (based on Windows 7, Windows XP and Gnome 2). That makes a total of nine (or six) different visual appearances you can have. If one of them isn't to your taste, go find another distro. This might seem an anathema to the mantra 'Linux is about choice', and it

probably is, but that mantra isn't central to Linux (see <http://islinuxaboutchoice.com>).

Zorin is aimed squarely at new users, and new users don't always want lots of options to endlessly tweak the interface. They want a few choices so they can find a look and feel that somewhat approaches their natural style. For these people, Zorin provides a natural choice.

The easy option

If you're thinking that this GUI only alters some config files that you could go in and change to tweak this or that element of the interface, you're probably correct – but again, Zorin is not the distro for you.



The Zorin Theme Changer gives a simple way to tweak the look and feel of the desktop.

Underneath, Zorin is based on Ubuntu 14.04 which makes a solid, if unremarkable, base.

Zorin is probably the best Windows-alike distro, but we're becoming less and less convinced that Windows-alike distros are really necessary. We've found that even lifelong Windows users take to interfaces like Mate or Cinnamon without too much trouble, but for those who can't cope with the change, there's always Zorin.

GhostBSD 4.0

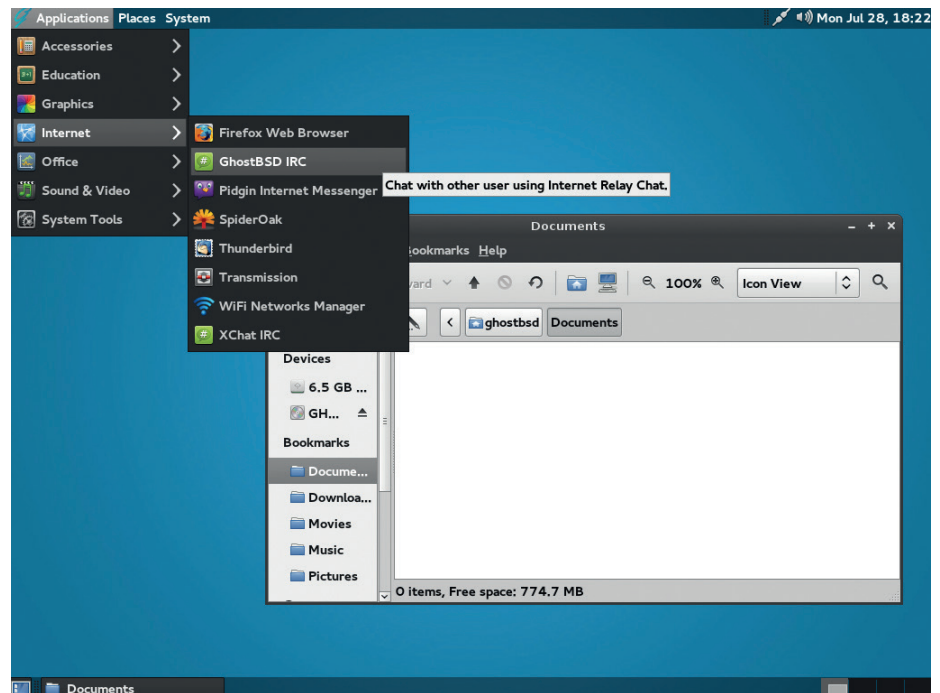
FreeBSD + Mate = an easy introduction to the mysterious world of BSD.

GhostBSD is a project that builds on FreeBSD with the aim of making it a bit more palatable for desktop users. To this end, it comes with the Mate desktop environment and a set of tools geared towards desktops rather than servers.

The basic install leaves you with a fairly minimal system, but there's nothing essential missed out. It's got *LibreOffice*, *Firefox* and a few other tools, but depending on what you hope to use it for, you'll probably need to install a bit of software to get a useful desktop. This is quite a surprise, since the ISO comes in at a fairly large 1.2GB, but shouldn't cause any problems.

Privacy enthusiasts will be pleased to see that *SpiderOak* is included by default. This is a cloud file backup service similar to *Dropbox*, but the files are encrypted on your device before uploading, which makes it far more secure. This backup software has gained a bit of popularity recently thanks to a public endorsement by Edward Snowden.

Another slightly unusual program is *Fish* as the default shell (others are available). This works in basically the same way as *Bash* (the default shell environment in almost all Linux distributions), but comes with far more graphical niceties, which can come as a bit of a shock to people used to simpler shells.



GhostBSD comes with an IRC client that will connect you straight to a GhostBSD channel – perfect for when you're having a little trouble and just can't find the solution.

GhostBSD is relatively easy to use, though not quite as beginner-friendly as some Linux distros. It also has a smaller user base than common Linuxes, so you're not likely to find as much help online should you get stuck (the Ubuntu and Mint forums are still the

best place to get answers to most questions that beginners may have). That said, it's not hard to use, and anyone with a basic knowledge of Unix-like systems who is comfortable on the command-line shouldn't have too much trouble.

OpenElec 4.1.1 Time to take advantage of the new audio on the Raspberry Pi B+

It's never been a secret that the Raspberry Pi makes a good media player. In fact, the SoC (system-on-chip) at its heart was originally intended for set-top boxes. However, up until now, it's been let down by the poor quality of its analogue audio. This hasn't been a problem for people plugging Raspberry Pis into TVs, since the HDMI audio has always been good. However, Pis have been almost useless for anyone wanting to plug them into stereos.

With the improved sound on the new model B+, the situation is a lot better. There's also a new version of the OpenElec media player distro, and we took this as a sign that we should hook our Pi up to our stereo and rock out in the name of investigative journalism.

Installing was simply a case of downloading a tarball and running a script that sets everything up. This is a touch more complex than the Noobs install method (which is also possible for OpenElec). Once this is done, you just need to pop the SD card in your Pi and start the machine. For the initial setup (entering Wi-Fi passwords, etc), you'll need a monitor and mouse, though once set up, these aren't needed any longer. Just connect the

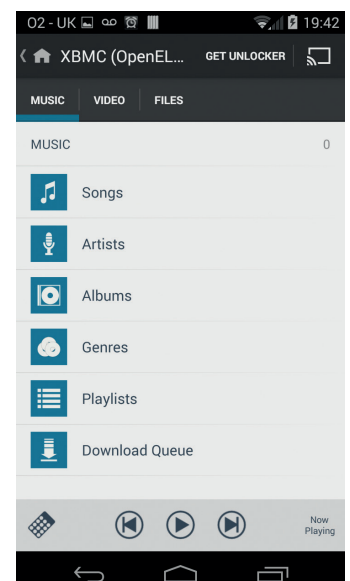
audio out into your sound system's line in, and you're ready to go.

You can control *XBMC* (the media player upon which OpenElec is based) using a smart phone app. There are several options available for most types of smartphones including an official one created by the *XBMC* team. (*XBMC* is more commonly used as a video player, and OpenElec performs admirably at this as well).

The SD card has a partition that mounts at `/storage` and contains all the media. Adding new music is just a case of using the `scp` command to copy it from another computer to this location on the Pi.

The whole setup took us less than half an hour, and then we had a smartphone controlled sound system.

We found the *XBMC* remote by Music Pump to be a little easier to use without a screen than the official app from the *XBMC* team, but it's not open source.



GAMING ON LINUX

The tastiest brain candy to relax those tired neurons



BOWLED A GOGLY



Liam Dawe is our Games Editor and the founder of gamingonlinux.com, the home of Tux gaming on the web.

GOG.com (formerly Good Old Games), the biggest DRM-free gaming store on the internet, has officially launched support for Linux games old and new.

This is some of the biggest news in Linux gaming since Valve started paying attention to Linux with Steam. The reason for this is that **GOG.com's** standards closely align with those of Linux itself in the respect of freedom and value: if you cannot get your game to run (and your system is capable), then you can get a refund. That is great, as to get a refund from somewhere like Steam is damn-near impossible a lot of the time.

DRM is also a major issue for a lot of Linux gamers and is one of the reasons many people use Linux instead of products from a company like Microsoft or Apple due to the DRM mechanisms put into it. This is the same for gaming, as a lot of games may require nasty things like an always-on internet connection even for single-player games. You won't find anything like that on **GOG.com**, as they hate DRM as much as we do. GOG also wraps up really old Windows & DOS games that would never get a proper native port using DOSBox and Wine, but unlike other stores that have some Wine-wrapped games, **GOG.com** clearly label a game using Wine.

We see a fair few complaints from Linux gamers about games being on Steam and not DRM-free, so it's time for those gamers to put their money where their mouth is and support this excellent DRM-free store.

Civilization V

Civilization's finally back on Linux!

One of the most popular PC gaming franchises ever is back on Linux.

We haven't seen a *Civilization* game on Linux since Loki Software ported *Civilization: Call to Power* in 1999, so it's incredible to see it back on Linux considering how popular it still is today. The game is consistently in the top 10 most played games on Steam and with good reason, as it can get a bit addictive.

Civilization V is a strategy game where you're tasked with controlling a single nation as you fight or make peace with the other nations in the world. You build cities, research new technologies, and it's



awesome. You get to pick what nation you play as, which will affect your starting abilities and how you progress through the game.

There isn't a traditional campaign mode with a linear story; instead it's more of a

do-as-you-please type game. You could play as a warlord pillaging towns and cities, or as the most peaceful nation on earth – the choice of how you play is up to you.

<http://store.steampowered.com/app/8930/>

Darksiders

Prepare to get brutal with your enemy.

Now this is exciting! *Darksiders*, a popular action-oriented hack 'n' slash type game, is coming to Linux. It's all thanks to Leszek Godlewski (who previously ported *Deadfall Adventures & Painkiller*) of Nordic Games, so that's another big games company making the transition to Linux gaming.

Darksiders merges some excellent graphics with frantic gameplay as you battle it out with the forces of good and evil. The game itself features around 15 hours of gameplay



as you wield a massive sword and engage in epic boss battles. A gamepad is recommended for this type of

game to really get that button bashing going.

http://store.steampowered.com/app/50620

Halfway

Beautiful pixel art...

Halfway is a fantastic space sci-fi strategy game recently released on Steam, and when we say fantastic, we really mean it. The visuals and story in *Halfway* are just amazing, and that's without even getting into how good the gameplay is as well, which is fantastic.

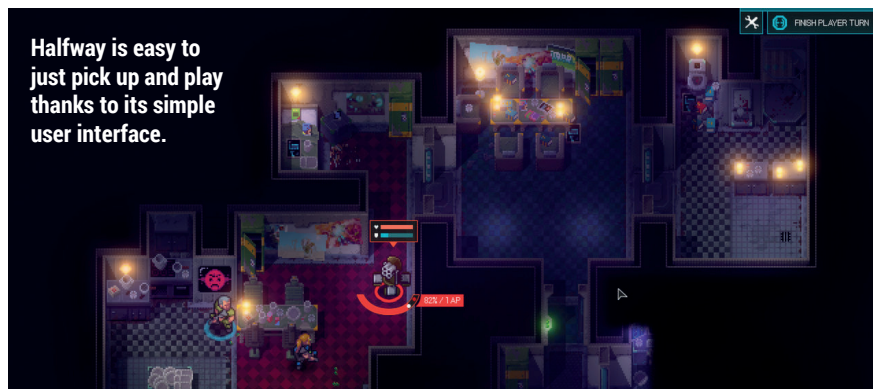
Halfway is much like *XCOM* in that it pits humans vs aliens in a turn-based strategy setting, but that's where the similarities end. *Halfway* is a much more intimate

game as you get to know your characters a bit more while searching your spaceship for those pesky aliens.

The battles are all at pretty close quarters and luckily, if you run out of ammo (it can and will happen!), then you can just sit next to a foe and smack them down melee style.

We can easily recommend this one.

<http://store.steampowered.com/app/253150>



Unity of Command

The war isn't over yet...



Unity of Command is an epic turn-based strategy game in which you battle it out against the elements as well as your enemy. The game is set during the Second World War, and you play as either the Axis or the Soviets, rather than the western allies, as is more common.

The game is quite brutal and not easy to get into, so it would be a great game for hardcore strategy fans looking for something to get their teeth into, especially those of you looking for a game with a different setting. We highly suggest trying out the tutorial first to help you get to grips with the details – and we don't suggest this very often.

[http://store.steampowered.com/app/218090/](http://store.steampowered.com/app/218090)

Rochard

Show gravity who's boss!



Rochard is one of the first games to come to Linux built with the *Unity* game engine, and it's an absolute treat too. *Rochard* is an action platformer with puzzles that pits you against the familiar threat of alien invaders aboard your spaceship.

The puzzles can get pretty tricky in this one, as you bend gravity to your will using your trusty old 'G-Lifter', which doubles up as a weapon as you throw crates at your enemy, all while running around a beautiful cartoon-like backdrop.

Like *Braid* or *Portal*, *Rochard* isn't just a run, jump and shoot game: you need to engage your grey matter to play it, and that's why we like it.

[http://store.steampowered.com/app/107800/](http://store.steampowered.com/app/107800)

ALSO RELEASED...



Grim Fandango

I'm sure that name rings a bell for a lot of you! *Grim Fandango* is being re-released and updated for modern platforms, and Linux is go!

Originally released in 1998, *Grim Fandango* is quite an old game that many gamers would have probably missed, so especially for Linux gamers this is a great chance to replay a bit of gaming history.

<http://bit.ly/U4BK0C>



Mount & Blade: Warband

Mount & Blade: Warband is an open world sandbox game that allows you to recruit your own band of merry men and participate in medieval battles across the land. You can travel the country taking quests as you see fit, but be wary about travelling at night, as bandits roam the lands and they will try to capture you. The battles are really fun and you can even fight while mounted on a horse.

<http://store.steampowered.com/app/48700>



Terraria

Rumours abound that *Terraria*, a highly popular 2D sandbox game, is coming to Linux after the developers have finished working on the Mac port. It is similar to the game *Starbound* on Linux, but not on such a grand scale.

Terraria could be seen as a 2D version of *Minecraft*, but with more interesting combat and boss battles.

<http://store.steampowered.com/app/105600/>

LINUX VOICE YOUR LETTERS



Got something to say? An idea for a new magazine feature? Or a great discovery? Email us: letters@linuxvoice.com

LINUX VOICE STAR LETTER

MANY SUGGESTIONS

I'm a volunteer at a mental health charity (www.contactmorpeth.org.uk) and I have been encouraging people there to use *LibreOffice* and *Gimp*. One particular boon was the End-Of-Lifeing of Microsoft Windows XP and Office 2003. I've been putting Ubuntu onto unwanted laptops or desktops to give away instead of sending them off to landfill. Your decision to relicense your content as Creative Commons is particularly helpful, because not everyone can justify buying a magazine.

There is one more thing I'd like you to do with your content, though. Every year or so, look back at your content and group related articles together as a PDF – for

example, a PDF of the “core technologies” articles would be good – and put them on your DVD and website.

Because resources are always a bit stretched, it would be helpful if the “Gaming On Linux” page included things like cost and hardware requirements.

Bling would be good too. Stuff like stickers (to be put on computers being given away) or posters (to advertise this stuff in the activity room). Or maybe a booklet specific to a particular topic, similar to the O'Reilly pocket references or Addison-Wesley phrasebooks.

Ian Bruntlett

Andrew says: Wow, that's a lot to go at! Well done on



We'll add prices and hardware requirements for games from next issue.

spreading the Free Software gospel – every pound that doesn't go to Microsoft is a pound that can be spent doing something better, as you know. It's always great to see when organisations realise this, especially charities that should be spending it doing important work in the communities they serve. We

know that eventually we'll have a load of content that we're not doing much with, and that's why we took the decision to relicense it as Creative Commons when the time comes. But yes, I do like what you've suggested, and we'll look into bundling collections of related articles for the web.

GROUP TESTS+

Thanks for the great magazine, I love the programming tutorials, the computer science history, the geeky Group Tests (window managers! Awesome!), and pretty much everything (also the CC-BY-SA licensing, and the DRM-free download). If it was up to me the Raspberry Pi stuff would be out the window, I really don't care, but I understand that many people want it so I can live with it.

Anyway, a suggestion for your wonderful Group Tests. At the moment they are sorted first by

feature (accessibility, installation, whatever), and then each package is assessed on that criterion. It would be easier to read if they were sorted by package (*Emacs*, *Vim*, etc) and then each package were rated on those criteria.

At present to pick the package that suits you it is necessary to read across all the sections and remember what does what and what they're all called, rather than reading about each piece of software in one section and coming to an overall judgement.

Other than that keep up the brilliant work, thanks again and I look forward to the next 6 issues and beyond.

Chris Beeley, Nottingham

Graham says: A quick defence of the Pi; if you replace Pi with Debian, almost everything Pi-ish works on other Linuxes too. Raspberry Pi has just become a great standard. Also, in a twist of wonderful serendipity, we've changed the format of Group Test exactly as you suggest. Thanks!

NOM DE MINT PART DEUX

I see that issue six of your magazine comes with a DVD; so as a digital subscriber I looked on your website for a link to download the corresponding .iso file, but saw only a message that a link will be added later. I wondered whether you might be able to send an email when this link is available in the same way that you let me know when the magazine is available? I'm particularly interested in your Raspbian remix.

I had a look at your 'Intro to Linux' videos; it left me wondering who you saw as your target audience. I have been reading Linux magazines since 2009, so most of the terms you used I was familiar with, but that wouldn't be the case if, say, my sisters had a go, as they wouldn't know the jargon. Having recently set up a laptop to dual boot Ubuntu 14.04 and Fedora 20, I couldn't work out how to set up the installation such that *Grub 2* would be installed on a */boot* partition, while I had a separate *"/"* root, */home* (one for each distro) and swap partition. So is there any chance of a video tutorial on advanced dual booting?

Galen (AKA YorkshireTyke)

Andrew says: We had some pretty banal technical problems around



LINUXVOICE DVD DISTROS | VIDEOS | PODCASTS

Linux Mint 17

Try the latest release of the world's most popular Linux distribution – 32- and 64-bit versions included!

NEW TO LINUX? GET STARTED WITH OUR EXCLUSIVE VIDEOS

Raspbian Linux Voice Edition
The best operating system for your Raspberry Pi

- >> More software included
- >> User interface improvements
- >> Free Linux Voice tutorials

Open [index.html](#) for all the details

Digital subscribers who want to take our custom version of Raspbian for a spin: go to www.linuxvoice.com/torrents/lv006.iso.torrent to download last issue's cover DVD.

the time issue 6 went on sale, but these should be fixed in time for the next issue to have a DVD on the cover(planned for issue 8). Sorry about that.

As for the videos, as long as they are useful to someone out there, they're doing their job. Keep your eyes peeled for a dual-booting guide soon.

ODF FOR UK

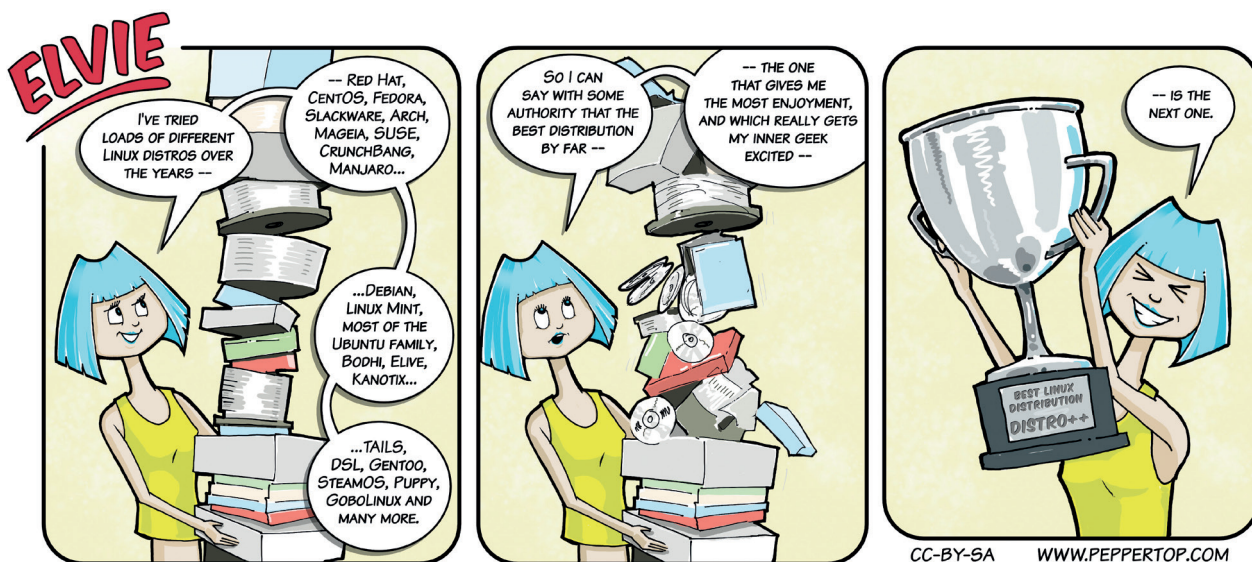
I'd like to send out a hearty message of congratulations to The Document Foundation for pushing open formats for the UK Government, and the UK Government for accepting that we shouldn't have to spend money licensing a file format in order to read data that belongs to us as taxpayers.

Unlike a lot of commenters, don't have a problem with taxpayers' money going to a big US company. If that offends you so much, then follow it to its conclusion and you end up paying British firms whether they do good work or not, just to keep spending local. That's silly. We need the best, and the best value for money, and that's why ODF is the best choice.

Here's hoping that other governments go the same way, and more organisations stop spending money on a product that only makes it more difficult for their customers to interact with them. Again, well played UK!

Remy Barrett, Worcester

Graham says: In any organisation as big as Her Majesty's Government, there are bound to be at least some sensible people who know what's going on. We too are immensely chuffed that the wise heads have prevailed in this issue.



CC-BY-SA WWW.PEPPERTOP.COM

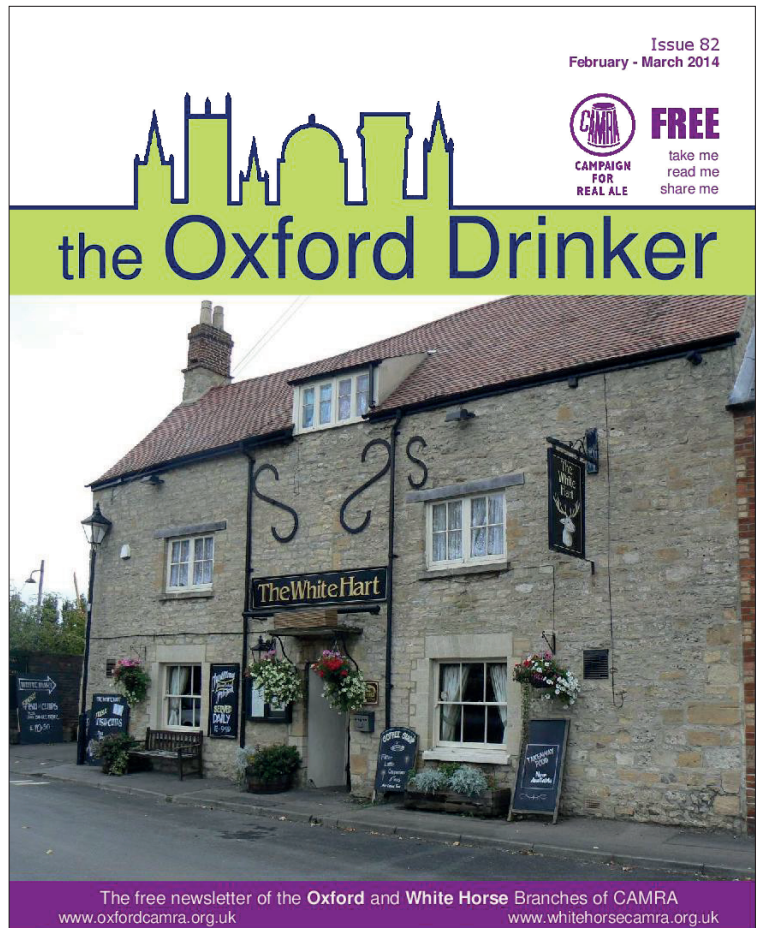
SCRIBUS

Why does Linux Voice use InDesign instead of *Scribus*? Has LV tried *Scribus* and found it lacking?

PR

Andrew says: A recent *InDesign* update followed by half a day lost due to a corrupt file crashing a machine has made us ask this same question. But the answer is still the same: the number of freelance designers in and around Bristol who know how to use *InDesign* is far greater than the number of designers who know how to use *Scribus*, so if our art boss Stacey ever wanted to go on holiday (perish the thought) we'd have an extremely small pool from which to find some cover for her (NB this is another argument in favour of open file formats).

However, we do want to keep our options open in future, and as such we've chosen open fonts that will enable us to move to *Scribus* when the situation changes. We're planning ahead.



Scribus is excellent software, used by a growing number of quality publications – including *The Oxford Drinker*, newsletter of Oxford & White Horse CAMRA.

Open Source technologists support enterprise grade, mission critical systems

Get some for your team

LINUXJOBS
.CO.UK

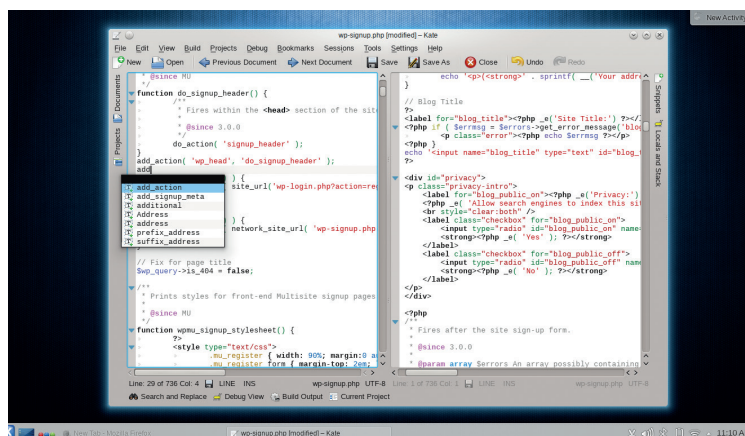
BORED WITH THE KEYBOARD

I'm enjoying the new mag, but I feel that you need a counterpoint to your chant of, "Keyboard, Keyboard, Keyboard!"

I've been an IT pro for 15 years, and regular computer user for 10 before that, resulting in about a year of RSI. Note that I've done everything I could in terms of seating position, quality ergonomic keyboards, regular stretches and screen breaks etc, and it hasn't been enough.

The biggest culprit is the long list of Ctrl key chord commands, which results in your weakest finger pressing against a spring with your hand stretched wide. Ouch. All this to actually slow you down. (See Bruce Tognazzini, www.asktog.com/TOI/toi06KeyboardVMouse1.html)

So here's some advice I've learned since. Remap your keyboard. The X.org keyboard extension has a large set of options to fine-tune your layout. The desktop environments – *Enlightenment* for me – often put a GUI in front of them. Otherwise, put your choices in a file named `/etc/X11/xorg.conf.d/10-keyboard.conf`. Some people set the Caps Lock to act as Ctrl, but that is still little finger territory and not really good enough. I use `altwin:ctrl_alt_win` to make Alt and Alt-Gr act as Ctrl keys, and the MS Windows key function as Alt. That puts the most-used modifier key under my thumb or strongest



finger. (Minor bonus: Win-Tab now tabs between windows.) Then I use **compose:menu** to turn the MS menu key into Alt-Gr, since I want to use accented characters correctly. It only took a short time to retrain my muscle memory.

Move your hands, both from keyboard to mouse and around the keyboard. Only copy typists are judged by words per minute – move off the home keys, and use a stronger finger for the extreme end keys.

Repeatedly double clicking the mouse will also hurt eventually. Many file manglers have a setting to use a single click to open and hover to select, which is much gentler. Sadly the *GTK* file dialogs don't have the option, despite repeated requests for it in their bug tracker. (Ever hear about the origin of the double click? It was a workaround for The Steve's decree that Macs must have just one

mouse button.) Finally, strengthen your forearms. I know some programmers who actually lift weights, and I use grip squeezers on days when my arms aren't too sore.

I hope you can publish this, and perhaps keep some of your readers away from the anti-inflammatory pills. Those are really hard on your stomach lining.

Dylan

Mike says: Damn right. Remapping your keyboard so the most often used modifier key is under your thumb rather than your little finger is the number 1 best thing you can do for your long-term finger happiness. I've had terrible RSI in the past, which is largely why I'm such a fan of tiling window managers now – it saves you from having to cripple yourself using a mouse. Thanks for the tips – everyone, heed Dylan's words!

Kate won the Group Test of text editors in LV002 (now online at www.linuxvoice.com/text-editors), but like all other apps that take text input, it can be injurious to your health.

THE VIRUS QUESTION

I can accept that there are no computer viruses on Linux, but I'm baffled as to why. The internet is full of theories, but the most persuasive seems to be that there are so many fewer Linux machines than there are Windows, that it's too much effort for the virus writer to even bother targetting Linux. But I've also read that Linux is just inherently safer. So which is it? **Davey McGregor, Perth**

Ben says: People who write malware probably want to do so to get the widest exposure for their little scripts and botnets, so it makes much better sense to target Windows. It still runs on the majority of desktops and we'd guess the average user is less able to defend themselves against malware than the average Linux user. Combine this with some huge (and undisclosed) security holes, and cracking Linux is just not worth the comparative effort.



Antivirus vendors do sell software to stop Linux servers spreading viruses to Windows boxes, but for desktop users this unnecessary.

LUGS ON TOUR

CHAR(14) & PGDay

Josette Garcia gets some serious database action in Buckinghamshire.

CHAR(14) is a 'must attend' international conference for anyone interested in Clustering, High Availability and Replication, plus all forms of parallel, distributed and grid architectures. The joint conferences were organised by the UK PostgreSQL User Group.

Some of the talks included "Replication in Security" by Magnus Hagander – Magnus is a member of the PostgreSQL Core Team and a developer and code committer in the PostgreSQL Global Development Group. He currently serves on the Core Team and as President of the Board for PostgreSQL Europe. I managed to follow the history of replication up to replication streaming. Unfortunately the rest was well above my head (see: www.2ndquadrant.com/bdr)

There next followed "Logical Decoding for Auditing and Replication" by Gianni Ciolli, a principal consultant for 2ndQuadrant Italia. He has been working with Free and Open Source Software for more than 15 years. He was co-founder and then president of the Prato Linux User Group

"Speakers came from Switzerland, USA, Italy, Germany, Czech Republic and Sweden."

(Plug); he has organised many editions of the Italian PostgreSQL day, and in 2013 was elected to the board of ITPUG, the Italian PostgreSQL Users Group.

Bi-Directional Replication (BDR) by Andres Freund. What is BDR? It is the latest Asynchronous Multi-Master Replication for PostgreSQL.

BDR is an Open Source project that follows the same license as PostgreSQL (TPL), which is an early form of BSD open source licence.

Simon Riggs talked about the Future of Replication. His views on the future of Bi-Directional Replication and its inclusion into the PostgreSQL core code.

Char(14) was followed by PGDay – some attendees left, some arrived but I can say that most people stayed for both days.

PGDay

Mike Fowler – Migrating Rant & Rave to PostgreSQL – engaging with your customer in every possible way. Mike narrated the pain of migrating from MySQL to PostgreSQL. He found that all the pains were well worth it!

Magnus Hagander presented the new features of PostgreSQL 9.4, which is to be made available in August or September.

Business intelligence with Window Functions by Gianni Ciolli – please note the absence of "s" after Window. Not only has Gianni got the most beautiful Italian accent but he is also extremely funny.

Simon Riggs gave an update on the AXLE project – Advanced Analytics for Extremely Large European Databases. The AXLE project is an EU funded project to greatly improve the speed and quality of decision making on real-world data sets. AXLE aims to make these improvements generally



Horwood House venue, near Milton Keynes, home of Bletchley Park – now the National Museum of Computing.

available through high quality open source implementations via the PostgreSQL and Orange products (<http://orange.biolab.si/>). Do visit the AXLE website for more details on <http://axleproject.eu/>.

I must admit that the content of this conference was well above my understanding. I should have been bored but the attendees and the organisers were so kind that I never felt out of place or not wanted.

The speakers came from Italy, Switzerland, the USA, Germany, Czech Republic and Sweden bringing information, tricks and latest developments. I believe PostgreSQL is one of the most used databases, so why are there so few attendees at the yearly conference? It doesn't make sense. If you are a PostgreSQL user, I really hope you'll come to Char(15) and PGday 2015.

TELL US ABOUT YOUR LUG!

We want to know more about your LUG or hackspace, so please write to us at lugs@linuxvoice.com and we might send one of our roving reporters to your next LUG meeting

Deer Shed Festival Science Tent

Andrew Gregory, worried about the future, is pleased to see that the kids are all right.

Music festivals on television always look so grubby. Yes, it's good to get several bands you like in one place, but they also attract hippies, henna, batik and tie-dye. And bongo drums. Bloody bongo drums. We were delighted then to learn of the existence of the Deer Shed Festival, now in its fifth year of glorious sunshine near Thirsk in the North Riding of Yorkshire.

The festival essentials were all there (beer, Johnny Marr, burgers), but what piqued our interest was the festival's science tent. Deer Shed is a family-friendly event, and rather than leave the kids to pointless displays of tribal drumming (Which tribe? It's never specified!), the organisers had filled a tent with demonstrations of science – Python, Arduino, astronomy and more.

Connor and Paul from Pimoroni had made the trip to show kids how to build little shrimp kits – tiny, functioning circuits made out of ordinary components. Next door to them, a crowd of six-year-olds were drawing circuits on cardboard, plugging in LEDs and practically jumping up and down with excitement when the circuits

dried and they what saw they'd made. This is a world away from any electronics lessons we had at school. Forget the theory, just show them the blinkenlights and they're hooked.

Over on the other side of the tent, Dan, a Raspberry Pi certified educator, was giving Python programming lessons for – you guessed it – *Minecraft*. This was so popular that slots for the weekend were fully booked within 15 minutes of the science tent opening, and we can see why.

The children playing with *Minecraft* looked to be around 10 years old, and Dan had them coding different block types. A few lines of Python saved into a configuration file and players were wandering the *Minecraft* world leaving a trail of flowers behind them (perhaps the hippy nonsense had pervaded the science tent after all), or flying through the sky leaving a spiral of watermelons. After that, you're only a variable away from being able to change block types on the fly from within the game.

More science

The universities of Teeside and York brought along a mobile



Deer Shed Festival is held in the grounds of Baldersby Park, built for communications technology pioneer George Hudson. deershedfestival.com

planetarium each; again, these were wildly popular. And the Teeside University representatives excelled themselves with CST: Teeside, a mocked-up crime scene complete with evidence to solve the case.

For parents of small children who want to let their hair down with kids in tow, Deer Shed is great. For the atmosphere, the brilliant local beer (no flat Carling here, thanks very much), the performances and everything else, it's well worth it. If you're at a loose end in late July next year, come along!



Far left: The science tent was packed for the whole sun-drenched weekend.

Left: Sebastian Jacques helps Paul from Pimoroni out with a simple circuit.



Tickets to OSCON are expensive, but one of the free hall or exposition passes will also get you into many of the sponsored sessions.



4,400 attendees create a buzzing hive mind of ideas, connections and possibilities.

OSCON 2014

Graham Morrison travels almost 5,000 miles from the Shire to the biggest open source conference on the planet.

We've been going to O'Reilly's Open Source Convention – better known as OSCON, for eight years. Not just because of the incredible number of fine India Pale Ales on offer, or the highest concentration of microbreweries this side of the Orion Spur, but because Portland is a friendly, warm and creative city host to an incomparable variety of bars, beards and body art; the perfect backdrop to a week of geek communion.

And each successive year here helps OSCON imbibe more of Portland's spirit. This year's event was noticeably less corporate and less ostentatiously

sponsored, for example. Of course, there's still the major backers – bluehost.com, PayPal, Citrix, Google and HP plus a plethora of smaller companies. But many of the extracurricular functions wouldn't be possible without their contributions. Monday night's Attendee Party, for instance, with its surreal combination of oxygen bar, quad-copters, bungee-trampolines and glittery cupcakes was funded by three different sponsors. However, this year's event felt hackier, more makery and more open source-friendly than previous events – truer to its roots as a Perl conference, and we enjoyed the difference. All of

which gave the many, many corridor meetings, the Birds of a Feather get-to-togethers, the after-session parties and the sessions themselves an atmosphere not unlike a University campus.

This year's event started unusually on a Sunday (and singularly; Monday–Friday service resumes in 2015), and one of the most inspirational parts of this year's conference was that you saw children in the halls and corridors, and sometimes, in the sessions themselves. This was because on the first day, Sunday, there was the beta version of an experimental track that O'Reilly called Kids Day. On this track, 70 kids of all ages – and we spoke to attendees who could have been anywhere between 6 to 16 – got their hands dirty learning Python, modding a Java game with a touchscreen Raspberry Pi or hacking *Minecraft*.

Open Source satellites

This refreshing approach continued through to Tuesday's keynotes, where you had to pity the sponsors that followed enthusiastic teenage coder Shadaj Laddad, after he delivered a talk full of wonder and the freshly squeezed potential he's found through programming. We also loved Wendy Chisholm's coming out session for introverts, where she, along with what appeared to be 95% of the other attendees, admitted that they're not super-confident supreme beings after all, and keeping up appearances can be exhausting. But it was Will Marshall's final talk on the briefcase-sized imaging satellites his company is launching that got us emailing Planet Labs to ask for an interview. They're throwing hundreds of these units into orbit and creating a system that updates a complete image model of the Earth every 72 hours, potentially down to individual fields, houses and trees. More importantly, they're going to create an open API to deliver universal access to this data, and to allow anyone to perform their own analysis. Each satellite is also powered by a tiny x86 computer running Ubuntu.

On Wednesday, Simon Wardley's keynote was preceded by an impromptu slide informing the audience of the UK Government's intention to switch internal documentation to ODF and PDF/A, which was greeted by an enthusiastic cheer. In a doubly surprising UK reference, Tim O'Reilly later talked about technological and cultural revolutions, and in a part where he's referencing the difficulty of recognising

Shadaj Laddad gave an impassioned talk about how programming is shaping his future.




Simon Wardley started his talk by giving a huge shoutout to the UK Government for committing to open document formats.

how best to serve your users, he mentioned **gov.uk's** design principles (<https://www.gov.uk/design-principles>) document, and how this list is a "fabulous idea about how you start with [your users] needs."

Tim Bray (co-creator of the XML specification and lots of other good things) is currently concerned about privacy, and his talk was mostly about pleading with developers to add proper encryption.

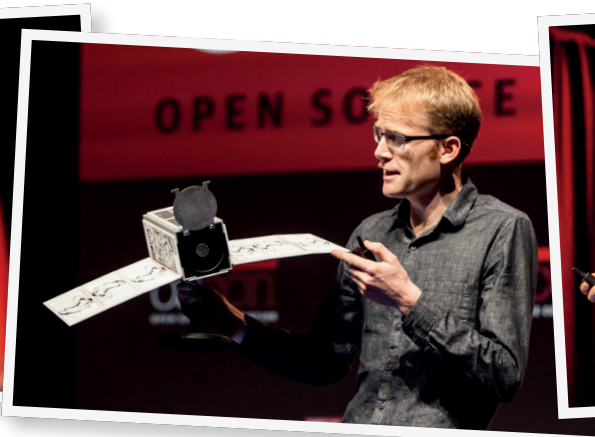
"We don't want to talk about this stuff because it involves two really horrible things:" he said, "really hard math and... politics." He went on to make what we think is a vital point, "But I'm going to argue that you should get interested and here's why: first of all, you can ignore the math these days. There's good libraries for that. And as for the politics, politics and policy are reality, and if you want to blow that stuff off then you've just lost the right to complain."

OSCON is always going to be a difficult proposition for Europeans and people a long way from the North West coast of the United States. But if you're lucky enough to work for a company with money to spend on training, we can think of no better way of doing so. For anyone else who can afford it, it's without doubt worth the trip. 

Below left Wendy Chisholm admitted she was an introvert. Along with almost everyone else.

Centre Will Marshall is holding one of his satellites. It's tiny, based on an x86 PC and runs Ubuntu!

Below right Tim O'Reilly shares his usual insight and wisdom into the way things are going.





BEST DISTRO

2014

Is your current Linux distribution really the best in town, or are you missing something even better? Graham, Ben and Mike put a bunch to the test.

We're going to get a lot of flak for writing these words, but we're not scared – Linux Voice drops ice cubes down the vest of fear. So here we go: you might be using the wrong Linux distribution. Or to put it more diplomatically, you might not be running the distro that's best suited to you. "What a load of codswallop!", you respond.

"My distro does exactly what I need it to do. I've been using it for years and I'm happy with it."

That's great, but could it still do a lot

more? Have you really tried all of the big-name distros in depth? Could there be another distro out there that's better than yours in a key area such as security, performance or documentation? Is your distro really the best when you're trying to convert newbies to Linux? It's good to settle on a single distro and learn its ins-and-outs, but given the rapid

pace of development in Linux, it's always worth keeping your eyes open for something better.

With all these things in mind, we decided to look at the current state of play in the Linux distro world. We wanted to see which distros excel in certain important areas, to find out who's leading the charge here in mid-late 2014.

"Given the pace of development in Linux, it's always worth keeping an eye out for something better."

In tests like these, it's often possible to bundle certain distros together as they're so closely related. In the Packages section, for instance, we look at

Ubuntu and Mint together because they share the same repositories. In any case, we want to give you all the information you need to make an informed choice about the best distro for you. So if the one you're currently using comes up tops in the categories important to you – congratulations! And if not, fire up *VirtualBox* and start exploring...



Best for beginners

The ideal gateway into Linux for new users.



For beginners, two things are important. One is whether you can work out how to do something by yourself. The second is how easy is it to find a solution if you hit a problem.

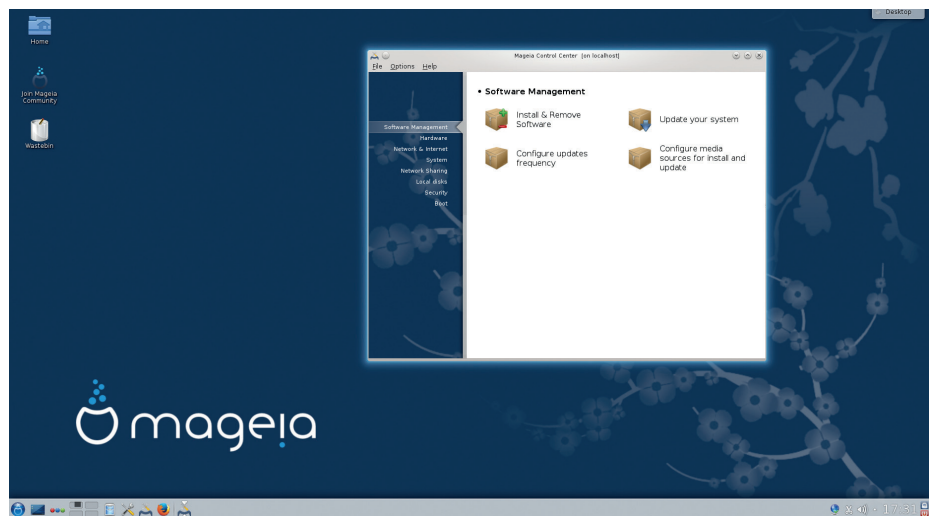
For a long time, the standard distro for any beginner was Ubuntu. However, since the introduction of the Unity interface, it has become less popular. The non-traditional layout of the desktop could lead to beginners feeling unfamiliar, and the Launcher and scopes can take a little getting used to. People coming from Windows may also get confused by the way the window menu bar blends into the top menu bar.

The others distros we've looked at are all based on a traditional desktop, and the layout should be familiar to anyone who's used a computer at any time in the last 20 years. They have a task bar along the bottom and an applications menu in the lower right-hand corner.

Mint is the most popular of these. Its two main flavours (Mate and Cinnamon) are sufficiently similar that we'll consider them together. The last of the contenders in this category is Mageia.

Simple interfaces

Overall, we feel the KDE environment of Mageia is a bit too cluttered to be ideal for beginners, though it does have an important place. Both of the main Mint desktop environments (Cinnamon and Mate), are clean with unnecessary detail tucked away. It also looks really nice, which helps give a



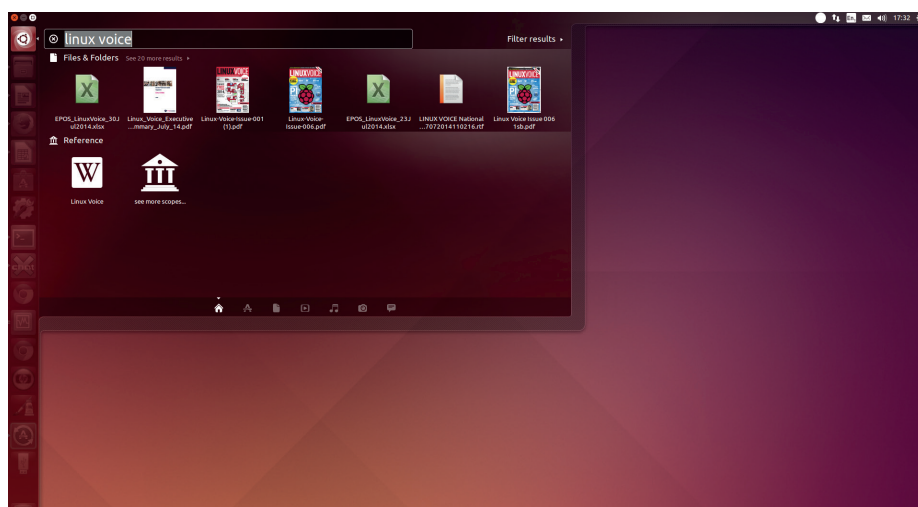
The KDE desktop can be confusing at first, but Mageia's implementation is relatively intuitive.

good first impression – no one wants their new operating system to look worse than their old one.

The biggest difference between Ubuntu and the others from a beginner's point of view isn't the interface, but the huge amount of help online in the form of tutorials, forum posts, and solutions to problems. If you get stuck on Ubuntu, you're far more likely to find a solution online than if you're using another flavour of Linux. Of course, an experienced user will know that if they have a problem on Mint or Zorin (another distro aimed specifically at new Linux users, with an interface designed to look and feel like Windows), they could look for a solution for Ubuntu and it would probably work.

However, we can't really expect a new user to know this.

Ultimately, we think that the amount of help available for Ubuntu outweighs the unfamiliar user interface. However, everyone is different, and any of these distros would make a good choice for beginners. We would recommend Mint (either version) for beginners who had trouble getting used to Unity, and Mint Mate edition for people with lower-powered hardware.



The Ubuntu Launcher does far more than a typical desktop menu. This can take a bit of getting used to, and has drawn criticism from privacy groups for its internet searching.

When beginners aren't beginners

We've based this category on the idea that a beginner is non-technical. This may not be the case. They could be new to Linux but have experience configuring Windows systems, in which case they may be uncomfortable at the command line and editing configuration files, but still have a good idea what's going on. There is a certain logic to saying that the best distro for people like this is Arch Linux. Using this, they'll have no other option than to learn how their new Linux system works.

Another option would be Mageia, because it has the Mageia Control Centre. This enables you to configure much of the system from within one unified graphical application. Rather than having to memorise different commands for each task, you just fire up the Control Centre and make any changes that are needed.



Best looking

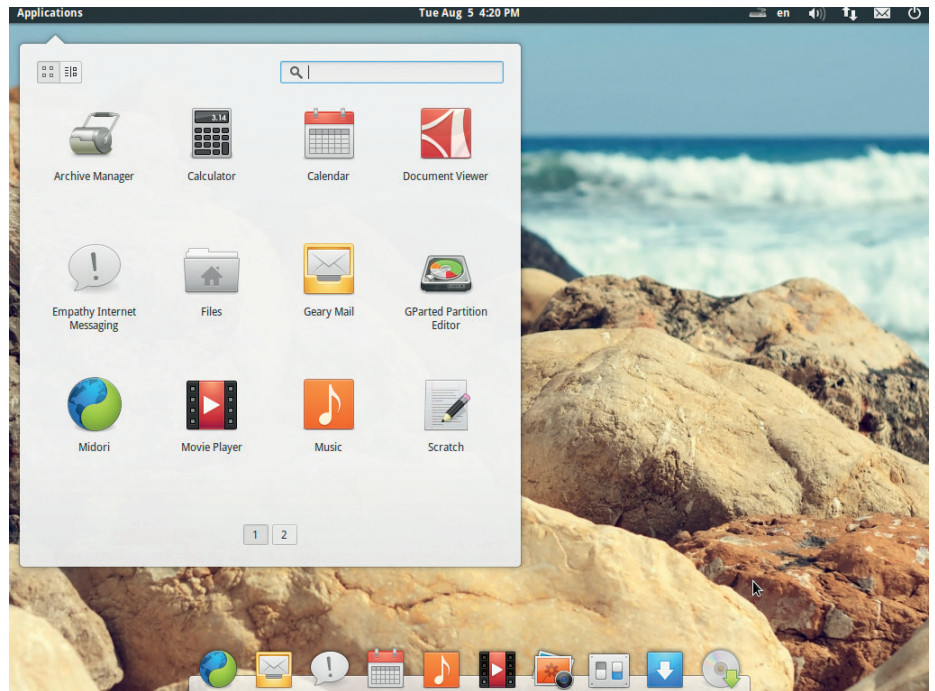
An attractive environment makes everything better.



This category is particularly contentious for two reasons. First, what is beauty, and who gets to define it? Second, since almost any distro can be made to look like almost any other distro, how do we decide which is the best looking? These are both valid questions, but we will crush them both with an authoritarian boot. Firstly, we know beauty when we see it, so we get to define it (if you don't like that, start your own magazine). Secondly, we'll look at each distro naked, straight after installing it.

Bodhi Linux is based on Enlightenment, which bills itself as the original eye-candy desktop environment. Perhaps the most impressive thing about Bodhi (and Enlightenment) is how many graphical treats it can supply with very little strain on the hardware. This makes it a good choice if you're after a slick distro for a low-powered machine. However, some of the graphical niceties feel a bit like they're there to show off, rather than to make using the desktop a more pleasing experience.

Lots of distros come with KDE, but default KDE is a bit lacklustre. OpenSUSE and Mageia do quite a good job of improving it, but they're not in the top tier. Our favourite KDE flavour is Rosa Desktop Fresh. As soon as the desktop loads, you can see it's not standard KDE. At the bottom of the screen,



The clean, simple style of Elementary OS flows through the desktop and all of its included applications to make a beautiful computing environment.

the RocketBar replaces the panel, but looks a lot nicer. Along with the usual icons and widgets, there's a Downloads stackfolder that enables you to see the contents of `~/Downloads` without having to open up the file manager. Simple Welcome takes the place of the KDE menu, and works a little like

a souped-up Gnome Dash. All these enhancements mean it's not the best distro if you prefer to use your own KDE configuration, but for people who want a good-looking distro on first boot, it's great.

Mint Cinnamon does a good job of getting out of the way, while still being pleasing to the eye. It's the least ostentatious of the environments we've looked at here, and this comes from having a clean desktop and a well-themed set of *GTK* widgets.

Pantheon – the desktop environment of Elementary OS – also uses *GTK* to provide a clean and elegant look. Elementary takes this approach further than Cinnamon, and the environment is stripped down to its bare essentials. Every icon feels like it's been placed for a good reason, and every pixel tweaked to fit in perfectly.

All of these distros look good. However, Elementary OS does the best job of carrying its style through the wide range of apps that comprise it, and so we're declaring it the best-looking desktop distro.



It's hard for a static picture to capture the eye candy on Bodhi, as it's all in the movement. But picture the menu icon spinning, the terminal pulsating, and everything fading in and out.

WINNERS

- 1st Elementary
- 2nd Rosa
- 3rd Mint



Best for packages

Which distribution has the most software?



Brace yourself for some controversial statistics. Counting the exact number of packages in a distribution can be tricky, and different distributions package up software in different ways. For instance: imagine you've got a program called FooApp that has support for 10 different languages for its interface. One distro might bundle everything together into a single package – whereas another may give each language its own package. Multiply this over thousands of programs with multi-language support, and it drastically changes the package counts between distros, even if they have the same number of applications.

Similarly, many programs support the use of plugins and extensions; again, these may be placed into the main package in some distros, or split out across dozens of extra packages in others. Quite a few distros make use of “virtual” packages, so installing, for example, the package **xfce4** actually pulls in 20+ other packages. And some distros that provide long-term support include multiple versions of packages for maximum compatibility (eg older versions of SDL, SDL-mixer, SDL-image etc).

So the end result doesn't necessarily reflect the range of software in a distro. Although it has twice the number of packages in its repositories, Debian doesn't simply have twice as many standalone programs as OpenSUSE. But one thing is for sure: if you're looking for a lesser-known or obscure piece of software, you're more likely to find it in the distros with the high package counts. A big chunk of the programs in Arch and Debian are old and haven't been

The screenshot shows the Arch Linux website's AUR search interface. At the top, there's a navigation bar with links for Home, Packages, Forums, Wiki, Bugs, AUR, and Download. Below that, there's a search bar with options for Category (Any), Search by (Name, Description), Keywords, Out of Date (All), Sort by (Name), Sort order (Ascending), and Per page (50). A 'Go' button is present. Below the search bar, it says '50927 packages found. Page 1 of 1019.' A table of results is shown with columns for Category, Name, Version, Votes, Description, and Maintainer. Two packages are visible: 'yaourt' (system category, version 1.5-1, 2984 votes) and 'dropbox' (network category, version 2.10.27-1, 2319 votes).

The Arch User Repository is huge, and new software ends up here more quickly than in other distros.

updated in years, but they're still being rebuilt to work with the latest distro versions.

Abaci at the ready

Now, let's talk about Arch Linux. We separated its package statistics into two parts here: one for the main distro (community, core, extra etc. repositories), and the other for AUR, the Arch User Repository. The latter is enormous and updated at a breakneck pace, but the packages are not in the “official” distribution (although they often end up there after extensive testing). Officially, Arch only had 6,836 packages at the time of writing – not actually that many, but that's what you get if you stick to the main distro.

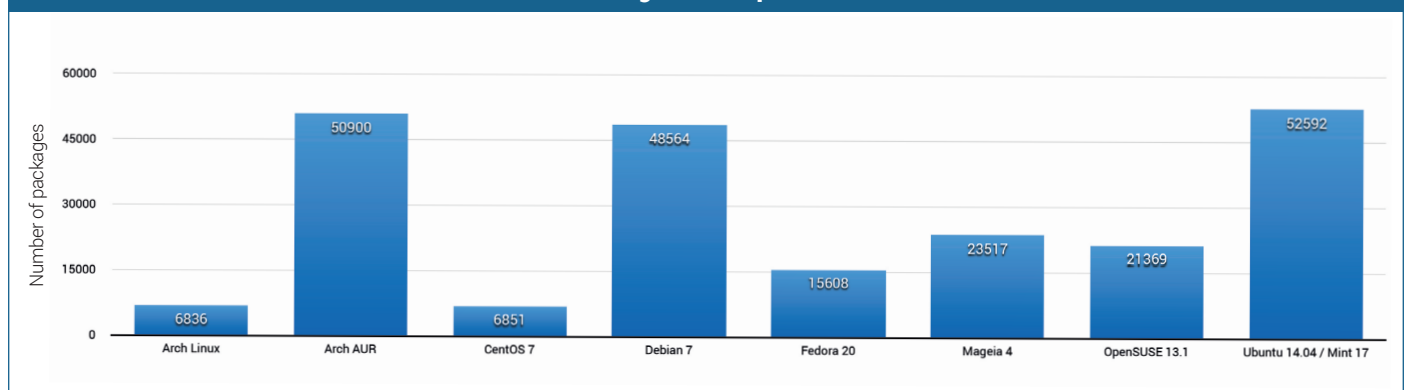
“But hang on”, you say, scratching your head. “I've just been to www.archlinux.org/packages, and it says there are 11,459

packages. What gives?” Well, that's the total for i686 and x86_64 packages – there's a lot of overlap. It's unfair to count the packages for all architectures (otherwise Debian's bar in the chart below would extend beyond the top of the page), so in the case of Arch and other distros, we chose the x86_64 and any/noarch repositories. Basically, the stats below show the number of packages you can install on an x86_64 box.

After all that a caveat: quality does not mean quantity. If you're looking for a server box, packages of synthesizers, games etc aren't going to be much use to you.

A blue badge with a trophy icon and the text 'WINNERS'. Below it, a list of the top three distributions: 1st Arch (with a blue triangle icon), 2nd Ubuntu (with an orange circle icon), and 3rd Debian (with a red circle icon).

Package counts per distro





Best for documentation

When you need help, who you gonna call?



Quality is a lot more important than quantity when it comes to documentation. Over the years we've seen many free software projects that have reams of guides, tutorials and FAQs, but if the content is badly written, unorganised or out of date, it's not much use. The same applies to distros: a short, concise and well-written guide is much more useful than poorly maintained scraps of information scattered around the web.

Debian's official documentation is generally well crafted, but it suffers from a lack of centralisation. Go to www.debian.org/doc and you'll see that there are plenty of resources, but it's not clear where to start if you're seeking help about a specific problem. Should you look at the FAQ? Or Debian Reference? Maybe the wiki has the answer... It gets a bit messy, but we have to give a mention to the separate Debian Administrator's Handbook (<http://debian-handbook.info>). This is exactly what we're looking for as end users and admins: everything you might need, in one place.

Ubuntu's docs (<https://help.ubuntu.com>) are mainly focused on desktop end-users, with well categorised mini-guides to common tasks. The Server Guide has more advanced user material – but it's not

“Back in the days of dialup internet connections, SUSE Linux was our favourite for documentation.”

exhaustive. Plenty of other tips are scattered around the wiki at <https://help.ubuntu.com/community>, and there's also

www.askubuntu.com, which is a good way for getting quick-fire responses to questions.

Many guides for Debian and Ubuntu apply to Mint, but the latter also has its own PDF installation guides in various languages:

www.linuxmint.com/documentation.php.

Some of the versions are very out-dated, however, missing the latest Mint releases.

Mageia, meanwhile, doesn't really impress with its limited range of guides at www.mageia.org/en/doc; there's some information on the installer and control panel, presented in an unwelcoming fashion, but not much else on the wiki.

Back in the days of dial-up modem connections, SUSE Linux was our absolute

Preparation

Note: If you wish to install from an existing GNU/Linux distribution, please see [Install from Existing Linux](#). This can be useful particularly if you plan to install Arch via [VNC](#) or [SSH](#) remotely. Users seeking to perform the Arch Linux installation remotely via an [SSH](#) connection should read [Install from SSH](#) for additional tips.

System requirements

Arch Linux should run on any [i686](#) compatible machine with a minimum of 64 MB RAM. A basic installation with all packages from the [base](#) group should take less than 800 MB of disk space. If you are working with limited space, this can be trimmed down considerably, but you will have to know what you are doing.

Prepare the latest installation medium

The latest release of the installation media can be obtained from the [Download](#) page. Note that the single ISO image supports both 32 and 64-bit architectures. It is highly recommended to always use the latest ISO image.

Tip: The [archboot](#) ISO images can take several steps explained in this guide [interactively](#). See [Archboot](#) for details.

- Install images are signed and it is highly recommended to verify their signature before use. Download the `.sig` file from the download page (or one of the mirrors listed there) to the same directory as the `.iso` file. On Arch Linux, use `pacman-key -v iso-file.sig` as root; in other environments make use, still as root, of `gpg2` directly with `gpg2 --verify iso-file.sig`. The file integrity checksums md5 and sha1 are also provided

Arch's guides may look excessively long at first glance, because they include absolutely everything.

favourite for documentation. You'd order a boxed set over the phone, and a few days later a hefty lump of Linux goodness would arrive at your door, containing three chunky manuals. It was bliss. Today, OpenSUSE still has an excellent set of documentation at <http://doc.opensuse.org>: the Startup guide (for regular end users), Reference (for administrators) and extra guides for security and virtualisation. There's some overlap and we'd like to see them combined more

scattered around and would be better organised into a single reference document. Of course, CentOS users can read the official Red Hat documentation at <http://tinyurl.com/rhdocs>, which is very thorough, straightforward, and polished. You can see the results of Red Hat paying people to work full-time on documentation.

Super Arch

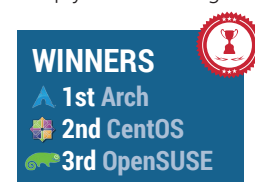
Finally we come to Arch Linux, and we've saved the best until last here. Arch's documentation is almost entirely provided on the distro's wiki at <https://wiki.archlinux.org>,

which has some of the most in-depth and detailed guides we've seen of any software project. The Beginner's Guide is especially good, if a bit long-winded (but then, Arch is targeted at experienced Linux users). Then there's the General Recommendations page, which is a superb one-stop-shop for all things administration: user management, packages, power management and so forth.

But what makes Arch our winner is this: for the large part, its information applies to other distros. In discussions on the web, we've seen users of Ubuntu, Debian, Fedora and other distros paste links to the Arch wiki, simply because its guides are so good.

effectively, but the information contained therein is clear and well presented.

Then we have Fedora and CentOS. The former, at <http://docs.fedoraproject.org>, is in a sorry state: you're told to select a language and then Fedora version, and read the docs from there. Our test case was to find a guide to adding new user accounts – and for Fedora 20, it wasn't there. Nothing. When we opened up the documentation list for Fedora 18, however, we saw the System Administrator's Guide, which had the information we needed. So lots is either outdated or badly sorted – it's hard to navigate and needs to be cleaned up. CentOS doesn't fare much better. The manuals at www.centos.org/docs don't cover the last two major releases, while the wiki has some useful guides, but they're





Best for security

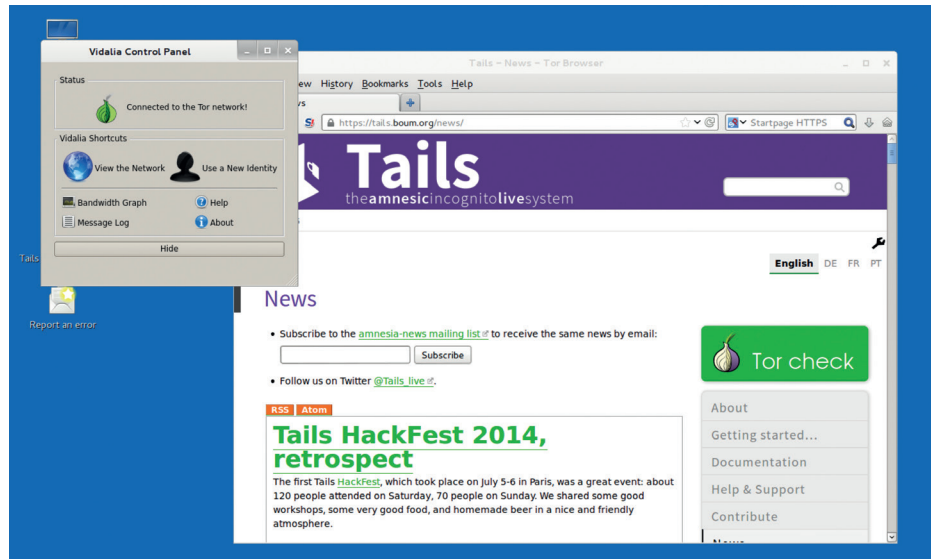
The price of security is eternal vigilance.



There are lots of different aspects to security – enough for eight pages on its own. Your first step is to understand your own requirements. If your first priority is the security of your own data, for example, you would require a distribution that's happy to encrypt your home folder or root partition and handle the complexity that that involves. You may also want to extend that requirement to easy integration of *GnuPG* into the default email client, or even making sure *Firefox* is pre-configured to always use HTTPS. But most importantly, security needs to be easy, because if you don't understand what you're doing, a bad configuration is worse than no configuration at all, because it gives you a false sense of security. This is the problem with Arch. It can be the quickest distribution to patch a vulnerability, and it makes an excellent server, but you need to know what you're doing, because a mistake could be costly.

We have to give credit to Ubuntu here. It took the relatively brave step of moving its full-disk encryption option from behind the advanced settings in its installer to the forefront of the installation processes, giving many more users the opportunity to encrypt their data. For a distribution as user-friendly and as popular as Ubuntu, this was a brave move. Even the EFF was impressed.

Ubuntu also made a lot of noise when its shopping scope searches from the dash sent unencrypted data through its own servers to Amazon. Many of us had strong



Booting Tails from a USB stick will keep your connections anonymous through the *Tor* network.

feelings about this, especially as there was no way of turning it off. But these problems have been mostly addressed, and while it's still turned on by default, there's a simple way of turning the shopping scope off. If you wanted to be certain, for course, we'd recommend using an Ubuntu derivative, but Ubuntu is still a good choice for easy, comprehensive encryption.

Reactive security

The other principal concern is online security. This always used to mean the pre-configuration of a firewall blocking external access to services running on your

machine. This can still be important – you may only want a web server accessible on your LAN rather than across the internet, for instance. But it's more important to worry about the services and applications you run. This is where most problems occur, and the recent Heartbleed bug in OpenSSL highlights this issue perfectly. It's used by so many applications and services that many became vulnerable as soon as this bug was found, and consequently, the best distribution for security became the quickest distribution to patch the vulnerability.

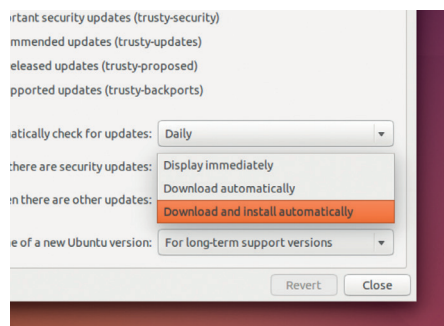
But it's not just speed of deployment that's important, it's the quality of any patches as well as the testing that goes into the original distribution. And for that reason we'd recommend a distribution with a proven track record of defending itself online. CentOS, for example, with its Red Hat provenance is rock solid, although it still requires some know-how.

However, if security and privacy are of the utmost importance, nothing can touch Tails, a distribution designed for anonymity and secure communication, so we're putting that top of our list, followed by more pragmatic solutions that can be used more as day-to-day installations.

Turn on automatic updates

When a vulnerability is detected and the problems with a package are corrected, you need to install the update to patch the problem on your distribution. You can do this manually by triggering the update procedure, but that means you also have to be proactive by keeping an eye on any security issues. This is what many Arch users do as a matter of course. But for more general use, you're better off turning on automatic updates, and if possible, limiting those updates to security patches of a certain severity.

With so many distributions being derived from the latest Ubuntu release, they can all take advantage of Canonical's updates. Just open the 'Software & Updates' control panel, switch to the updates page and use the drop-down menu to select an option that works best for you. We'd recommend the 'Download and install



Don't worry about security updates. We'd suggest installing them automatically.

'automatically' option, because then you can simply forget about it – it takes any hassle out of staying up to date.

WINNERS

- 1st Tails
- 2nd Ubuntu
- 3rd CentOS





Best for performance

Forget upgrading your hardware – get a new distro!



Back in the olden days, when every megahertz was sacred and PCs were beige, the performance of your distribution was important. It would make the difference between a system being snappy and usable, or a system being re-installed or consigned to the bin.

Nowadays it's the case that in some ways, especially for desktop use, performance has plateaued. Multi-core CPU cycles, storage and memory are cheap, and most of us barely touch their limits. Your choice of distro normally has much more to do with package provision and the default desktop environment than whether it's making best use of your hardware. And because that hardware is always so different from one user to the next, it's almost impossible to provide a comparative metric that's going to have any meaning.

Therefore, if you care about performance it's because you need to get the best out of limited hardware, and we can, sort of, test for that. In a completely unscientific way, we installed six different distros alongside Windows 8 on the same PC and onto the same (large) hard drive. This is a real working computer (3.3GHz Core i5 with

Performance benchmarks						
	Scientific 17 (beta)	Mint 17	Slacko Puppy 5.7.0	Arch KDE	Lubuntu	Ubuntu 14.04
Firefox Sun Spider (ms)	158.4	150.2	162.7	170.5	150.5	151.7
Boot to login	42	37	21	18.2	37	40
BASH tar x	10	8.5	12	12.5	7.6	9.2
BASH tar zcvf	19	18.2	18.6	20.9	16.5	16.5
GUI tar x	18	16.7	307	19	15	15
GUI tar zcvf	31.2	25.9	281	30.1	21.6	21.6
GUI rm -rf	11	9	22	17	17.1	17.1

Times are in seconds unless otherwise stated

If you want the best performance from your system, use a low-power window manager and the CLI.

16GB RAM) with dozens of devices connected, so it was a good real-world test. It also meant that the test was unfair on some distributions, as they made a much better job of parsing the many USB devices than others while taking longer to load for their trouble. This is why Arch does well at boot time – we haven't installed anything to make it do otherwise.

Take it with a pinch of salt

All of which is a long way of saying benchmarking and tests say very little about the performance you can expect on your own hardware, but there are three lessons

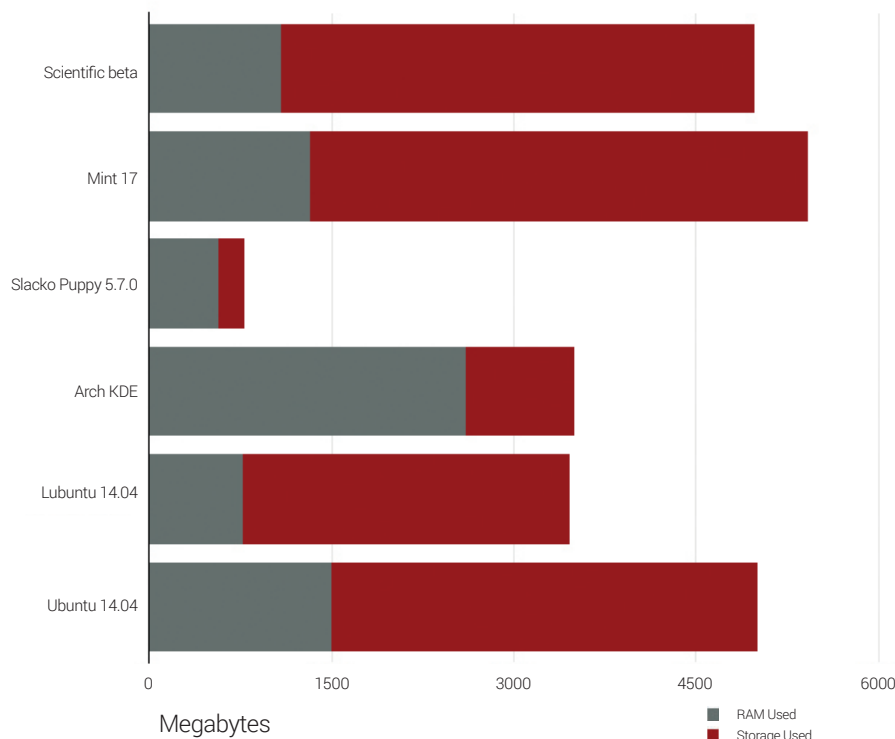
we've learnt from these tests:

- 1 Firefox runs almost identically, regardless of your distribution or desktop. If all you do is browse, don't worry about it.
- 2 GUI tools for file management can have an effect on file operations, especially if you're installing third-party applications. Use the command line if you can. If not, take the time to configure a cut-down window manager or desktop.
- 3 If you're looking for a distribution for low-powered hardware, use a low-powered distribution. Slacko Puppy 5.7.0 easily won nearly all the performance tests, only failing on the GUI compression. That's mostly because its creators wouldn't imagine the typical user not using the command line.

That makes Slacko Puppy our choice of distro if you need something to run on limited hardware. It's also pretty addictive running it on fast hardware, as you suddenly realise the reason why the window isn't moving immediately after you click it is because your desktop is drawing shadows and wobbly windows. Everything else suddenly feels sluggish.

But we also have to say that Lubuntu, the LXDE-based derivative, did remarkably well, which makes it our recommendation if you're looking for modern fittings on a frugal desktop, and one that still looks fantastic. Our third place goes to Arch simply because it's the easiest way to build your own minimal distribution for your own hardware, only installing exactly what you need.

Resource usage



In a comparison of the amount of RAM and storage used after installation. Slacko wins easily.

WINNERS

1st Puppy

2nd Lubuntu

3rd Arch

BEST DISTRO 2014

Just as the sports team with the best stats doesn't simply win the game, the distro with the best scores in six areas doesn't simply get awarded the best distro status. To come up with an ultimate winner, we stared deep into each distro, and drew on our personal experience. We looked into every option, and meditated on the concept of distro nirvana.

We were looking for a distro that performs well in every area, and excellently in many, making it a good all-round distro. However this alone isn't enough. It needs to have something that pushes it ahead of the competition – and the competition is getting better every year. It needs that certain X factor to make it stand out. It should be a distro people want to install; a distro that people get passionate about; a distro that makes you remember why you love Linux.

Arch Linux does all this and more. The two things that make it stand out aren't fancy bits of software, or slick user interfaces, but its philosophy and its community. Arch is built around the simple principle that the user should control the system. Instead of fancy graphical tools to autoconfigure everything you need, it provides you with just the bare essentials you need to build your own system.

Just as a mountain climber becomes one with the raw mountain in order to climb it without technical assistance, and a surfer needs just a carved plank to harness the power of a wave, so a computer user needs just the basic tools that Arch Linux provides to get the most out of their system.

The community keep the documentation up to date, and build the Arch User Repository – one of the largest collections of software in the world.

All this doesn't mean that we think everyone should stop here while they go and install Arch on every computer they have. While we think it's the best Linux distro currently available, it's not perfect for every situation. For example, Tails is still the best distro for online anonymity, and the cutting-edge nature of Arch means that only the bravest sysadmins will use it on public-facing servers.

There are hundreds of Linux distros for a reason, and that reason is the hundreds of



Arch: Linux that doesn't disguise itself

different uses people have for Linux. It's an endlessly flexible system, so there will never be just one form that is perfect for everyone.

That said, we think that Linux users should try Arch at least once. Even if you

don't fall in love with the distro, you'll learn a lot about how Linux works, and get a better understanding of why other distros do the things they do. It's not just for super-geeks – it's a distro for the masses. 🐧

OUR CHAMPIONS

Best for beginners: **UBUNTU**

Best looking: **ELEMENTARY**

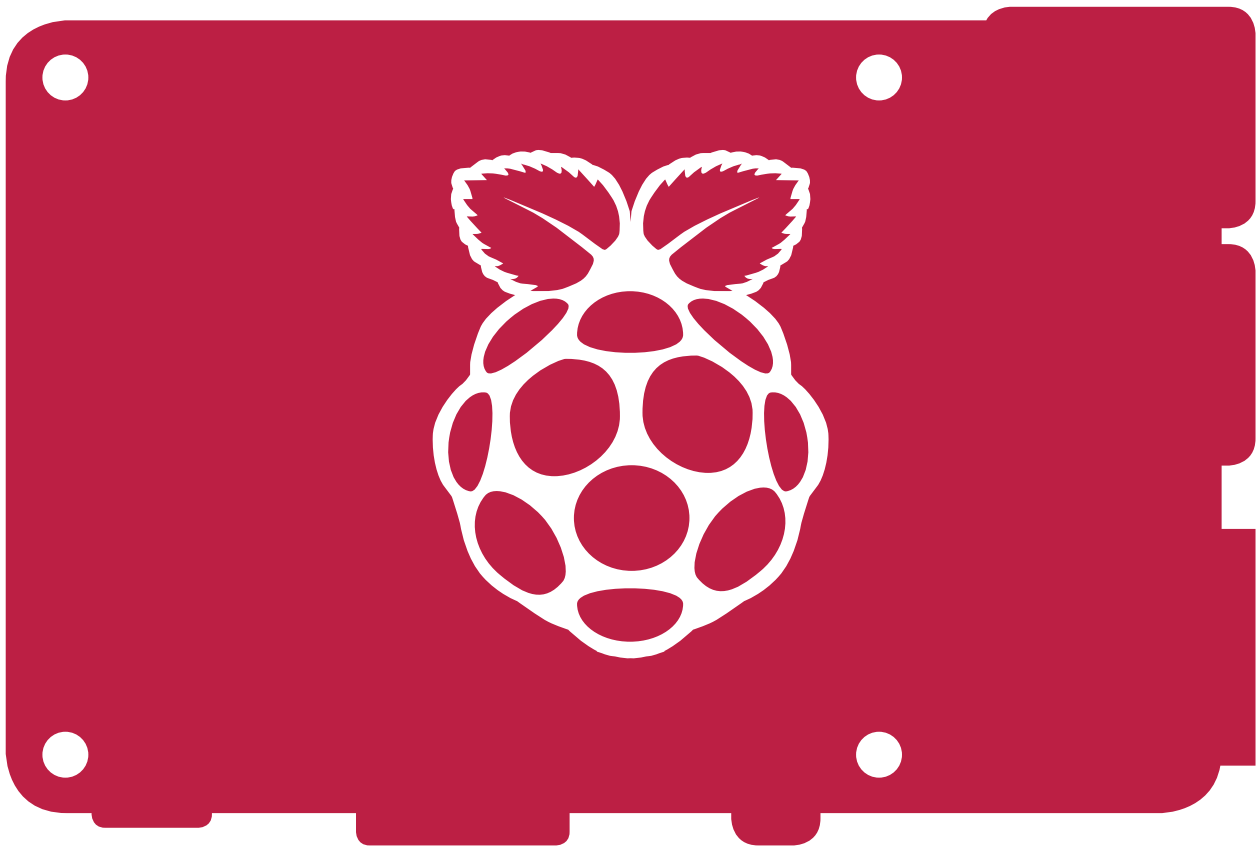
Best for packages: **ARCH**

Best for documentation: **ARCH**

Best for security: **TAILS**

Best for performance: **PUPPY**





RASPBERRY PI

B+

Two computers, one SoC. Ben Everard takes a look at the new Raspberry Pi that ships with the old processor.

On Monday 14 July, The Raspberry Pi Foundation officially announced the biggest change to the design of the Raspberry Pi model B since its launch at the start of 2012: the model B+. In a surprise to some, the new Pi doesn't have a new processor or any more memory than the previous revision. However, many things around the System on a Chip (SoC) have changed.

The new version is almost exactly the same size as before, but again there have been a number of important tweaks to the physical design. The corners are now rounded; the connectors only lead off two

sides; the USB ports no longer stick out; and the mounting holes are now in a rectangle. These mounting holes are also used to support expansion boards, making a far more secure attachment than

relying on GPIOs alone (as most did previously).

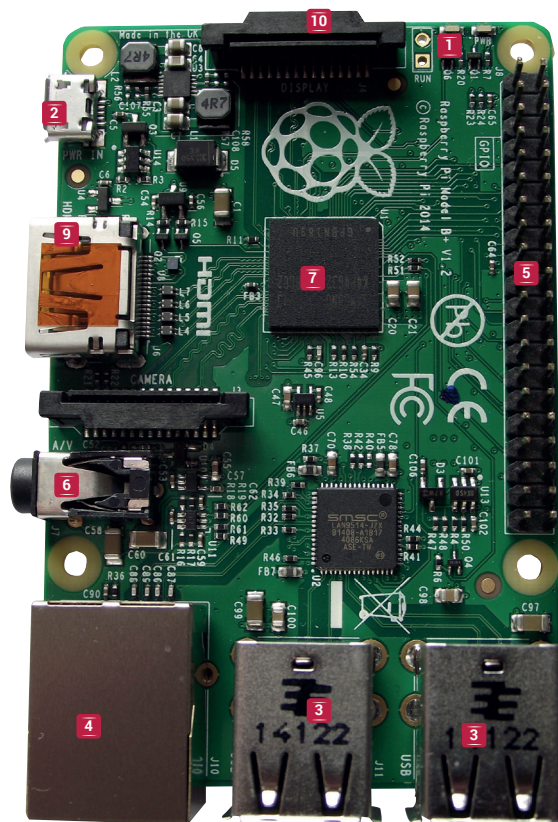
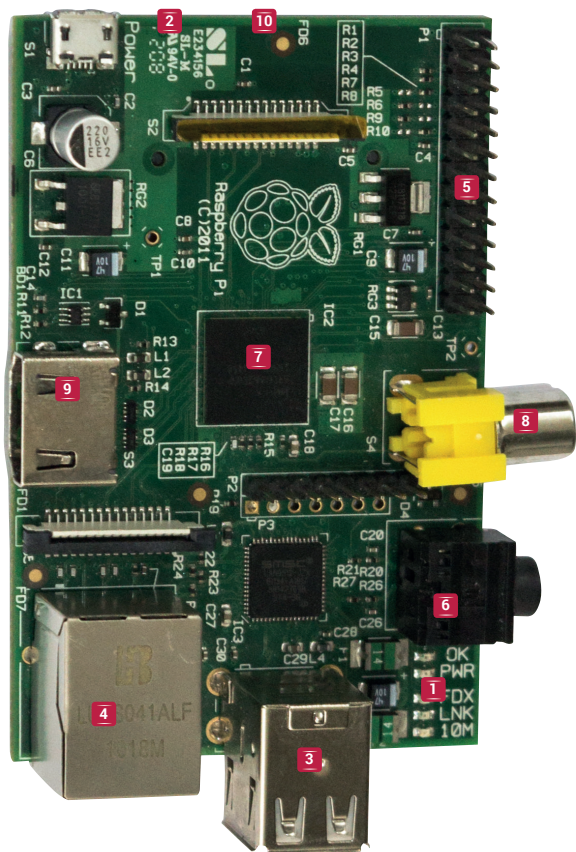
These are all cosmetic changes, but together they do make the device nice to use. Perhaps the

biggest beneficiary of these changes aren't normal users, but the people making boxes and enclosures for the Pi.

If you're used to thinking about computer performance in terms of the usual metrics of

“The differences between the B and B+ are important, and add up to a much better computer.”

Side by side The changes mapped out



- 1 Status LEDs** The Power and ACT LEDs have moved to a new position on the board and the network LEDs on the B+ are on the network port.
- 2 Power Supply** The B+ is still powered by the same micro USB power supply; however it uses the power more efficiently, so it's less prone to power-related problems than the model B.
- 3 USBs** Thanks to the improved power supply, the USBs on the B+ are capable of running higher-powered peripherals, though a powered USB hub is still recommended if you're using anything with a significant power draw.

- 4 Ethernet** The B+ still connects Ethernet through the USB hub, so speeds can be affected by heavy USB usage. The network status LEDs are now on the Ethernet port.
- 5 GPIO Pad 1** This has been expanded from 26 to 40 pins. The top 26 pins are identical on both the B and B+, so and boards that connect to them should work on both. However, some boards designed for the B (such as the PiFace) fit closely around the raised components may not physically fit onto the B+ unless you add something to raise the GPIOs.
- 6 Audio** The audio output on the model B isn't very

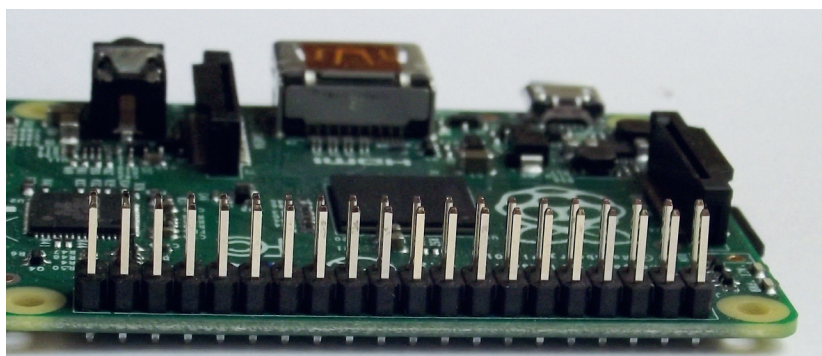
- good. The B+ improves this dramatically, and is good enough for most applications.
- 7 SoC and Memory** These are exactly the same on the B and B+, so performance should be identical.
- 8 Composite Video** On the B this had its own connector, but on the B+ it's on the fourth pole of the audio jack. This means you'll need the appropriate cable to connect it to a TV.
- 9 HDMI** HDMI video and audio is unchanged.
- 10 SD Card** The B+ uses a micro SD card rather than the full sized SD card. A micro SD card will work in a model B as long as it's inside an adaptor.

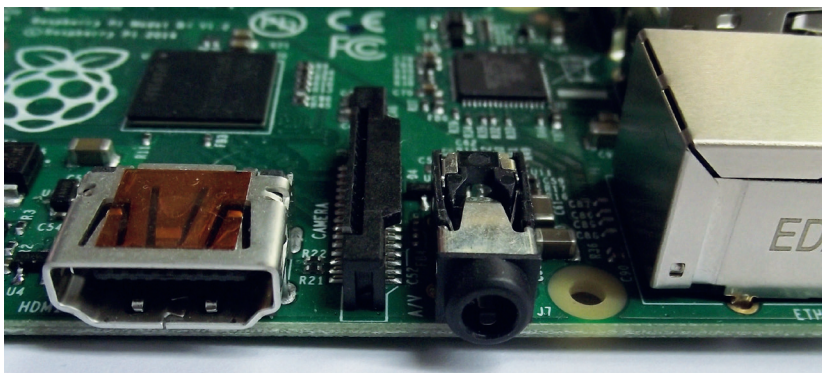
processing power or amount of memory, it would be easy to pass over the differences between the B and the B+. This is a mistake. The differences are important, and add up to a much better computer despite the fact that, underneath it all, it still has the same engine.

The key to the new board is the new regulator. A regulator is a device that adjusts the voltage from a higher voltage down to a lower one. There are several on the Pi, but the main one converts the 5V that comes in from the micro USB power supply into 3.3V that's used on many of the components. The original model B had a linear regulator, which is basically a device that converts the difference between the input voltage and the output voltage into heat. This is highly inefficient, but linear regulators are cheap and simple to use.

The B+ has a switching regulator. This actually converts the input energy at one voltage to a new output voltage. They still waste a little power, because no component is 100% efficient, but far more power makes it through than does with linear regulators.

The 40 GPIOs on the B+ feature 15 PWM channels, one UART, one SPI, one I2C, and one connection to header flash, plus 26 programmable pins.





As well as producing better sound, the B+'s new AV jack looks a lot better than the hulking black monstrosity that was on the Model B.

The new regulator saves between half a watt and one watt of power. This, by itself, isn't very important – it's not a big enough difference that you'll notice lower electricity bills. It will, however, have an impact on anyone running their Pi off solar power or batteries, but this isn't the main reason we're excited about the lower power usage. Most USB power supplies can deliver between one and two amps at 5V, with many of the more common ones being much closer to 1A than 2A. At 1A, this means there's 5 watts of power for the Pi, so the saving is equivalent to 10–20% of the total power available.

By wasting less power, the model B+ effectively makes more power available for peripherals. This 10–20% increase is the difference between the model B not being able to handle an unpowered hub with a mouse keyboard and USB memory stick and the B+ being able to.

Making connections

On the board itself, the thing that stands out more than anything is the addition of two USB ports. This has been made possible by replacing the Lan9512

USB to Ethernet chip with a Lan9514. The bad news is that this still sends Ethernet and USB traffic over the same bus to the SoC, so if you connect high-bandwidth USB devices and a network connection, you will notice the speeds slowing down.

The good news is that the improved power supply on the B+ means that the USB ports are quite usable for low-power devices without an external powered hub. We had no problems at all with just a mouse and keyboard. A mouse, keyboard and USB memory stick also worked fine. A mouse, keyboard and two memory sticks did work, though the power to the mouse dropped out when both the memory sticks were active. We had some success with a USB web cam, though for this and anything higher power (such as a USB hard drive) we'd still recommend using a powered hub.

The exact number of devices you can drive will depend on the power supply your using. The above was tested with a supply rated at 1A. More powerful supplies are available and these will be able to pass on the extra power to the USB ports.

We can't give any hard-and-fast rules, but previously we advised anyone getting a Pi to get a powered USB hub as well (unless it was for an embedded project). Now, our advice has changed to: only get a powered USB hub if you find you need one.

Making music

Let's be honest, the analogue audio on the model B was terrible. It was okay if you just wanted to make a few noises and didn't really care what they sounded like, but for anything more than that, you needed something extra.

The audio over HDMI worked fine, so home theatre systems didn't have a problem. However, many monitors don't have inbuilt sound (or at least not very good inbuilt sound). With the Pi Foundation pushing the musical programming language Sonic Pi (www.cl.cam.ac.uk/projects/raspberrypi/sonicpi/index.html – and indeed on page 78 of this magazine) as a way to get children interested in programming, the poor audio performance over the headphone port needed to be addressed.

To test the audio, we've had a B+ plugged into our stereo running XBMC (see Distrohopper on page 8) for the past two weeks. So far, we haven't noticed any difference between the quality of the B+ and the quality of the sound out of the CD player. Of course, true audiophiles seeking top-quality sounds aren't going to get them from the audio output of a Pi (or anything else that sells for £30, for that matter). The Wolfson Audio Card for the B will no longer work on the B+, because it relies on the P5 GPIOs, which are no longer available.

The B+ has 14 more pins in the GPIO header, but that doesn't mean there are 14 extra programmable outputs. Only nine of these pins are programmable, three are ground and two are reserved for communicating with HATs (see boxout, left).

Hardware Attached on Top (HATs) A new way to configure add-ons

The Raspberry Pi's GPIOs allow programmers access to pins that they can both write to and read from. They also allow hardware manufacturers to create add-ons that use these GPIOs to communicate with the processor. There are a number of additional functions – such as I2C and SPI communications channels – that can be accessed through these pins. However, at present, the process of setting these up is a little awkward.

With the B+, the Raspberry Pi Foundation has introduced what it calls HATs (Hardware Attached on Top). For a device to be classified as a HAT, it has to conform to a set of standards designed to make sure it behaves itself when communicating with the Pi. Expansion boards don't have to conform to the HAT standard to work on a Pi, but they can't call themselves HATs if they don't.

The most important of these standards is that devices must contain an EEPROM (a bit of memory that the Pi can read). This can be used to tell the Pi a bit about what the device

does, and how the GPIO pins should be configured to work properly with the Pi. In technical terms, the EEPROM should contain a device tree that can be loaded by the kernel and will set up the GPIOs correctly. At the time of writing, no HATs are available, though manufacturers will of course be working on them. It's too early to say if they will become popular. We suspect that there will continue to be a significant market for B-style 26-pin boards without EEPROMs. These will be significantly cheaper to manufacture because they won't need as much PCB space if they only cover 26 pins (PCBs are surprisingly expensive, especially for small production runs). The more advanced capabilities won't be needed by many, especially if they only rely on turning GPIOs on and off rather than using a communications protocol. Though the most persuasive reason for hardware manufacturers to keep making devices for the Model B is that there are around 3,000,000 model B's in circulation and they aren't going to be replaced overnight.

A few extra GPIOs aren't usually that important, especially as they don't include any additional support for low-level communications protocols. There's still one each of serial, I2C and SPI. It's easy enough to use the existing I2C or SPI busses to add more GPIOs anyway should you need to, and this is what many add-ons do, as it also protects the Pi from damage due to electrical problems in the circuit they're connected to. Boards like the Protect Your Pi by My Pi (www.modmypi.com/protect-your-pi) use these to provide more GPIOs than the Pi actually has.

However, using these port expanders slows down the speed at which you can turn the GPIOs on and off. This is almost imperceptible if you're just using them to turn LEDs on, or get input from a button, but if you're using them to connect to some other electronics, the delay can be too much. An extra nine GPIOs is enough to be able to implement some communications protocols that require 26 channels, such as those to drive some LCD displays, so it increases the Pi's potential enormously.

Shrinking storage

Of all the changes, perhaps the most superficial is the switch from full-sized to micro SD cards. The two formats are the same from an electrical point of view, so it shouldn't have any impact on speed. In fact, it's perfectly possible to use a micro SD card with a model B if you put it in an adaptor (which many micro USB cards come with). These adaptors also make it possible to copy data onto the small cards from computers that only have full-sized SD card slots.

The only real difference between the two (other than size of course) is that micro SD cards have a barb that can be used to hold the card in place, which should make it a bit more reliable if the Pi's being moved around.

What's next?

Eben Upton, founder of the Raspberry Pi Foundation, has confirmed to us that the Foundation is looking into a model A+, which will do to the model A what the B+ has done to the B. We haven't heard exactly what form this will take yet, but we suspect that some people are particularly interested in a model A with a switching regulator, since this will be even more power efficient than the B+, making it the ideal device for running on batteries (the Model A is used in many embedded projects).

Beyond this, many people are waiting for a version of the Raspberry Pi with more processing power or memory than those currently available, sometimes called the Model C (though people inside the Raspberry Pi Foundation refer to this as the Raspberry Pi 2). It should be obvious by now that the brains at the Pi Foundation aren't interested in constantly chasing the latest hardware

– they are more concerned with providing a stable base and developing software to use it efficiently. Schools – which are the primary target of the Raspberry Pi – don't want to have to spend time and money to change their hardware every year or two just to be able to follow the latest projects.

Having a slow release cycle also helps companies making add-on boards. It enables them to spend time understanding a product, learning what the users want, and designing something properly rather than just rushing to market because it may be obsolete soon.

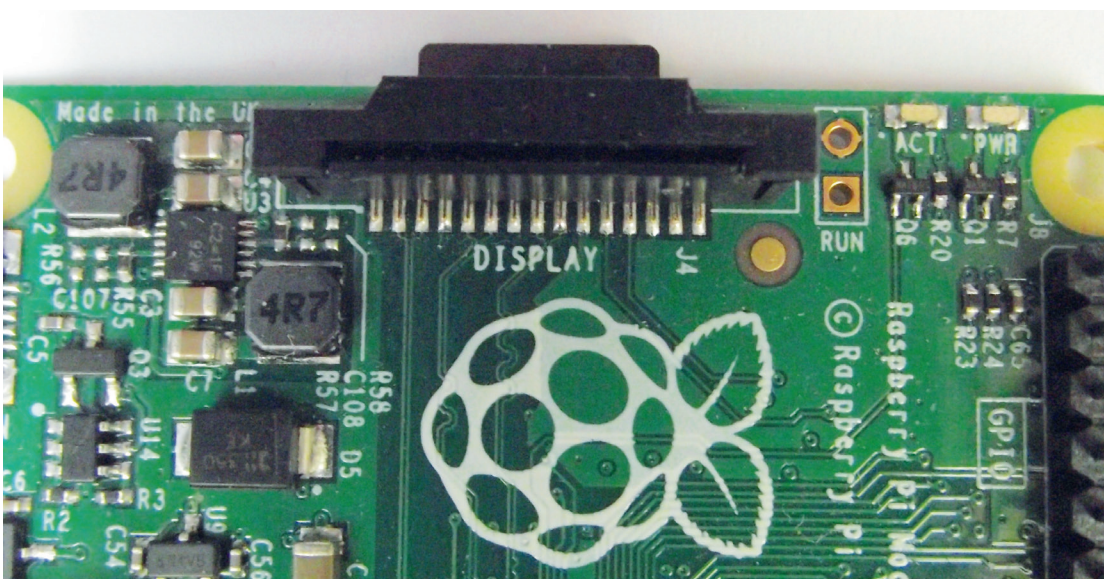
The next version of the Pi is expected in 2017, but don't expect it to be the most powerful ARM board on the market. However, it will be well supported with a large number of add-ons, it will have a large community behind it, and it will be developed by an organisation with the resources to make sure it runs well.

The B+ gives us a number of improvements, but still keeps almost complete compatibility with the older device. We used the word almost because of the lack of Pad 5 GPIO, but this didn't get much use anyway. We're sure that a few people will be disappointed by lack of a new processor or more memory, but in a way, we're not.

The Raspberry Pi is great because it's a stable platform we can build projects on and know they'll

“We're delighted that all the software from the Foundation and community will still run on the Model B+.”

work when recreated by other people. As model B owners, we're delighted to know that all the software from the Foundation and community will still run on our devices, and as B+ owners we're pleased that some of the niggling problems of the Model B have been solved. Now let's get building! 🍷



There's now a connector labelled as the display waiting for the official display module.

Elementary, *my dear Freya*

Elementary OS started out as a collection of attractive icons, but now a small team has taken the ethos and turned it into a completely bespoke top ten distro... **Russell Barnes** investigates.

In many respects elementary OS is the perfect microcosm of the open source scene. It's designed and built by a disparate team working on a project considerably greater than its parts. Unlike many modern Linux distros, though, Elementary OS isn't a hodgepodge of different elements drawn in from different corners of the ecosystem. While it's built on Ubuntu's solid back-end, every other aspect of the distro is entirely custom made. Everything from the desktop environment to the file manager, the

application launcher and even many of the applications themselves have been developed especially for Elementary OS.

As we discover from talking to the core development team, this is probably why it's being embraced by the open source community. But as we also find out, Elementary OS's biggest market isn't Linux at all and as such, they're not afraid to sidestep a few open source norms in their quest to reach the top of the distro pile...

The core elementary OS team

They're separated by thousands of miles, but the core elementary OS team is a tight-knit bunch...



Name: Daniel Foré
Location: Sacramento, California
Career: Design and marketing manager
Project role: Founder and UX designer
Quote: "I started the project drawing icons. My role has evolved from visual design to UX design."



Name: Cassidy James
Location: Denver, Colorado
Employment: Front-end web dev and UX designer at System 76
Project role: Director of Operations & UX Designer
Quote: "I manage the legal and financial side... and help manage the community and guide the project."



Name: Cody Garver
Location: Jackson, Mississippi
Career: IT Consultant for SMBs
Project role: Project and Release Manager
Quote: "I do the packaging and the ISO builds. I help out with the road maps and bug triage among other things."

LV Why do you think the Linux community embraced Elementary OS in the way it did?

Cody Garver: I think it's pretty clear that Elementary as a project has a strong sense of design and a focus on simplicity. So the strong reaction to it shows there's an appreciation of design in open source software. The nature of open source development is that anyone can make anything and just give it away. This type of creation has typically not had a particularly strong design process. There's definitely a growing community of users who enjoy something that is well designed and open source.

LV Were you surprised when Luna (Elementary OS version 0.2, released in August 2013) took off the way it did?

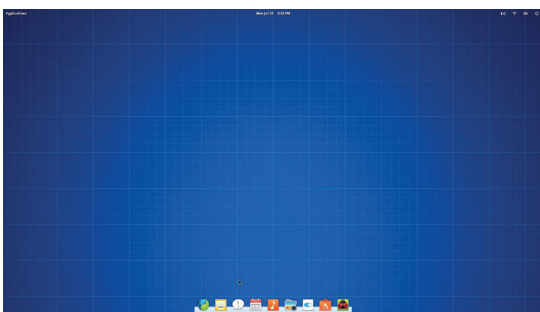
Daniel Foré: Personally, I was pleasantly surprised. I think it goes to show that an exceptional user experience can be something that differentiates you regardless of whether you're an open source or commercial project – having a good user experience is something that users want.

Cassidy James: We saw an opportunity in the open source space for a top-to-bottom solution. With most Linux-based distros someone else builds the environment and other people integrate the apps, the packages and so on... For us, not only do we do the integration, but we build all the apps and we build the desktop environment. Having that approach is something that's really pushed us ahead. You have to expect a certain level of acceptance when you're doing something so different.

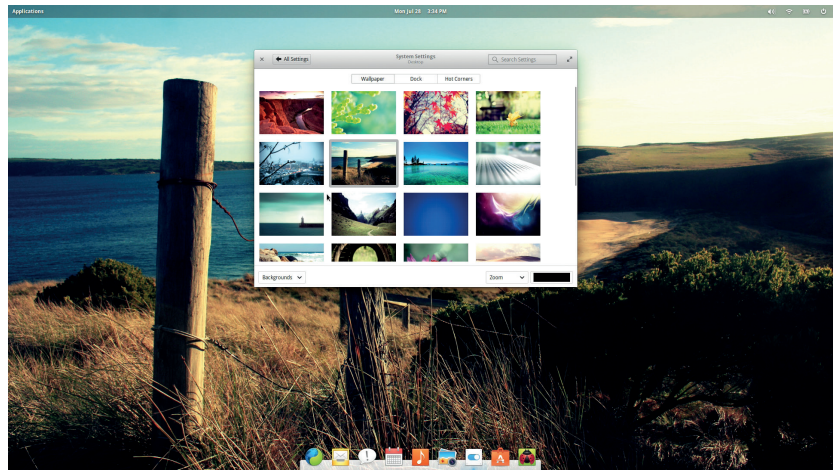
LV So who, in your opinion, is downloading Elementary OS? Are we talking about creative types, beginners or is it a broad cross-section?

CJ: I think the majority of the downloads come from non-Linux users. They're mostly from Windows, several from OS X. I think there's a dissatisfaction with proprietary operating systems out there like Windows 8 and OS X. People are looking for an alternative and the simplicity of Elementary draws them in.

LV Did you see a bump in the numbers when Windows 8 stumbled so spectacularly on the start line or when Windows XP shut-up shop?



Freya's bare desktop has a cleaner look to it than that of most Linux distros.



Freya's top panel is now mildly intelligent, deciding on the fly if it needs to be visible at all

CG: I know there was a lot of chatter on social media sharing Elementary: "I'm switching my/my grandparents computer from XP to elementary so they can get security updates". Or "My school is installing Elementary on the lab computers". We've got a lot of those kinds of stories shared with us since then, which is cool.

DF: Someone shared a photo on Google+ of a prep school in China and kids using Elementary on computers there. That was amazing, and it's really validating. We're making something that not only friends and family are going to use, but that people around the world and are passionate about. It's an incredible feeling.

CJ: Some of the most rewarding stories are the ones related to the accessibility of

computing in general. We get Tweets that say things like: "My mum was using Windows and was having a hard time doing what she wanted... Now she can get on Facebook and send email." Where technology before was a blocker, we're enabling people to communicate, connect and do things.

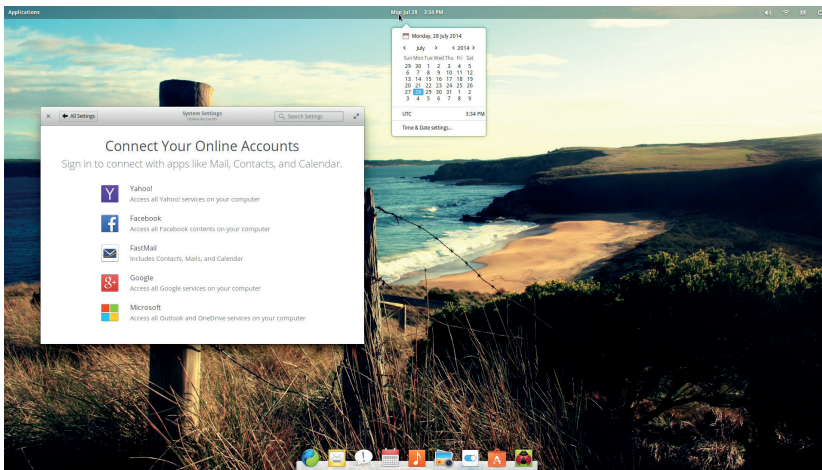
"Where technology before was a blocker, we're enabling people to communicate and do things."

LV What about the other side of the coin? Luna did a lot of things really well, especially in terms of user experience, but what areas were you less pleased with that you're addressing with Freya in September?

CG: File management wasn't optimal. We suffered from a lack of developers in that space. It was a bit crashy but we've rectified it since then, and I'm really excited about it.

CJ: In a lot of ways Luna was the first release for a lot of those apps. It was the first release of our desktop environment too. There were a lot of unexpected things and users threw a lot of cases at us that we hadn't considered throughout the development cycle. As with any new software the first release wasn't jam-packed with features, so we've been working on putting in a lot of new features and dealing with those issues that we hadn't encountered ourselves.





The addition of online accounts will help integrate popular web apps into the Elementary OS experience.



DF: Another of our weaker areas with Luna was networking. There were some issues there that we've gone through and done a lot of work on. One of the most popular requests was Google Calendar sync in our calendar app, so that's another thing we've been working on. There are literally hundreds of issues we've closed during the current release cycle that were reported by users.

LV Does it pile on the pressure? You're ranked in the top 10 on Distrowatch now... are you feeling the heat?

CG: No, but now you mention it I feel like I should!
DF: There's so much that we have in our vision of where we want to be, that we're not really concerned with 'how are we possibly going to continue to compete?'. We know where we're going and we've got such a huge plan for the future that it should naturally keep us there and keep us pushing towards the top.
CJ: The only thing I get nervous about is the next release, because I look at all the bugs we closed in the current one and I wonder how can we find anyone else or ask anyone to complete something like that, yet again.

LV Has the development process changed since the last release?

DF: The end of the Luna cycle was about learning how to work together as a team and focus on our goals. Before then everyone was doing their own thing, and at the end we'd try to tie them all together. Now other developers are more likely to be aware about what everyone else is doing.
CG: We do 100% code reviews now too. Any code that changes is peer-reviewed by other members of the team. It slows things down a little, but we find we're clearing up after ourselves much less now.

LV When you released Luna, it came with the option to donate. How was that received by the community, and did it work for you as a source of income for the project?

CJ: Yeah, it's worked really well for us. I think there's this cultural thing that's been popularised by the 'pay

what you want' approach for software, like the Humble Bundles. With digital software you don't have to pay significant amounts of money for distribution. If you were sending out physical CDs, that's expensive, but as something that's available for free or cheap – pay what you want – people really latched onto that and enjoyed it. We set a default payment of \$10 and people could change it to whatever they want, but several of those payments come through at the default amount.

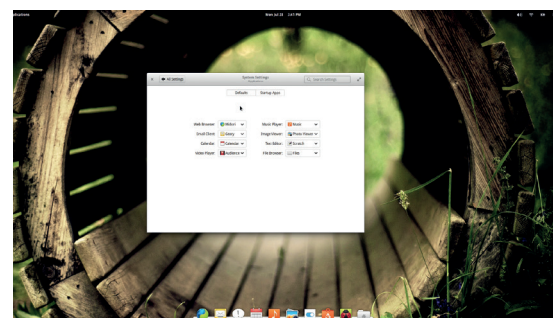
I think it's exciting that you can be creating open source software and producing revenue at the same time. People think it's worth paying for.

We had 1.5 million downloads of Luna alone. That's exciting. I've watched payments come in and a lot of times people either pay £10 or they'll pay \$1 or \$2. Most download for free, but if they're going to pay it's either a small payment or the default \$10.

DF: None of us really know what to expect as far as numbers go. There are so many people in the world it could be anything, or it could be nothing. I had no idea what to expect. I'm looking at the site we have up and it's showing 1.45 million. That's based on figures we can accurately track – how many people have directly downloaded from SourceForge. Then we have a percentage of how many people decided to download from torrents. We can only estimate downloads from that. In theory it could be far more – we can't track downloads from outside either direct or by torrent.

LV With anything between 1.5 to 3 million downloads it doesn't take much napkin mathematics to realise you've made a reasonable amount of money from Luna in the last twelve months. What have you been spending it on and could elementary become your full-time job?

CG: We obviously incur operating costs like running servers, paying for the website, but beyond that we just have small stuff like office supplies and costs for doing in-house shipping of merchandise. Aside from those kinds of operating costs all the money is being invested back into fixing elementary OS with bug bounties. We've been to a few hackathons and meetings too.
CJ: No one takes home a pay cheque. It's all directed straight to goods and services to benefit the project.



The core software offering has expanded and the team have finally introduced the ability to set applications to start with the system.

In terms of it becoming a full-time job... that's a direction we'd like to go in, but currently the money we're making doesn't support full-time employment. We want to work towards that and we want to put money into open source developers' hands. I think getting involved in doing bounties has been a huge first step for that.

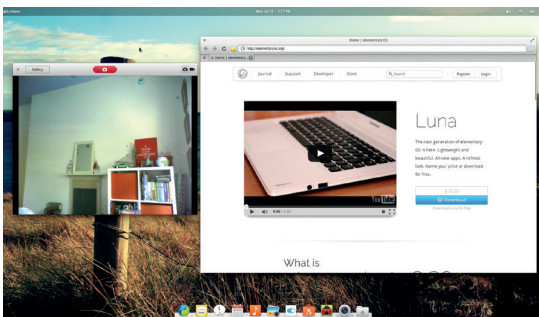
We've posted over \$8,000 in bounties and so far we've paid out \$2,500 of that. It's been a tremendous help, not only in attracting new developers, but keeping our current developers engaged and making sure they feel appreciated. It's hard work and a lot of what we do is really boring stuff that nobody's ever going to know we did. Having that incentive makes it better, because you say "Hey, this is hard work, it's not a lot, but buy yourself something nice". It's a small token of our gratitude.

DF: Our average bounty size at www.bountysource.com has grown with time. We started out offering \$5, \$10 and \$15 dollar bounties on things, but now we have several at \$100 or more for single bugs. As we generate more income through people donating or paying for the download, that's an area we're going to keep investing more and more money into.

LV You're currently working towards the latest release, called Freya, of Elementary OS. Since your focus is on simplicity and cleanliness, have you been finding it hard to add features without adding mess and bloat?

DF: That's one thing we talked about a lot when we started working on the new search back-end. As we introduce new features we want to make sure that – no matter what happens – we don't ever encroach on that original experience of launching apps really quickly. It's the primary purpose of the UI after all. While we want to introduce interesting and useful little features, if it gets in the way of the primary purpose we don't want to add it.

Our app launcher – *Slingshot* – has a new back-end for its search view. For now that doesn't mean anything for users, but going forward it means we'll be able to add different kinds of plugins. We've added a calculator plugin, so you can just open *Slingshot* and do some math. Little convenient things like that. It will sort all your results by most used too, which is nice.



Snap, the new webcam app, couldn't be simpler, and while *Midori* wouldn't be our first choice for browsing, the team are investing heavily in its development.



The team put a lot of stock in Elementary as a brand.

CJ: We've got a new video app too, called *Audience*, a web camera app and a refreshed UI. We're introducing the ability for apps to use a dark theme, like the terminal or media-centric apps. And we're introducing HeaderBar. It's a new widget created in *GTK* that enables the applications title bar and tool bar to be just one line. It's space saving for things like notebooks especially.

DF: HeaderBar was something that was introduced in either *GTK* version 3.10 or 3.12. Typically you have this area above your toolbar that just shows the app's name, and that's it. You're adding 16 vertical pixels for this completely useless area. So it just enables us to compact that area and save that space. In general, *GTK* apps now are using what are called client-side decorations. That kind of goes hand in hand with HeaderBar. What that means on the themeing side is that the window borders are being drawn by *GTK* instead of a separate window manager. There's no tearing when you re-size, because it's part of the contents. Shadows look nicer. We get all the advantages of *GTK 3*-like transparencies, we can animate things – it's just really nice.

CG: We've also rebuilt our multitasking view in the window manager. We have a much more interactive and clearer distilled multitasking experience. It's really hard to describe it, but when you see it and use it it's just intuitive. You can move things around and it feels really good.

"We've posted over \$8,000 in bounties and so far we've paid out \$2,500 of that."

LV You mentioned earlier that you're finally bringing Google calendar synchronising to Elementary OS. Will you be offering similar online account integration elsewhere in Freya?

CJ: We are introducing an online accounts service so apps can tie into that, much like you see in Mobile, Ubuntu and OS X 10. We've added a firewall configuration, start-up apps configuration. Every time you boot up you can start your Twitter client or web browser, for example.

CG: We currently support Facebook, Google,



Collaboration made easy

How do the core developers manage to stay on the same page when they're all at least 1,000 miles apart?

Cody Garver: It's a necessary evil when you're working with open source. We have people in all different locations, timezones and languages. We've adapted really well because we're all of the age that we've grown up with the internet, so we make heavy use of it. We use tools like Launchpad to manage code, but we recently switched over to *Slack* [www.slack.com] from IRC – it's actually a pretty effective way of collaborating.

Cassidy James: One of the things that was important as we moved from Jupiter and into the Luna cycle was learning how to work remotely. We're still learning and evolving. It

was just in this cycle that we started using *Slack* as opposed to IRC. It's been huge. In IRC we had to have all these different services to paste code snippets or share images and we built bots to log the channel and view history... things like that. All these things come built-in with *Slack* so we've been able to take a lot of tools and have them in one place.

Daniel Foré: One area we're still facing some challenges is in animation and motion design. I'm making prototypes in CSS and HTML and trying to hand those off to the dev team, but it can be hard when you're trying to do sound or motion and you're trying to design something like that remotely. You can't use your hands to gesticulate meaning.

Microsoft and FastMail, and there's work going on for general IMAP support. The idea is that apps on the desktop can plug in to the online accounts. You don't have to sign in to everything. It's perfect for things like Twitter clients and email clients.

LV **There's clearly a lot of work going into Freya and there's a lot to be excited about, but do you agree with people who say you're being very protective by not publicly sharing your alpha builds?**

CJ: Absolutely we have! There's this interesting thing with open source... because the code is open and available to everyone, people expect to be able to download and try it out every step of the way. Even with Luna we ran into this.

Even if we provided a preview to our developers and it would get leaked out to a publication or a website and they would review it as a finished product and say, "This is buggy, this didn't feel complete, such-and-such doesn't work well." Well of course not – it wasn't complete. It's really hard when you put so much time and effort into a product we want it to be the best representation of all our work, people judging it early can be a big problem.

LV **It seems that brand management is very important to you, and that's the driving factor here, not secrecy.**

DF: It really does come down to brand management. When people hear Elementary OS or see our logo it needs to make them feel like this product is well thought out, it's stable and easy to understand. When people see others say 'hey, this is Elementary's next release and it's broken and unstable', then it really hurts our potential growth.

LV **On the other side of the coin though, do you not worry that the lack of crowdsourcing and testing on the fly is hurting development in some way?**

CJ: The people who contribute to Elementary on a regular basis and people who are new to contributing to Elementary can get in touch with the dev team and we can help them get started and give them access to pre-release versions so they can work on their apps.

If you're involved in development, or want to be involved in development and you're committed to it you'll definitely get the opportunity to run the pre-release software. It's mostly about people who are less involved and are just going to report everything we can see already. It creates a lot of extra bug triaging and creates a lot of extra work at our end.

CG: We know things are broken and we know what we're focussed on – it hurts our focus.

DF: Throughout Freya's cycle we've picked up quite a few new developers. I don't think the availability of an ISO test image really relates to the ability to hack on the source code at all. All the source code is publicly available. Running Elementary isn't really a requirement to build the apps in most cases, so a lot of people are doing dev work in Ubuntu or Arch or whatever before they're working with us with *Slack* or running these test images. People can contribute code without having ever run Elementary OS.

CJ: Release early, release often... we kind of do the opposite of that!

DF: I think it's the way that big commercial projects work. They may have a yearly release cycle, but they're not held to a specific date – it comes when it's ready. You expect a new Android every 6–12 months, or a new OS X every year, but they could be working on something really cool and don't deliver until it's ready.

Open source has had this other model of release on a really strict schedule – whatever's ready, just release it. That doesn't work as well for us, because when you're building an entire OS you're not going to be ready in just six months.

CG: It has to do with target market too. Typically open source software developers are releasing to other open source software developers, but like we said earlier, the majority of people downloading Elementary aren't coming from Linux.

We're seeing a different kind of consumer. Our consumers don't necessarily know how to deal with this stuff. We can't expect them to run through an unstable system and use commands at a terminal – we can't release a product like that.



The app launcher, *Slingshot*, is adding applications slowly but surely – software only gets included when it's ready.

Hands-on with elementary OS Freya

How is the latest release shaping up?

There's no escaping the fact that Freya, like Luna before it, has taken its lead from Apple. While Apple has continued to build, add and augment its proprietary operating system until it's just as complex and bloated as Windows, Elementary has been incredibly sparing with extensions to Luna's lightweight mix of software and features.

This theme continues elsewhere with brilliant use of *GTK's* HeaderBar. They haven't wasted a pixel in any open window, and it adds a new edge to Elementary's visual appeal and utility. By removing the title bar there's much less wasted vertical space, and that's sure to go down well with users in the mobile space.

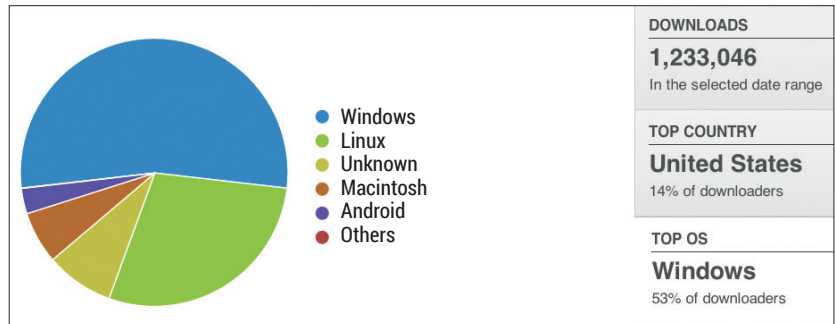
This visual zen doesn't end there – it's also transformed the top panel too, which works particularly well when combined with one of the few new applications to appear in Freya, the *Audience* video player.

Zen minimalism

Like everything else in Elementary OS, it's built from scratch in Vala and offers the barebones of media playback in a very modern, minimalist package. *Audience* has been in development since version 0.2 Luna and draws inspiration from online players like YouTube and Vimeo as opposed to standalone media applications. Using *GTK 3* for the UI and *GStreamer* for its back-end, *Audience* offers animated overlay controls and preview pop-up that lets you scan around to find the start of a scene.

Another new application for this release is *Snap*. As the name suggests it's a simple webcam application, very much in the style of OS X's *Photo Booth*. Like *Audience*, there's very little to say other than it allows users to quickly snap pictures, videos or screencast.

With these new applications the team are clearly fleshing out the core offering to ensure all the major



bases are covered, but they're resisting community pressure to include third-party applications in the mix like *Birdy* and the popular podcast app, *Vocal*.

You certainly get a sense of a more mature Elementary from our early preview of Freya. Features are getting polished, the basic application offering is being refined and it's all happening on the reliable backbone of Ubuntu 14.04.

The only missing piece of the puzzle is *App Centre*. Development here, it seems, is going to take much more than one release cycle to flesh out, and it's clear the team plans to take a firmer grasp of the reins than your average package manager. While this is sure to upset more than a few in the community, as the team explain in the boxout below, they find unlimited freedom has a tendency to lead to a lesser user experience. We'll just grab the popcorn, retreat to a safe distance and let that last statement sink in.

With development timed to coincide with the release of *GTK 3.14*, it's very likely that Freya will see a final release during September. 🍿

It came as quite a surprise to learn that most Elementary OS users aren't Linux users at all – the vast majority of Luna downloads come from Windows and Mac OS X.

“The team plan to take a firmer grasp of the reins than your average package maintainer.”

The absence of App Centre

One of the big banner features people have been waiting for is *App Centre*, Elementary's bespoke app store. Its not happening this year, and here's why...

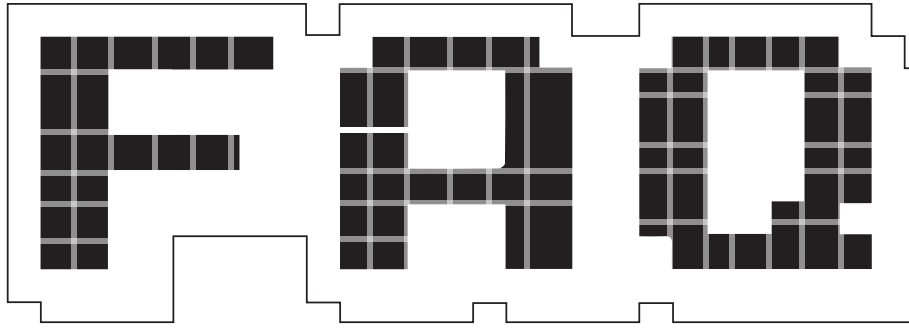
DF: Unfortunately it won't be in Freya. The thing about *App Centre* is that when you're looking at building a new app store the easiest part of that is writing a new client. Gnome has a really great client already. When you open up Gnome Software it looks new and shiny... but when you dig into it – all the content – is all the same content you had before in the old one. You're not really moving forward. We're still getting the same selection of apps written in a billion different toolkits with really horrible descriptions. Some of them are crashy and half complete – it's not curated at all.

The most difficult part in building a new app store is firstly a proper app submission process. That's something we talk about all the time. When we have third-party developers

making cool apps like *Birdy* or *Vocal*, how do they get them to elementary OS users?

A big piece of making it successful is some kind of curation process. We need to have some kind of rules for apps that we present to users... some kind of standard. People in the open source community aren't going to be super-happy about that concept at first, but from our experience unlimited freedom leads to a lesser user experience in the end. Instead of browsing through a collection of all these really nice native apps that are presented well and integrated with the OS, you're just looking at everything everyone ever posted to the internet.

CJ: There was an app in the Ubuntu repos for quite a while called *PornView*. We used to use that as our example of how anything can get in there. *App Centre* is a big project and we think it's a very necessary one. It's just not something we'll be able to complete in just one release cycle.



BSD

Had history been slightly different, you'd be reading FreeBSD Voice today...

MIKE SAUNDERS

Q So what's the deal with this Birsa Seva Dal then? Isn't it a political group in India?

A Very funny – you looked up the "BSD" disambiguation page on Wikipedia just to make that joke, didn't you? Here we're talking about the Berkeley Software Distribution, a family of operating systems that are much more widely used than you might think.

Q Sorry, I couldn't help myself. OK, so what's the deal with these OSes?

A There are three main BSD operating systems in use today. They are based on Unix, they are open source, they tend to be used in server roles, but can also make good desktops and workstations as well. They run KDE, Firefox, LibreOffice, Apache, MySQL and pretty much any open source application you can name. They're reliable, secure and support a lot of different hardware.

"The BSDs are developed as complete projects from centralised source code trees."

Q Congratulations – you've just described GNU/Linux...

A True. Linux has all of the things I've just mentioned, and that's why a lot of people never investigate BSD. In day-to-day usage, there isn't a lot of difference between the BSD family and Linux, largely because they all have Unix underpinnings, and also because they share a lot of software. You could be logged into a remote machine, hacking some Python code in *Vim*, and checking your email in *Mutt*, and you wouldn't know you were running BSD. Or you could be using an internet terminal in a cafe somewhere and not know it's BSD.

The biggest differences are in the development model and licence, and to understand this, we need to step back in time. The B in BSD refers to the University of California, Berkeley, which was a hotbed of open source Unix development back in the 1980s. As the 90s came, x86-based PCs were becoming popular and many people were interested in having a Unix-like OS on their home computers. A project called 386BSD was released in 1992 to provide just that.

Q And where were all the Linux distributions at this time?

A Good question! You might know that one year before, Linus

Torvalds had announced his kernel, which, when paired with the GNU project, formed a complete open source operating system. Linus had been following GNU's own kernel (Hurd) and 386BSD, and said that had either of them been ready for daily use, he probably wouldn't have created Linux. So the first few years of the 90s were tremendously lively for open source operating systems, and nobody was really sure which ones would succeed.

Then it got messy for BSD. AT&T, the original developer of Unix, was trying to monetise its work on the operating system and claimed that BSD infringed its intellectual property rights. This culminated in a lawsuit in 1992 which severely held back BSD development. In the end, various chunks of the BSD source code had to be rewritten – while all this time, GNU/Linux was gaining features, stability and popularity.

BSD was arguably in a more mature state than GNU/Linux in the early 1990s, and without these legal complications it could have become the standard on x86 PCs. We could all be using it today instead of Linux.

Q But you said earlier that BSD is still widely used, so things improved after that?

A Yes. 386BSD development stagnated, but two teams of

developers working over the internet created separate successor projects. FreeBSD became the most widely used flavour of BSD, and is now the closest to Linux as a desktop and server operating system, while NetBSD focused on portability (today it runs on over 50 platforms, all built from the same codebase). The third flavour, OpenBSD, forked off from NetBSD just a few years after NetBSD started due to a developer spat, and today it's well known for its concentration on security. Over the years, OpenBSD has created many programs that have become standard on Linux, such as *OpenSSH* – and now we have *LibreSSL* too.

Q So these three flavours of BSD are like Linux distributions?

A Yes and no. Each BSD has a separate codebase and separate development teams, although there is a lot of code-flow between them (especially for hardware drivers). But they are standalone operating systems with their own features, pros and cons.

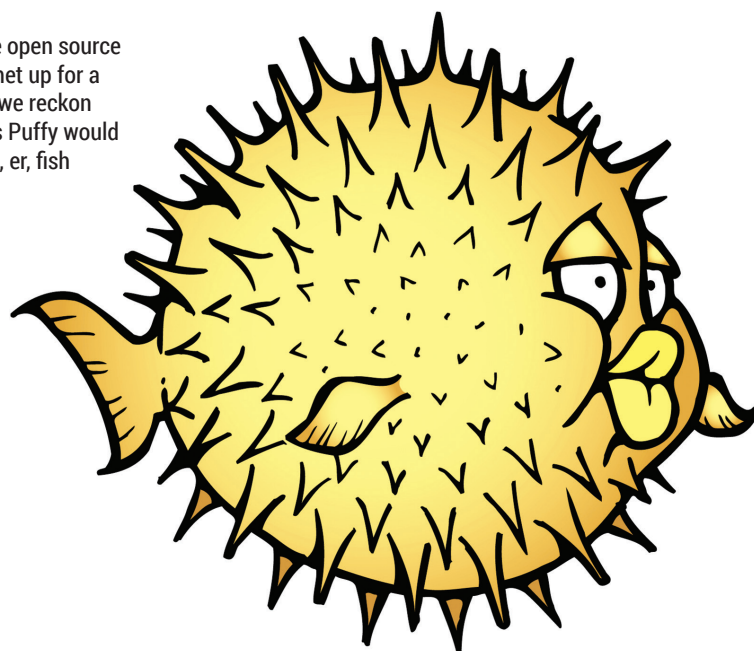
We mentioned that the development model of the BSDs is one feature that really distinguishes them from GNU/Linux. There's nobody in charge of GNU/Linux as a whole: some teams are working on the GNU components, some are working on the kernel, some on boot scripts, some on manual pages, some on libraries, and so forth. The development model is often called "wild west", with a lack of central authority, and distributions do all the hard work of fitting everything together.

The BSDs, in contrast, are developed as complete projects from centralised source code trees. The kernel, the libraries, the system utilities and the manual pages are all stored and worked on in the same place. Many BSD fans argue that this gives the operating systems more coherency and stability, and from our years of dabbling with BSD we can attest that the manual pages are largely superb.

Q Don't the BSDs use anything from GNU/Linux?

A Yes, especially *GCC*. The *GNU Compiler Collection* has been the *de-facto* standard compiler on free Unix systems for decades, although FreeBSD has recently moved to *LLVM/Clang*. It's important to note that the

If all of the open source mascots met up for a big scrap, we reckon OpenBSD's Puffy would be the last, er, fish standing.



BSDs also use other open source projects that aren't specifically GNU or Linux, such as the X Window System (*XFree86* and *X.org*), Perl and so forth. And thanks to standards such as POSIX, most programs that run on Linux can be recompiled to run on the various BSD flavours.

So, you could replace the L in a LAMP (Linux, *Apache*, *MySQL* and *PHP*) stack with FreeBSD, and get pretty much the same environment, with a different set of features (eg variations in filesystem and driver support). And there are some mega, super, huge users of FreeBSD, such as Netflix, which serve up ridiculous amounts of data every day. While FreeBSD makes a good desktop OS, its strengths really lie in the server room, with exceptional reliability and network performance.

OpenBSD tends to be used in smaller web serving, file hosting, firewall and gateway roles where security is imperative. NetBSD is the least popular of the main BSD flavours – it can run on almost anything though, including old Amigas and Acorn boxes, and sometimes finds itself inside closed-source network devices.

Q Hang on – how can someone close the source code? That ain't kosher in Linux!

A Correct, and here we come to the other major difference with GNU/Linux. The licence for the BSD flavours (called, funnily enough, the BSD Licence) is very different to the GPL that

we know. For starters, it's much shorter. The BSD Licence essentially says: do what you want with this code, but give the original developers credit for writing it, and don't try to sue them if it blows up your computer.

So there's nothing in the licence that forces the code to stay open, unlike with the GPL, which requires that users of the code also make their modifications freely available. This crucial difference has sparked countless flame wars over the years, with BSD fans saying that their licence is more free (because it's less restrictive), while GNU/GPL fans say that their licence is actually more free (because it preserves freedom down the road).

Q Blimey. Anyway, now that you've piqued my interest, where can I try out all these lovely BSD flavours?

A You can probably guess the websites – www.openbsd.org, www.freebsd.org and www.netbsd.org – where you can download ISO images, boot them in *VirtualBox*, and play around. If you've been using Linux for a while, you won't find any of them too difficult, although you're expected to know your way around the command line. If you're looking for something more newbie-friendly, PC-BSD (www.pcbbsd.org) is a customised version of FreeBSD focused on the desktop, with a fancy graphical installer and super-simple management of software. Have fun exploring! 🐟

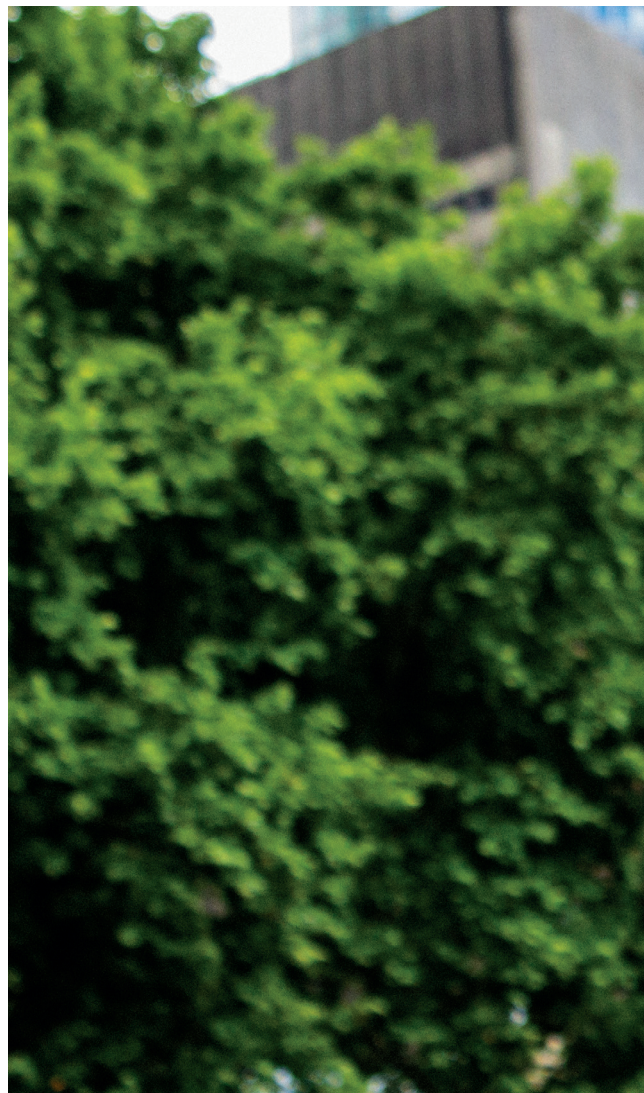
MIR VS WAYLAND: THE BATTLE TO REPLACE THE X WINDOW SYSTEM

Mir was big during the space race and it's a big part of Canonical's unification strategy. We talk to one of its chief architects at mission control.

Not since the days of 2004, when *X.org* split from *XFree86*, have we seen such exciting developments in the normally prosaic realms of display servers. These are the bits that run behind your desktop, making sure Gnome, KDE, Xfce and the rest can talk to your graphics hardware, your screen and even your keyboard and mouse. They have a profound effect on your system's performance and capabilities. And where we once had one, we now have

two more – *Wayland* and *Mir*, and both are competing to win your affections in the battle for an X replacement.

We spoke to *Wayland*'s Daniel Stone last month, so we thought it was only fair to give equal coverage to *Mir*, Canonical's own in-house X replacement, and a project that has so far courted controversy with some of its decisions. Which is why we headed to Frankfurt and asked its Technical Architect, Thomas Voß, for some background context...



LV Let's go right back to the beginning, and look at what X was originally designed for. X solved the problems that were present 30 years ago, where people had entirely different needs, right?

Thomas Voß: It was mainframes. It was very expensive mainframe computers with very cheap terminals, trying to keep the price as low as possible. And one of the first and foremost goals was: "Hey, I want to be able to distribute my UI across the network, ideally compressed and using as little data as possible". So a lot of the decisions in X were motivated by that.

A lot of the graphics languages that X supports even today have been motivated by that decision. The X developers started off in a 2D world; everything was a 2D graphics language, the X way of drawing rectangles. And it's present today. So X is not necessarily bad in that respect; it still solves a lot of use cases, but it's grown over time.

One of the reasons is that X is a protocol, in essence. So a lot of things got added to the protocol. The problem with adding things to a protocol is that they tend to stick. To use a 2D graphics language as an example, XVideo is something that no-one really likes today. It's difficult to support and the GPU vendors actually cry out in pain when you start talking about XVideo. It's somewhat bloated, and it's just old. It's an old proven technology – and I'm all for that. I actually like X for a lot of things, and it was a good source of inspiration. But then when you look at your current use cases and the current setup we are in, where convergence is one of the buzzwords – massively overrated obviously – but at the heart of convergence lies the fact that you want to scale across different form factors.

LV And convergence is big for Canonical isn't it?

TV: It's big, I think, for everyone,

especially over time. But convergence is a use case that was always of interest to us. So we always had this idea that we want one codebase. We don't want a situation like Apple has with OS X and iOS, which are two different codebases. We basically said "Look, whatever we want to do, we want to do it from one codebase, because it's more efficient." We don't want to end up in the situation where we have to be maintaining two, three or four separate codebases.

That's where we were coming from when we were looking at X, and it was just too bloated. And we looked at a lot of alternatives. We started looking at how Mac OS X was doing things. We obviously didn't have access to the source code, but if you see the transition from OS 9 to OS X, it was as if they entirely switched to one graphics language. It was pre-PostScript at that time. But they chose one graphics language, and that's it. From that point



“Mir will be significantly more relevant than Wayland in two years.”

on, when you choose a graphics language, things suddenly become more simple to do. Today's graphics language is EGL ES, so there was inspiration for us to say we were converged on GL and EGL. From our perspective, that's the least common denominator.

Obviously there are disadvantages to having only one graphics language, but the benefits outweigh the disadvantages. And I think that's a common theme in the industry. Android made the same decision to go that way. Even *Wayland* to a certain degree has been doing that. They have to support EGL and GL, simply because it's very convenient for app developers and

“We want to provide a super fast experience for users. It should be rock solid.”

toolkit developers – an open graphics language. That was the part that inspired us, and we wanted to have this one graphics language and support it well. And that takes a lot of craft.

So, once you can say: no more weird 2D API, no more weird phong API, and everything is mapped out to GL, you're way better off. And you can distill down the scope of the overall project to something more manageable. So it went from being impossible to possible. And then there was me, being very opinionated. I don't believe in extensibility from the beginning – traditionally in Linux everything is super extensible, which has got benefits for a certain audience.

If you think about the audience of the display server, it's one of the few places in the system where you've got three audiences. So you've got the users, who don't care, or shouldn't care, about the display server.



It's transparent to them.

TV: Yes, it's pixels, right? That's all they care about. It should be smooth. It should be super nice to use. But the display server is not their main concern. It obviously feeds into a user experience, quite significantly, but there are a lot of other parts in the system that are important as well.

Then you've got developers who care about the display server in terms of the API. Obviously we said we want to satisfy this audience, and we want to provide a super-fast experience for users. It should be rock solid and stable. People have been making fun of us and saying “yeah, every project wants to be rock solid and stable”. Cool – so many fail in doing that, so let's get that down and just write out what we really want to achieve.

And then you've got developers, and the moment you expose an API to them, or a protocol, you sign a contract with them, essentially. So they develop to



Whether *Mir* will dominate over *Wayland* remains to be seen, but Thomas is confident.

your API – well, many app developers won't directly because they'll be using toolkits – but at some point you've got developers who sign up to your API.

LV The developers writing the toolkits, then?

TV: We do a lot of work in that arena, but in general it's a contract that we have with normal app developers. And we said: look, we don't want the API or contract to be super extensible and trying to satisfy every need out there. We want to understand what people really want to do, and we want to commit to one API and contract. Not five different variants of the contract, but we want to say: look, this is what we support and we, as Canonical and as the *Mir* maintainers, will sign up to.

So I think that's a very good thing. You can buy into specific shells sitting on top of *Mir*, but you can always assume a certain base level of functionality that we will always provide in terms of window management, in terms of rendering capabilities, and so

on and so forth. And funnily enough, that also helps with convergence. Because once you start thinking about the API as very important, you really start thinking about convergence. And what happens if we think about form factor and we transfer from a phone to a tablet to a desktop to a fridge?

LV And whatever might come!

TV: Right, right. How do we account for future developments? And we said we don't feel comfortable making *Mir* super extensible, because it will just grow. Either it will just grow and grow, or you will end up with an organisation that just maintains your protocol and protocol extensions.

LV So that's looking at *Mir* in relation to X. The obvious question is comparing *Mir* to *Wayland* – so what is it that *Mir* does, that *Wayland* doesn't?

TV: This might sound picky, but we have to distinguish what *Wayland* really is. *Wayland* is a protocol specification,

which is interesting because the value proposition is somewhat difficult. You've got a protocol and you've got a reference implementation. Specifically, when we started, *Weston* was still a test bed and everything being developed ended up in there.

No one was buying into that; no one was saying, "Look, we're moving this to production-level quality with a *bona fide* protocol layer that is frozen and stable for a specific version that caters to application authors". If you look at the Ubuntu repository today, or in Debian, there's *Wayland-cursor-whatever*, so they have extensions already. So that's a bit different from our approach to *Mir*, from my perspective at least.

There was this protocol that the *Wayland* developers finished and back then, before we did *Mir* and I looked into all of this, I wrote a *Wayland* compositor in Go, just to get to know things.

LV As you do!

TV: And I said: you know, I don't think a protocol is a good way of approaching this because versioning a protocol in a packaging scenario is super difficult. But versioning a C API, or any sort of API that has a binary stability contract, is way easier and we are way more experienced at that. So, in that respect, we are different in that we are saying the protocol is an implementation detail, at least up to a certain point.

I'm pretty sure for version 1.0, which we will call a golden release, we will open up the protocol for communication purposes. Under the covers it's Google buffers and sockets. So we'll say: this is the API, work against that, and we're committed to it.

That's one thing, and then we said: OK, there's *Weston*, but we cannot use *Weston* because it's not working on Android, the driver model is not well defined, and there's so much work that we would have to do to actually implement a *Wayland* compositor. And then we are in a situation where we would have to cut out a set of functionality from the *Wayland* protocol and commit to that, no matter what happens, and ultimately that would be a fork, over time, right?

LV It's a difficult concept for many end users, who just want to see

something working.

TV: Right, and even from a developer's perspective – and let's jump to the political part – I find it somewhat difficult to have a party owning a protocol definition and another party building the reference implementations. Now, Gnome and KDE do two different *Wayland* compositors. I don't see the benefit in that, to be quite frank, so the value proposition is difficult to my mind.

The driver model in *Mir* and *Wayland* is ultimately not that different – it's GL/EGL based. That is kind of the denominator that you will find in both things, which is actually a good thing, because if you look at the contract to application developers and toolkit developers, most of them don't want *Mir* or *Wayland*. They talk ELG and GL, and at that point, it's not that much of a problem to support both.

“We never said we would come up with the perfect solution in version 1.”

So we did this work for porting the *Chromium* browser to *Mir*. We actually took the *Chromium Wayland* back-end, factored out all the common pieces to EGL and GL ES, and split it up into *Wayland* and *Mir*.

And I think from a user's or application developer's perspective, the difference is not there. I think, in retrospect, if there would have been

something like a full reference implementation of *Wayland*, where a company had signed up to provide something that is working, and committed to a certain protocol version, our decision might have been different. But there just wasn't. It was five years out there, *Wayland, Wayland, Wayland*, and there was nothing that we could build upon.

LV **The main experience we've had is with RebeccaBlackOS, which has Weston and Wayland, because, like you say, there's no that much out there running it.**

TV: Right. I find *Wayland* impressive, obviously, but I think *Mir* will be significantly more relevant than *Wayland* in two years time. We just keep on bootstrapping everything, and we've got things working across multiple platforms. Are there issues, and are there open questions to solve? Most likely. We never said we would come up with the perfect solution in version 1. That was not our goal. I don't think software should be built that way. So it just should be iterated.

LV **When was *Mir* originally planned for? Which Ubuntu release? Because it has been pushed back a couple of times.**

TV: Well, we originally planned to have it by 14.04. That was the kind of stretch goal, because it highly depends on the availability of proprietary graphics drivers. So you can't ship an LTS [Long

Term Support] release of Ubuntu on a new display server without supporting the hardware of the big guys.

LV **We thought that would be quite ambitious anyway – a Long Term Support release with a whole new display server!**

TV: Yes, it was ambitious – but for a reason. If you don't set a stretch goal, and probably fail in reaching it, and then re-evaluate how you move forward, it's difficult to drive a project. So if you just keep it evolving and evolving and evolving, and you don't have a checkpoint at some point...

LV **That's like a lot of open source projects. *Inkscape* is still on 0.48 or something, and it works, it's reliable, but they never get to 1.0. Because they always say: “Oh let's add this feature, and that feature”, and the rest of us are left thinking: just release 1.0 already!**

TV: And I wouldn't actually tie it to a version number. To me, that is secondary. To me, the question is whether we call this ready for broad public consumption on all of the hardware versions we want to support?

In Canonical, as a company, we have OEM contracts and we are enabling Ubuntu on a host of devices, and laptops and whatever, so we have to deliver on those contracts. And the question is, can we do that? No. Well, you never like a 'no'.

Usually, when you encounter a problem and you tackle it, and you start thinking how to solve the problem, that's more beneficial than never hearing a no. That's kind of what we were aiming for. Ubuntu 14.04 was a stretch goal – everyone was aware of that and we didn't reach it. Fine, cool. Let's go on.

So how do we stage ourselves for the next cycle, until an LTS? Now we have this initiative where we have a daily testable image with *Unity 8* and *Mir*. It's not super usable because it's just essentially the tethered UI that you are seeing there, but still it's something that we didn't have a year ago. And for me, that's a huge gain.

And ultimately, before we can ship something, before any new display server can ship in an LTS release, you need to have buy-in from the GPU vendors. That's what you need. **LV**



Thomas is based on Bochum, but ventured out to Frankfurt-am-Main to talk to us.

Write once,
deploy everywhere.

ubuntu 



Back issues are now available at
<http://shop.linuxvoice.com/products/single-issues>

LINUX VOICE REVIEWS



Andrew Gregory

The riddle of the broken DVD drive is solved: there was a broken DVD in it.

This issue we've tested two products that cost money. One is cheap as chips at £5, while the other starts at £195 and goes all the way up to £5,965. Which of these offers the most freedom, the most potential to empower intellectual growth? Of course, it's the one that uses Free Software. The CamJam Edukit is limited in function, but it can be the gateway to a world of experimentation and burnt-out motors. Mathematica is also a superb product, but its licensing makes it feel like a gatekeeper rather than a gateway; you can do more, you can unlock more features, as long as you pay more money and accept that you'll only be able to program what the makers want you to program.

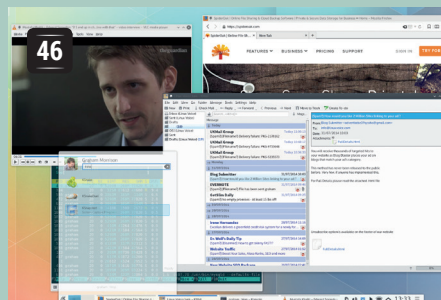
Freedom!

I must admit that I take free software for granted these days, and it's only when I come up against a licence imposition that I stop to think about how lucky we are. You don't get the basic Python interpreter for free and have to pay extra for the most useful modules. Our only responsibility is that we have to do something with all this great stuff that we have to play with, show others how much fun it is and help everyone to learn. Now, **import RPi.GPIO...**

andrew@linuxvoice.com

The latest software and hardware for your Linux box, reviewed and rated by the most experienced writers in the business

On test this issue...



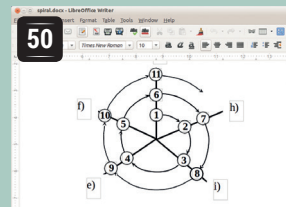
KDE 5

Ardent KDE admirer **Graham Morrison** really hopes that the latest version has learned the lessons of the 4.0 debacle...



Mathematica 10

If you want to analyse, visualise or program with some data (and **Ben Everard** often does) this tool is aimed at you. Oh, you should be rich too.



LibreOffice 4.3

Mike Saunders doesn't use the 1% of MS Office's features that make it worth paying money for. If you don't either, you really should be using LibreOffice.



Stellarium 0.13

This wonderful piece of Free Software makes **Ben Everard's** dark stumble home from the pub a far more astronomically educational experience.

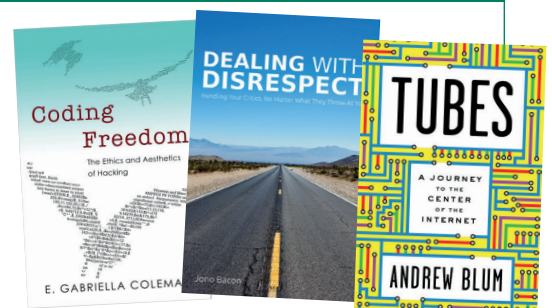


CamJam Edukit

This tiny beginner's kit has taught **Andrew Gregory** more about electronics than 13 years in English schools did.

BOOKS AND GROUP TEST

Facebook is a vampire squid, sucking the vitality out of our interpersonal relationships and reducing lifelong friendships to a tap of the F5 key or a downward swipe of the smartphone. Also, all your data are belong to Facebook, so don't try to delete anything ever again, because it's not yours anymore. We can't change Facebook, but we can recommend a better alternative: IRC. This ancient chat protocol is as open as they come, and there are about a million clients to choose from – we help you find the best one. Also in old but proven technology – books!



KDE Frameworks and Plasma 5 (aka KDE 5)

Previous major updates of this desktop environment have been as popular as the apocalypse. Fortunately, **Graham Morrison** is wielding Andúril this time.

DATA

Web kde.org
 Developer KDE developer
 community
 Licence GPL

Writing two pages on the latest release of KDE 5 is a tough proposition. But this is a good thing. Were we to step into Bill and Ted's telephone box outside the Circle K and take ourselves back to 2008, surrounded by the fallout from the release of KDE 4, we'd be in a rather different situation. There would be so much to write about, nearly all of it negative, that we wouldn't know where to start. KDE 4.0 didn't work, because its users expected a fully fledged desktop upgrade and the first major release should have been an early alpha release instead. It took years before the sum of all the new technologies that were tested in 4.0 became a viable replacement for 3.x.

This should never have happened, and we think that the KDE team and many other open source projects learned from the experience – even though the Gnome team initially seemed to follow the same path with Gnome 3.0. KDE 5 avoids making the same

“KDE 5 avoids making the same mistakes as KDE 4, but not in the way you may be expecting.”

mistakes, but not in the way you may be expecting. It's not a fully fledged desktop but nor is it a direct replacement for KDE 4, and

that's the difference. KDE 5 is being developed as a framework from which KDE 4 applications and technologies can migrate from their old systems to the new systems without stunting the development or progress of either and in a way that shouldn't cause any disruption.

KDE Frameworks 5 was released on 7 July. It contains around 50 different libraries that have been designed to be as modular and as portable as possible. These are split into four tiers, with

Everything in KDE 5 moves very elegantly, from the window transitions to the icon resizing.

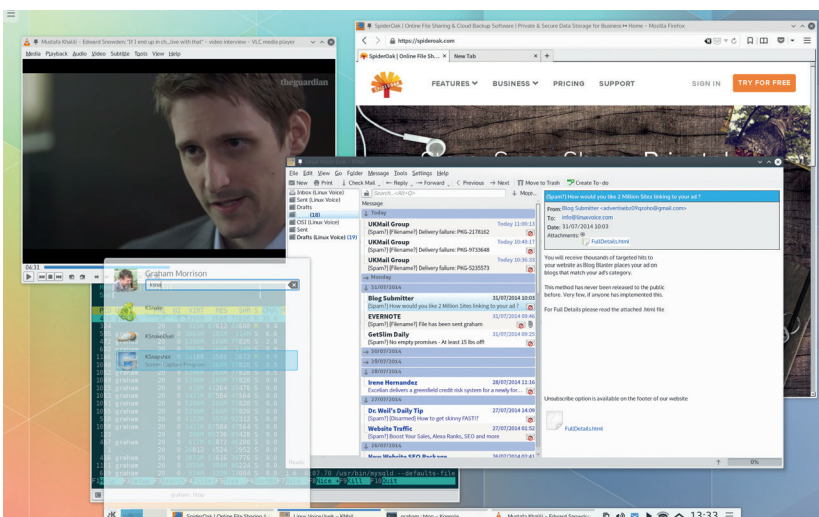
the majority of frameworks (21) falling into tier 1. This means their only dependency is *Qt 5*, which itself brings many, many performance and feature enhancements. Tier 1 frameworks include those that deal with archives, codecs, hardware integration and specific GUI additions, and developers can now include these without any further KDE dependencies, basically as a KDE-flavoured extension to the considerable features already offered by *Qt*. This will help many developers appeal to an audience who don't want to install the entire KDE desktop just to get hold of an application or two, and this should lead to the development of more KDE applications that run independently of the desktop.

What's waiting under the tree

This is all great for developers, we hear you say, but what about us humble users? Is there anything in KDE 5 we can click on now? The answer is yes, but it's far from ready. Released a week after frameworks 5, Plasma 5 is the beginnings of the KDE 5 desktop experience. At the moment, Plasma 5 consists of a new theme called Breeze, a new panel and notification system, a window and widget management system that's accelerated through OpenGL (ES), a new application launcher and a new interface to the Alt+F2 powerhouse known as *KRunner*. With the exception of the graphics acceleration, all of this could be done with KDE 4, and Breeze can already be made to run on older versions of KDE. The advantage with recreating these wheels for KDE 5 is that the design team can play to KDE 5's advantages, and that's exactly what they've done.

Breeze is a flat theme in the same style as Windows 7/8 with a default background that seems to borrow a triangular motif from the latest Ubuntu. The window borders are minimal and we like the pastel vs solarized colour palette. The system tray widgets look fantastic on our display, and we love the new notification system. As the KDE team themselves say, "interaction patterns are left intact", which we think means you interact with Plasma 5 in exactly the same way you interact with KDE 4. And we think is a very good thing indeed; KDE 4 works brilliantly, and there's no reason to mess around with the formula.

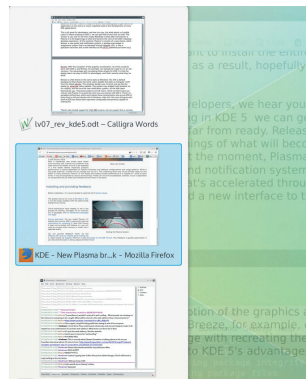
With its dependence on a launch menu and panel, some commentators accuse KDE of being stuck in the middle of the previous decade, but we're yet to see a convincing argument for doing things differently. The Windows 8 user interface is a disaster, both Gnome 3 and Unity are still trying hard to convince their users, and Apple's OS X hasn't really changed in over a decade. KDE's window management is



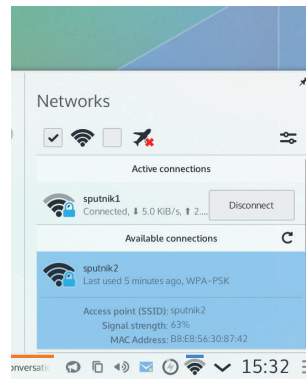
Plasma 5 in details



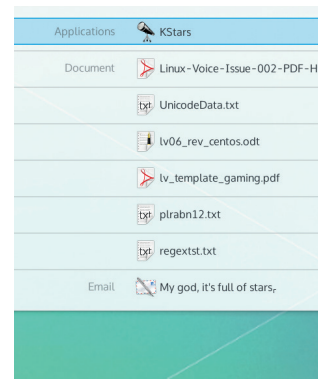
Launcher The launch menu now has pervasive search and a new flat look tied to your settings.



App switcher App and activity switching, and Plasma widgets, have all become more discreet.



Notifications One area now holds all notifications and the network manager settings.



KRunner The Alt+F2 launch system is now simpler and has had a graphical overhaul.

peerless, the desktop can be as minimal as you want it to be and you have access to an unrivalled number of configuration options. This is a desktop that can still be made to look and operate exactly how you want it to, but it takes effort.

High DPI

This initial release is supposed to support high-DPI screens, but we suspect this is coming for free from Qt 5. Its support in Plasma 5 suffers in the same way Qt does; it's very good at scaling GUI elements when it knows the pixel density of your screen, but there's no automatic way of telling it. And as you still need to use other applications, such as those from KDE 4 and those using GTK, it all quickly becomes a non-standard mess of changing font sizes and hoping for the best. As Aaron Seigo recently posted on Google+, "Fonts and screen DPI and scaling and kittens crying. Trust me, it all comes together."

The good news is that the KDE 5 high-DPI rendering looks fantastic on screens with a high pixel count – much better than KDE 4, and with more developers using laptops with unfathomable resolutions, this problem will hopefully receive some much-needed attention. And not too soon, in our opinion. This is a long overdue problem for Linux, and one where Ubuntu's Unity is currently leading the pack.


Don't upgrade yet!

As you might expect from an early release, there are enough teething problems with Plasma to stop us from recommending it now, especially on a machine you rely on. 'plasmashell' crashed five times over two weeks, and always restarted gracefully. We couldn't use the window manager's effects configuration page because there was an incredibly long delay whenever it loaded. It took many minutes to enable a single option, for example. Many of KDE's settings panels are missing, in particular the panel that configured a touchpad, which we found tricky. On our laptop,

brightness control worked when you pressed the button twice, while the buttons to control audio volume didn't work at all until we'd loaded the KDE 4 mixer. Both then had different on-screen display themes, and integration with *KWallet* didn't seem to work. Some apps were fine, whereas others – most importantly *KMail* – became unusable as they asked for a password every time they accessed the network.

KRunner has been humbled, not offering as many plugins at its KDE 4 counterpart, but hopefully that will come, and we can't believe the new battery applet looks so good and yet still doesn't tell you how much time is remaining, only a useless percentage. Convergence has also been mentioned, and there are different plasma shells for different form factors, but we've yet to see the point.

For everything else, there's KDE 4, and we don't think it will be long until Plasma 5 improves to the point where many people will be able to switch over. It will then be a case of waiting for apps to be ported to KDE 5 for the full native experience, a process that looks quite complex to our untrained eyes but not as difficult as KDE 3 -> KDE 4, and the process will be worth the transition.

KDE 5 looks good. It's faster and more efficient and it's the future. But until then, there's no disadvantage to sticking with KDE 4 and waiting a while before making the jump. 

"It won't be long until Plasma 5 improves to the point where many people will be able to switch."

LINUX VOICE VERDICT

For developers, the upgrade is worth the effort. For users, it's going to take some time before this patient strategy pays off.



CamJam EduKit

Andrew Gregory dips his toe into the ocean of robotics and GPIO programming with this cheap and cheerful beginner's kit.

DATA

Web
www.camjam.me/edukit
Developer
Tim Richardson, Michael Horne & Jamie Mann
Price
£5.00

When the Raspberry Pi launched in 2012 it was clear that it would rise or fall on the strength of the supporting material. And so it has proved; there are more powerful and cheaper devices out there, but the Pi has grown a huge community providing how-tos and projects, and several third-parties have popped up selling add-on equipment.

One of these is the CamJam EduKit. A collaboration between Michael Horne and Tim Richardson of Cambridge Raspberry Jam in partnership with The Pi Hut. The CamJam EduKit is a cheap (£5.00) box of components and a complementary set of worksheets downloadable from camjam.me/edukit.

The components in the box comprise a breadboard, three LEDs, some jump leads, a handful of resistors, a buzzer and a switch. Using these elements, you can make a simple circuit powered by the GPIO pins

on the Raspberry Pi, gradually adding more functions until you've got a little device that responds to input from the command line and from the included

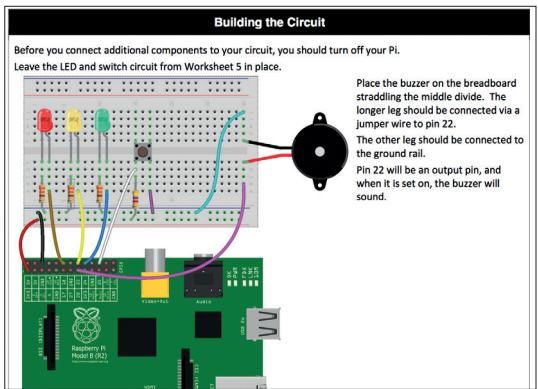
switch. And that's it. At the time of writing, there are six worksheets, which start with the very basics and move up to importing Python modules and accepting input from the user.

From the ground up

When we say that the EduKit starts with the basics, we mean the absolute basics. The first worksheet describes the process of plugging in the Pi, booting it and writing a Hello World script in Python. This

"If you want to have a go at robotics but don't know where to start, start here."

Is it a kid's spy kit? It could be... It's an EduKit, ready for the making!



The documentation is fantastic – everything is clear, even if you haven't played with electronics before.

assumes a Raspbian installation, but we used our brand new B+ with the Noobs kit. This is no big deal, as Noobs clearly identifies Raspbian as the recommended choice of OS.

Another consequence of using a B+, rather than the model B that the worksheets were written for, is that the B+ has an extra 14 GPIO pins. This could be enough to confuse an absolute beginner, but a quick Google search reveals that the first 26 pins are laid out in exactly the same configuration as they always were, so any old guides are going to be compatible with the new Pi. Actually, forget that: while we were writing this review, Michael Horne updated the (excellent) documentation to include a reference to the model B+.

For our money, the EduKit is an unqualified success. There are no moving parts; you won't be building a robot out of an ice cream tub with this kit, or anything more advanced than a traffic light system, but that's not the point. What it does do is open up the door, just a crack, into the possibility that you might build these things in the future, and that's what makes it brilliant.

Like the Pi itself, it's cheap enough to be a stocking filler for a curious child, and if they don't like it, you've only wasted a few quid. But if it takes root and fires something in your imagination, that £5 becomes the best value possible. If you want to have a go at robotics but don't know where to start, the answer just got a lot simpler: start here.

LINUX VOICE VERDICT

A perfect introduction into the complicated world of electronics tinkering. Our appetite is whetted.



Mathematica 10

Ben Everard wonders whether the new version of Mathematica is more intelligent than he is. Mathematica knows but won't tell him.

It's a little hard to say exactly what *Mathematica* is. It's a programming language, IDE, data source, natural language processing toolkit, equation solver and data visualiser all wrapped up into one piece of software. If you want to do something, and it involves data, *Mathematica* can probably do it.

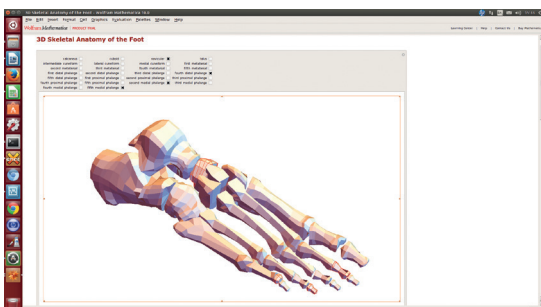
Prices start at £195 + VAT for an individual (or £80 for students), but quickly rise if you want more advanced features (including technical support). The top level costs £5,695 and includes (among other things) support for up to 16 processing cores, phone support, upgrades, *Wolfram Workbench* and *WebMathematica Amateur*.

That's quite a lot of money, but there is a one-month free trial available to help you find out if *Mathematica* suits your needs. Be warned, though, the trial version is crippled to the point that most of the example code on the *Mathematica* website won't run. The trial will also give you access to the Wolfram Cloud (<https://programming.wolframcloud.com/app>), which is able to run most things, but the trial account is limited there as well, and some of the more processor-intensive tasks exceed the trial limits.

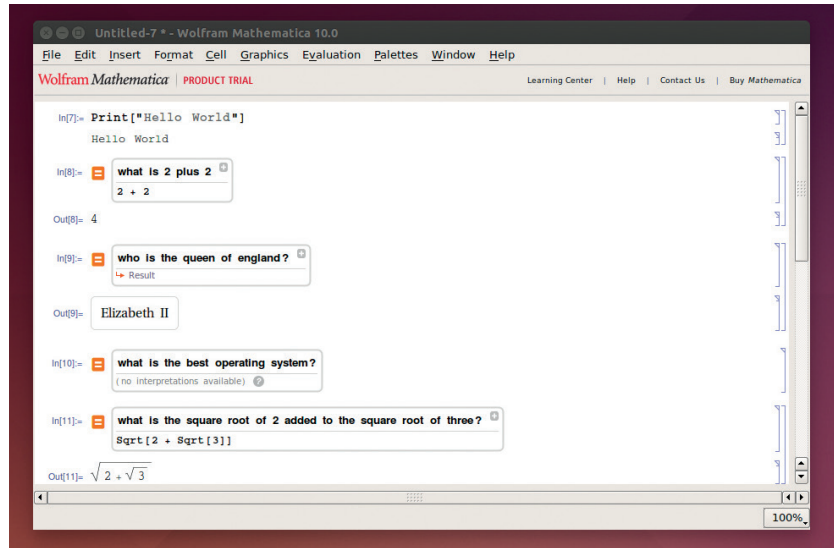
Version 10 brings three new areas to *Mathematica*: machine learning, geographic computation and geometric computation, as well as improvements in just about every area, including many in the Connected Device Framework. There's a list of new features at <http://reference.wolfram.com/language/guide/SummaryOfNewFeaturesIn100.html>.

Geographic computation is largely based on geovisualisation, which is just a fancy word for colouring in maps. This is something that's becoming increasingly popular as a method of visualising data. The integration of the map data and the graphing functions with the language are make the new version of *Mathematica* probably the easiest tool to do this available today.

The best feature of the new machine learning area is its ease of use. *Mathematica* can handle almost all of the algorithm selection and configuration – tasks



The new tools in geovisualisation and machine learning are especially exciting, and incredibly easy to use.




As well as its own programming language, *Mathematica* can take input in normal English.

that can take experience to get right if you have to do them manually – leaving the user with just the task of linking in the data set.

Enormous data processing power

The Connected Device Framework has also seen some improvements. This is the toolset that's designed to bring data from external sensors into *Mathematica* so that you can analyse it, and Wolfram is targeting this at hobbyists with example code for Arduino. Although this is very powerful, most sensor data is quite simple, and analysing it in *Mathematica* would be like using a sledgehammer to crack a nut. It's very rare to need this level of processing power outside of industrial settings.

We're pleased to see improved testing libraries in version 10. These are part of a push from Wolfram to make *Mathematica* a more attractive environment for software engineering, and are something that's been lacking in previous versions.

Mathematica is a uniquely powerful piece of software that, when used well, can help you perform incredibly powerful computations very easily. However, the price of using it is tying your work up with proprietary software. While we do use closed source software, we're uncomfortable with the idea of intertwining our programming this closely with software that we can't control. 

DATA

Web
www.wolfram.com/mathematica
Developer
 Wolfram Research
Price
 £195 +

LINUX VOICE VERDICT

Very powerful and easy to use, but hampered by a lack of freedom.



LibreOffice 4.3

The Document Foundation claims “you can’t own a better office suite” than this. Mike Saunders gets out his Truthometer 9000™.

DATA

Web
www.libreoffice.org
Developer
The Document Foundation
Licence
GNU GPLv3/MPL

On the desktop, *LibreOffice* is arguably the most important free software project in existence. Sure, we all love Linux and sing its praises from the rooftops, but there’s still a long way to go before every home user and business running Windows makes the switch. It’s easier to give people their first taste of open source by recommending applications to them, and *LibreOffice* is a great example: it does 99% of the jobs that 99% of people do in *Microsoft Office*, for zero cost. Home users and businesses can see that free software is more than capable of replacing proprietary applications, saving huge amounts of money along the way.

It’s a 215MB download (for the 64-bit .deb packages), and an empty *Writer* window consumes 98MB of your RAM banks, in contrast to 85MB for the previous version. So it’s slightly heavier, but much

“LibreOffice does 99% of the jobs that that 99% of people do in MS Office, at zero cost.”

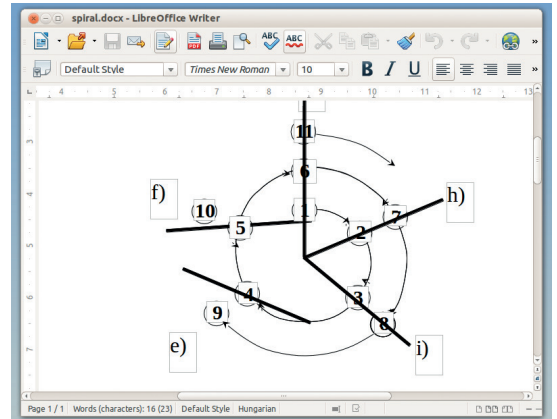
work is being done to make the overall program smaller and more suitable for use on mobile devices.

But the big changes in *LibreOffice 4.3* are the

end-user-facing new features. OOXML documents that previously looked broken in *LibreOffice* should now render much more correctly, especially those that use DrawingML (see the screenshots). Other import filters have been added for *Microsoft Works* spreadsheets and files from *ClarisWorks* on the Mac.

Calc, the spreadsheet, has been boosted in various areas: 30 formulas have been added to enhance

And here it is again in *LibreOffice 4.3*, with everything in its right place.



Here’s an OOXML file containing DrawingML, rendered completely broken in an earlier version of the suite.

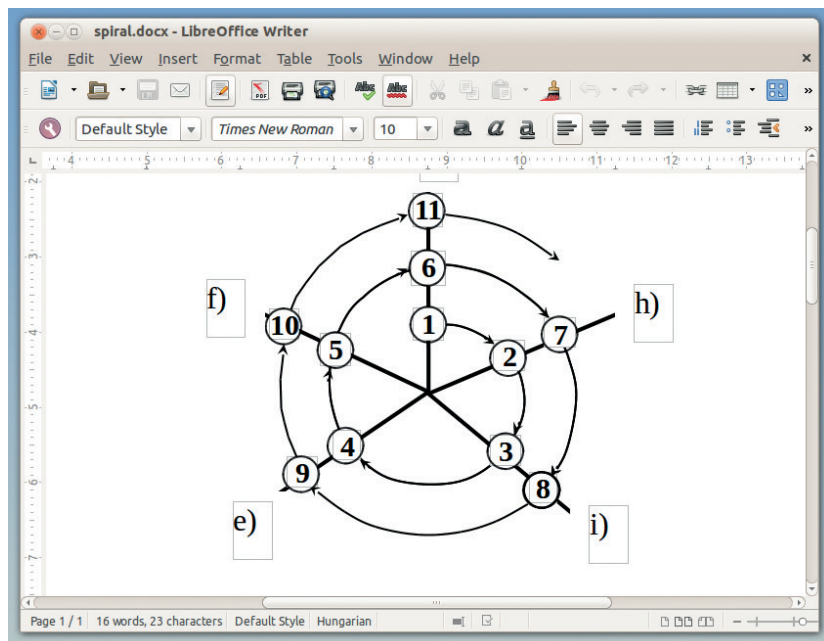
compatibility with *Excel*, and in-cell formulas are displayed with better highlighting. When you select a bunch of cells, the status bar now shows the number of rows and columns. Collaboration, meanwhile, has been made easier with improved commenting features, including nested comments and the ability to export these comments in various file formats.

Shiny new toys

Impress has seen its share of updates too. Presentations can be made prettier with 3D models (created in the emerging glTF format), while initial support for COLLADA and .kmz files has been incorporated. Then there’s a mountain of bugfixes, GUI improvements and documentation updates covering every aspect of the program.

LibreOffice is miles ahead of *Microsoft Office* in many key areas: file format import and export; support for many different operating systems; cost of ownership; and so forth. The advancements made in the OOXML import filters make it increasingly viable as a drop-in replacement for Microsoft’s products.

But then, we don’t use *MS Office* and don’t rely on some of its obscure or rarely used features. Home users and businesses will have to try *LibreOffice 4.3* to see if it finally does everything they need, and opens all their files without problems. One thing’s for sure: there’s never been a better time to switch. 🍷



LINUX VOICE VERDICT

New features are good, but it’s the improved *Microsoft Office* compatibility that really makes this a worthwhile upgrade.

★★★★★

Stellarium 0.13

View the infinite glory of the night sky through your computer monitor. Now **Ben Everard** has another reason not to go outside.

If you haven't used *Stellarium* before, you're in for a treat. It creates a skymap of stars, planets, comets, and most other astronomical phenomena. If, like us, you live in a city, and the night sky is washed out by a sheen of light pollution that only a few of the brightest stars can break through, then *Stellarium* gives you the chance to see what the sky should look like. If, on the other hand, you're lucky enough to live somewhere with a dark sky, *Stellarium* provides the tools to learn the different stars and constellations. If you already know the constellations, then it gives you the chance to view past and future astrological events from any position.

When you start *Stellarium*, it takes over your screen and provides an OpenGL-rendered view of the sky at the current time. You can set the location to anywhere you want on earth, or you can set off on a virtual tour of the galaxy and see the night sky from the surface of other planets, moons and stars without needing a NASA-sized budget.

If a real-time rendering of the sky isn't exciting enough for you, you can speed it up and watch the stars move at super-speed, or start a meteor shower to add a little graphical delight. There are options to tweak just about every aspect of the scene, including which astrological objects appear and how bright they are, what projection is used to compress the universe onto a 2D screen, and the amount of twinkle the stars have. This might sound like pointless clutter, but as you play with the settings, you get a feel for how the real physical objects and effects interact to create the night sky.


Aside from the graphical rendering, *Stellarium* also has a great user interface, which manages to be both powerful and unobtrusive. Unobtrusive in this context means that it both stays out of the way and gives the whole screen to the sky map, and that it works well



in low-light conditions so that it won't ruin your night vision if you use it in the dark. This makes it perfect for running on a laptop outside on a clear night. You can also get *Stellarium Mobile* for Android and iOS devices (a port by the original author of *Stellarium*), but this isn't free software.

Planets in your pocket

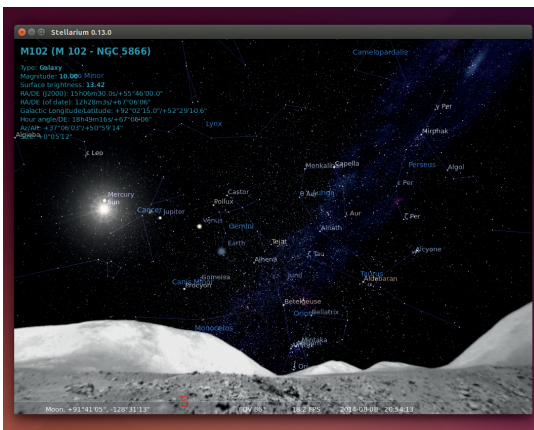
Version 0.13 comes with lots of new graphical wizardry, though none of it changes the basic way *Stellarium* works. Comets and meteors are now a bit prettier, and there are new plugins to help with field of view and time settings. If you're hoping to be the next Bear Grylls, then you may find the new plugin with navigational stars useful. It should also hog fewer resources now, so is a useful upgrade for people running older hardware. All in all, it's a gradual – though not particularly exciting – improvement on the previous version.

Don't be put off by the low version number: *Stellarium* is a great, stable application that's been around since 2001, and has been widely used for much of the past 13 years. It's fascinating as a curiosity, but you can also delve deeper and use it as a tool to learn more about the universe and our place in it. Be careful though: it will make you want to buy a telescope, travel to the desert and stare at the white-flecked darkness above. 

Stellarium can show you the position of the stars – and constellation artwork – at any time of day or night.

DATA

Web
www.stellarium.org
Developer Stellarium
 development team
Licence GPL



A view of the heavens from the surface of the moon.

LINUX VOICE VERDICT

Even the least astronomically-inclined person is likely to have fun with *Stellarium*. It's wonderful.



Tubes: A journey to the center of the internet

Ben Everard is now wondering if he can spend his next holiday on a guided tour.

What is the internet? Think about that question for a moment. Is it an abstract concept that doesn't exist in physical space? Or is it a tangible, finite thing that you could reach out and touch if only you knew where to find it?

In *Tubes*, Andrew Blum takes the view that it's a set of computers and routers, and the high-bandwidth cables that connect them. In this case, then, the internet must exist, and since it exists, it must be possible to see it. *Tubes* is a book documenting his quest to find the internet, or at least the physical things that comprise it.

Where is your mind?

Blum manages to get access to a surprising amount of infrastructure, and he takes the reader along with him as he tours internet exchanges and cable landing points, and meets the people who keep them running.

This isn't a detailed book about the structure of the internet, though it does leave the reader with a basic overview of this. Instead, this is a book that adds colour to the bland network topographies that comprise most descriptions of the internet.

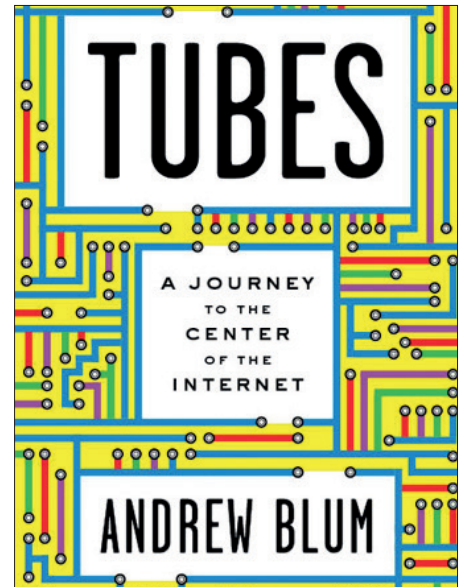
The reader is left with gives a splendid overview of what goes on behind the scenes of the defining engineering accomplishment of our age and one we often don't give ourselves the space to think about.

LINUX VOICE VERDICT

Author Andrew Blum
 Publisher Ecco Press/Penguin
 ISBN 978-0-061-99493-7/978-0-141-04909-0
 Price £9.99

Discover the wondrous locations your data visits after you send it down the intertubes.

★★★★★



The looks like some form of abstract PCB routing that has nothing to do with the internet.

Great North Road

SCI-FI

It took a year, but Graham Morrison finishes a book about Newcastle upon Tyne.

There have been times when we've really enjoyed Peter F Hamilton, most recently with the completion of the *Void* trilogy in 2010. This was a fantastic series of multithreaded yarns that weaved medieval adventure into a possible future where fictionalised versions of Larry Page and Sergey Brin ruled the ultimate philanthropic corporate universe. It had moments of brilliance, even if its length meant the thrill of those original themes became a little jaded. But it was another reason why Hamilton is still one of the best current proponents of 'the space opera' – the literary equivalent to listening to Bohemian Rhapsody by yourself in the car.

The same could be said of *Great North Road*, currently a rare one-off title from the same author. It features many characters in different environments that may or may not come together at some pivotal point in the story. The setting for most of these

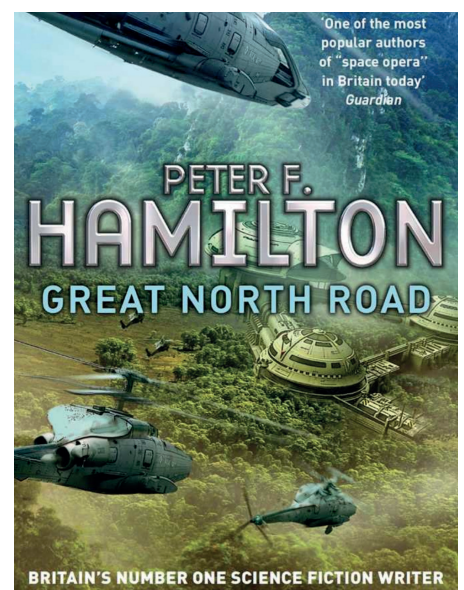
characters is rain-sodden Newcastle upon Tyne circa 2142. There are wormholes, invisible alien threats and lots of Geordie accents, but this is essentially a crime drama. Hamilton is also trying to highlight carbon emissions, sustainability and the unforeseen consequences of taking from the environment. These elements do feel awkward, and the book never quite feels as harmonious as his other work, which is perhaps why we finished many other books before finally completing this one.

LINUX VOICE VERDICT

Author Peter F Hamilton
 Publisher Macmillan
 ISBN 978-0-230-75005-0
 Price £6.99

Hamilton's usual mix of grandiose themes with many characters, but lacking his usual impact.

★★★★★



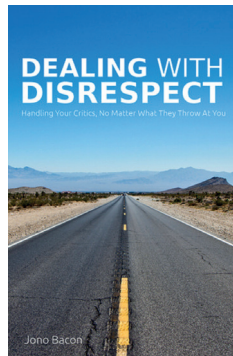
Even with wormholes and awesome computing power, those silly humans are still wreaking havoc on unsuspecting environments.

Dealing With Disrespect

Graham Morrison learnt long ago to never read the comments.

Jono Bacon has fought more than his fair share of flamewars online, and he's not the only one who has noticed the vehemence of commentators increasing. And we completely agree. Disagreements can reach disproportionate levels of hate, and this can be a real problem to the average sensitive and introverted geek. Not only can it lead to depression and diminished self-worth, but it's also pushing valuable contributors out of the community. We've seen more than one insightful mind turn their back on a project simply because of too much bile.

The message behind Jono's short book is simple; you're not alone. Using anecdotes and a personal style, the book walks the reader through the mental journey we're assuming Jono went through in learning to come to terms with disrespect, and how to turn any knock in confidence and uncertainty into a positive force. That the book is also available for free is also rather noble, and while it won't



We're on a road to nowhere.

be for everybody, we suspect there's a considerable number who will take quiet solace from its publication.

LINUX VOICE VERDICT

Author Jono Bacon
Publisher self-published
Price CC BY-NC or \$2.99 on Kindle
 There's too little written about this subject, and it's great to see Jono highlighting the seriousness of a growing problem.



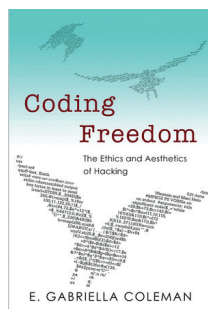
Coding Freedom: The Ethics and Aesthetics of Hacking

We've been studied, and Ben Everard has the results.

Have you ever watched National Geographic as an anthropologist describes the customs of a tribe and wondered what the tribe thought about the film? If you have, and you're a free software coder, this might be your chance to find out.

Gabriella Coleman is an anthropologist who studies us. By us, I mean the elusive tribe of people that work on open source software. In *Coding Freedom* she takes a scholarly approach to analysing what it means to be someone who develops free software, and she's taken the time to understand this properly. In doing so, the book also covers the concept of free software, what it means to the community, and why people within the community are so attracted to it.

Coding Freedom is heavy going, especially to someone not familiar with the languages of the social sciences. If you make the effort to read it, it's quite



The birds on the front cover are made up of the text of a Perl script to remove encryption from DVDs.

interesting, but in its current format, it's more suited to reading by social scientists than computer scientists.

LINUX VOICE VERDICT

Author E Gabriella Coleman
Publisher Princeton University Press
ISBN 178-0-691-14461-0
Price £16.95

A detailed, informative, but hard-to-read guide to the people behind free software.



ALSO RELEASED...



One day soon, we'll all need big data tools to search our photos.

Data Algorithms

After reading Ben's excellent data analysis tutorial in this very issue (p82), we've got a genuine taste for big data, which is why this book looks rather excellent. It promises to give you the super-powers to crunch through petabytes of data. Excellent!



Create the next *Inception* from the comfort of your Linux box.

Mastering Autodesk Maya 2015

Blender is incredible, but we often forget that there's a tier 1 3D application available for Linux that many studios already use, and that's *Maya*. It may be expensive, but it's a native application capable of Hollywood-quality rendering. This book takes it up a level.



Think of the children.

Design for Kids

It's lovely that there's a renaissance in teaching technology to kids, but developing interfaces for children is a completely different challenge to the ones we're used to. This is why this ebook looks useful. It helps designers deal with the emotion, ego and impatience of the average child, while helping them learn.

GROUP TEST

Graham Morrison Experiences the old world charm of a group chat technology that predates Facebook by a generation.

FIND US ON IRC!

#linuxvoice is on Freenode, and it's a friendly and welcoming channel for everyone interested in Linux, Free Software and beer.

On Test

Konversation



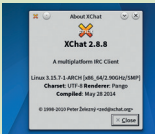
URL konversation.kde.org

Version 1.5

Licence GPL

The strongest of several KDE IRC clients with more config options per pixel.

XChat



URL xchat.org

Version 2.8.8

Licence GPL

Perhaps the mostly widely used default IRC option for most distributions.

Smuxi



URL <https://smuxi.im>

Version 11.0.0

Licence GPL

Super powerful, despite its austere GUI (which is currently being upgraded).

Quassel



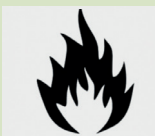
URL www.quassel-irc.org

Version 0.10.0

Licence GPL

A lovely GUI for power users with the best server/client split we tested.

Irssi



URL irssi.org

Version 0.8.16

Licence GPL

Lots of script and a great console GUI, but lacking recent development.

WeeChat



URL weechat.org

Version 0.4.3

Licence GPL

It's difficult to use, and its command-line based, but it's soo powerful.

IRC Clients

We've become so used to the idea that newer is better that it's difficult to envisage anything old competing with anything new. Web browsers, desktops, laptops and even distros are overhauled so often that running an old version feels difficult and anachronistic.

But there's one significant exception, and that's something called Internet Relay Chat (IRC). To the uninitiated, it's like chatting in Google Talk or Facebook Messenger with more than one person at the same time. IRC is a child of the BBS-era (Bulletin Board Systems), predating the world wide web, the first SMS messages, hashtags and the rise of social media. And because you'd often have to dial into a BBS from a low-bandwidth modem, efficiency was everything. Even the fastest modems of 1988, when IRC was created, connected at a mere 2400 baud – that's only 2400 bits per second in the technology of the time. JPEGs might take 30 minutes

to load, video conferencing and voice was impossible, and that left text, and the initial rise of IRC.

That IRC has survived and thrived in the 21st Century is a testament to its original design and simplicity. Get a client, connect to a server and join any channel you find interesting. Channels in this sense are a little like the channels on Citizens Band radio of the 80s, and there are channels for everything, from exploring your Arch fetish or early masterpieces of the Ultima franchise with the Exult channel, to 3D printed psycho robots (#robotics) and your very own Linux Voice (look us up on Freenode).

IRC use is also growing, not just because it's an open platform out of the control of big corporations; it's also mature, secure (if you want it to be), and globally accessible. Now that more of us are working remotely, IRC has become the perfect medium for both informal chat and serious planning. Which is why finding the perfect client has never been so important.

“IRC has become the perfect medium for both informal chat and serious planning.”

WHAT MAKES A GOOD CLIENT?

IRC is a contentious platform, a little like email. Old-school users will swear by their command-line tools, while others will like the cuddly ease of a nice GUI. As far as we're concerned, it doesn't matter as long as it gets people using IRC instead of Facebook Messenger or Google Hangouts. But a client does

need to be reliable, transparent and flexible, and if possible, accommodate as wide a spectrum of users as possible. It's these attributes that we've focused on, so that whichever client you end up using, you should be able to use indefinitely until something better comes along.

Get started with IRC

If none of what we're writing about here makes any sense, read this first

As we've mentioned on the first page, IRC is a form of group messaging where the groups are hosted on a server. You can create your own server using something like *UnrealIRCd*, but the majority of users connect to servers that are already running. The most popular is a

network with a Free Software bias called Freenode (it's not really a single server, but a portal to a network that's automatically load-balanced and managed).

Freenode peaks at around 80,000 consecutive users, it's still growing, and it's where you'll find our own channel,

#linuxvoice. Other networks include IRCNet, Efnets and QuakeNet. Channels are usually moderated by one or more operators who have the power to kick (or ban) people from the channel if they're not adhering to the rules. Any rules will appear when you connect to a network or to a channel.

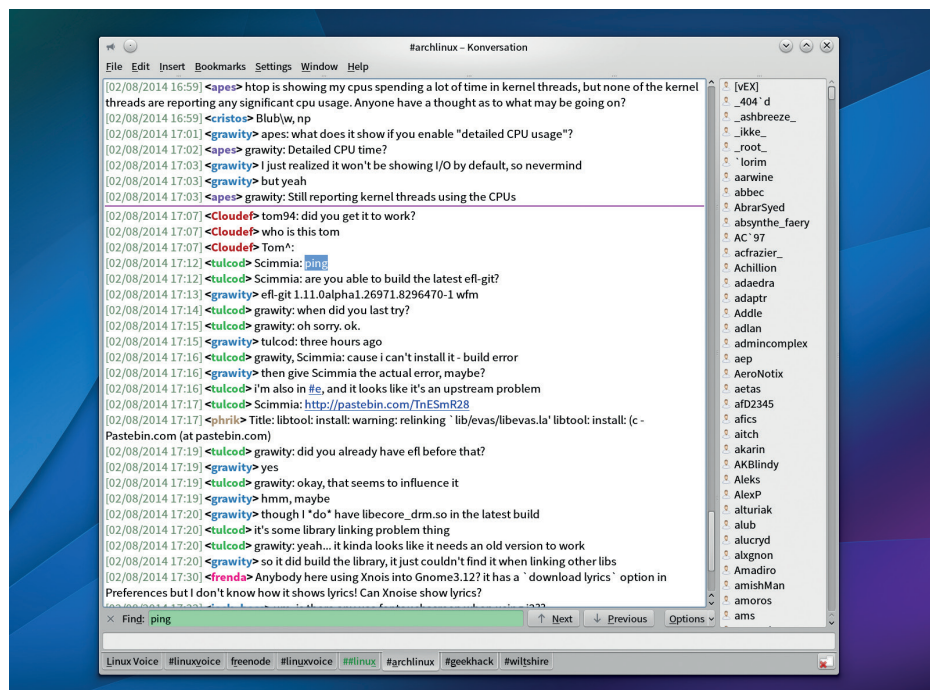
Konversation 1.5

The KDE Kontender is a tough act to beat.

Firstly, we like *Konversation's* GUI because it's both minimal and utterly configurable. The list of users within a channel can be discreetly slid over to the side of the main window, and while it's a difficult option to find, you can expand the input box to use multiple lines. Font support is excellent and you can change the colours for everything. We wish these options could be encapsulated into a theme engine to aide easy import, export and sharing, or take some hint from the global colour scheme, because we like to change between dark and light themes depending on the time of day, but it can be made to look exactly how you want it to.

Server logs and messages for each channel are tabbed. Tabs can be moved to the lower, upper or left borders of the main window, and the clever Watched Nicks, URL catcher service and the DCC status panel can exist within their own tabs too. You get on-screen notifications containing new messages, and the system tray icon flashes with new updates. A channel list can also be opened on a separate tab, and kept open, which is a better solution than the pop-up windows offered by most other IRC applications. Entering messages themselves is easier with the multi-line input, and we rely on the excellent auto-spellchecking. As with other clients, pressing Tab will complete a nickname, and you can right-click on various GUI elements to create shortcuts to a variety of IRC commands. Right-click on a nick, for example, and you can enter message mode or perform a 'whois' on a user. All useful stuff for people without IRC in their DNA.

Finally, this wouldn't be a KDE application if it didn't enable you to open another tab



If this group test were about the number of configuration options, *Konversation* would win.

containing the excellent KDE terminal console, *Konsole*. We also like the way *Konversation* handles multiple connections and servers, although it's a little counter-intuitive. This is because, to add a new server, you need to link a server with an identity. We think this is to facilitate KDE's global identity functionality, so that your name and contact details are set in one place and used in many. If you use multiple servers with the same nick, you can simply add them from the identity dialog.

You can also have more than one nick per identity (this is getting complicated), but the separation between identities, servers, nicks and channels is useful if you use IRC for

both work and for social networking. There are plenty of options that enable you to connect to channels automatically, register your nicks or accomplish almost anything else through a script. A separate field for identities is useful if you use ZNC for multiple servers and need different login values, but you'll need different identities for different servers as there's only one field per identity (and not per nick).

VERDICT

A excellent option for KDE users, and worth the KDE library install for everyone else.

★★★★★

XChat

This is the client you've probably already got installed.

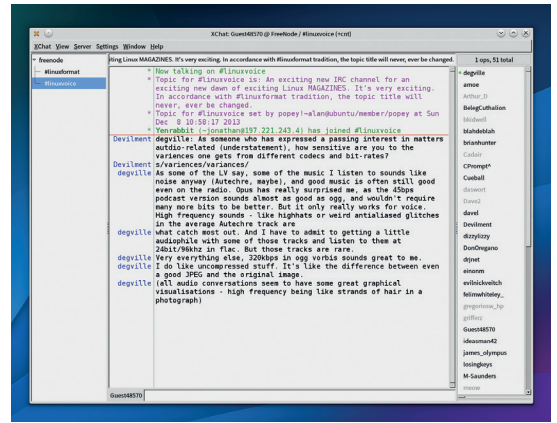
If there's an Old Pretender to the IRC client crown, it's *XChat*. It's been around since 1999 and it's also one of the most portable graphical clients we're looking at. There are versions for Windows, Linux and OS X, and it's also possible to run the client as both a graphical application and a command-line utility.

XChat is also the default IRC client for many distributions, and the first client many people go to when they start experimenting with IRC. This isn't a bad thing at all. *XChat* is stable, functional and easy to use. When you first launch the application, for instance, it's one of the few clients that gives you a list of servers and a pre-configured username based on your login name (albeit one that will change to Guest??? when you're connected to a server where that nick is already taken).

Clicking on a server will connect, and you can easily join a channel you know

or download a searchable list from the server. It's easy and works well, although we wish it cached the channel list for a while.

We really like the hierarchical view of connections over on the left. This lists the servers and channels you're connected to, and if you're connected to a few, takes up less space than a tabbed view. But you can also choose a tabbed view if that's what you prefer. The GUI is drawn using an older version of *GTK*, and this gives the application something of an old Unix feel. This isn't bad – and it also means you'll be able to use *XChat* wherever you install it, but neither is its appearance going to satisfy the eye candy brigade (if there is



XChat is a great application if you use lots of desktops and require the same interface.

one). We also miss proper desktop notifications and a system tray icon that highlights unread or missed messages. And while there is a plugin system in *XChat*, it's little more than a scripting engine.

“XChat is the first client many people go to when they start experimenting with IRC.”

VERDICT
 A good option if you've never used IRC before. It's uncomplicated, but also unimaginative.
 ★★☆☆☆

Smuxi 11.0.0

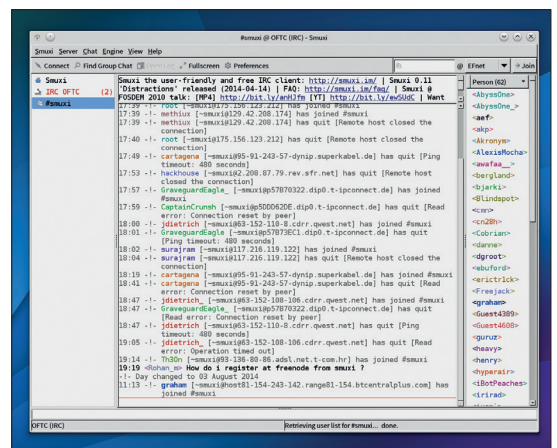
Ignore the name and there's lots to like here.

Smuxi is an unassuming IRC application that can also connect to Twitter, Facebook and several other instant messaging protocols. But this in no way diminishes its IRC credentials, unlike in *Pidgin*, for example, where its inclusion is more of a convenience. It can also be launched in console-only mode, in server mode (referred to as the engine) and with a straightforward Gnome-based GUI. It's one of the most powerful applications in this group test, while remaining easy to use.

On launch, it will helpfully connect to its own support IRC channel while also asking which server you'd like to connect to. Its interface is *XChat*-like, and you can start using IRC immediately without any further familiarisation. We like any application that includes presets for servers, as most of us will only be browsing for

groups on a small selection of well known addresses, and the 'Find Group Chat' function lets you quickly search through the channel list (and caches that list for a time), which feels very intuitive. Despite a *GTK*-based GUI that's in transition to version 3, and still looking like a throwback to the late 1990s, we love the nick colouring that keeps the same colours across channels, and it's definitely an upgrade from *XChat*. System tray notifications also work across desktops,

We also love the inclusion of a powerful filter interface that can be used to cut almost anything out of your chat windows, from 'join', 'left' and 'quit' events through to only highlighting conversations you may be interested in. It's not simple to configure, but it is powerful and it's a feature unrivalled in any of the other graphical clients we've looked at. All of which makes *Smuxi* a



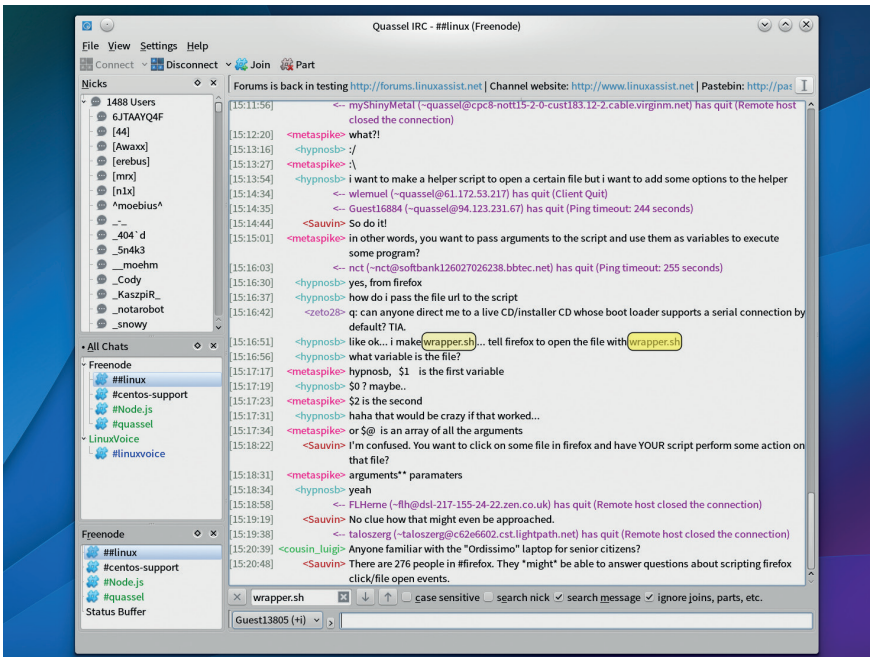
Smuxi is perfect for the power user or anyone who wants a single application for GUI, console and remote work.

brilliant option if you're not fussed about austere GUIs. It's perfect for the power user or the new user who knows they're going to need room to grow.

VERDICT
 A decent upgrade to *XChat*, and worth keeping an eye on for a *GTK3+* overhaul.
 ★★★★★

Quassel

It's powerful and good looking. Just like us.



The split between the core and the client versions of *Quassel* makes it a powerful option.

If you include *Konversation* (and *Kirc*, though we're not covering it) via KDE's dependency on *Qt*, the *Qt* toolkit is doing rather well in our group test. *Quassel* is another *Qt* application, similar to the other two but without the dependency on KDE. It uses a similar array of identities, servers and nicks to *Konversation*, which can make configuration a little tricky, but it's also easier to install and more portable. Like *XChat*, you can find *Quassel* on both Windows and OS X, as well as your favourite Linux distribution.

It's also an application that borrows its visual style from *XChat* – there's a hierarchical server and channel panel on the left, the chat window in the middle and the nick panel on the right. Any of these elements can be moved around, giving you maximum flexibility in how you like your IRC sessions organised. There's even an option to remove the input field, which could be useful if you're only monitoring a channel, although we missed the option for multi-line visualisation even when the input lines can be increased.

There are some great GUI touches. Hover over an image URL, for example, and you get an image preview. You can also configure custom chat lists, which is useful if you want to limit a list to a specific server or a specific number of

channels, and a channel will turn green when a new message is posted. Senders can have a different colour (as they can in *Konversation*) and the search highlighting is very easy to see. There are also plenty of notification options including a working event for the system tray.

But we've kept *Quassel's* best feature until last. While you can run it as a standalone application just like any other IRC client, *Quassel* also provides two split components – a core and a client, which can be run separately. It can split the core and the client components so that the core connects to your servers and channels while the client(s) provides the input and interface. This has one huge advantage – create a core user from the command line and you can run the core on a server that's always connected to your channels. Connecting from a client will then play back messages while you've been away. It's a simple way to get offline buffering of your channels, which can be essential if you use IRC for work, but it also integrates perfectly with the client.

VERDICT

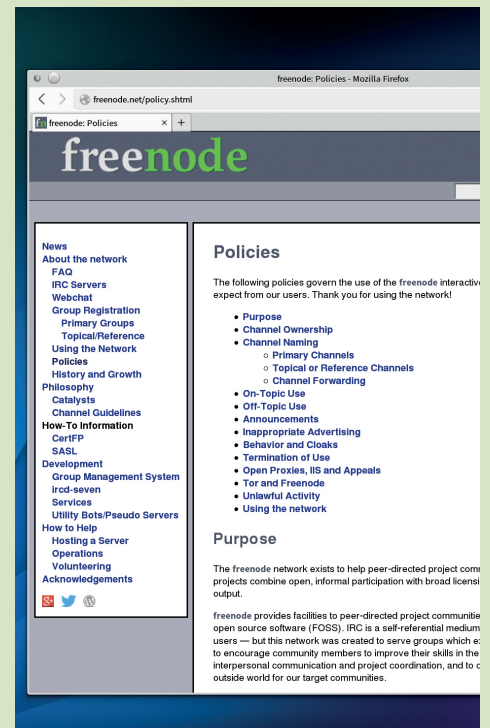
Looks fantastic, and almost matches *Konversation* for configurability.

★★★★★

IRC Commands

Our 200 word guide to interacting with IRC

There are many guides to getting started with IRC, but to help with the demystification, here's our tips to getting started. Anything you type will appear to everyone else in the channel unless it's a command preceded by the `/` character. Typing `/help` will give you some hints from the server on getting started. `/connect SERVER` will connect from the command-prompt, while `/join #CHANNEL` will join the channel. `/nick` changes your nickname, but this will need to be unique to your network. You can send a private message to someone with `/msg Mike MESSAGE`, and you can connect peer-to-peer to someone using the `/dcc` command (ie not through the server). `/dcc chat Mike` will open a chat session with Mike, for example, or we can ask to send a file to him using `/dcc send Mike file.odt`. Though not necessary, many people animate their chat with `/me (/me has another glass of wine)` will appear as **Graham has another glass of wine**, and `/describe`. Finally, if you step away from your machine for a while, use `/away` (your client may do this automatically), or `/quit` to leave.



If you start spending more time on a network, it's worth reading its policy guide.

Irssi vs WeeChat

It's the battle for the command-line!

We're about to dive into a couple of command-line clients, which means we're heading into contentious territory. Users typically invest so much time getting terminal clients exactly how they like them, and the command-line is perfect for such modification, that they become wedded to their favourite. And we're particularly fortunate because there are two awesome command-line tools that are both brilliant and probably good enough to tempt many of us away from the padded luxury of point and click.

Irssi is the one to beat. It's been around for a long time and is the default choice for many CLI users. Development has been slow over the last few years, but in June, the project moved over to GitHub in the hope of attracting a new developer community. It's not even difficult to get started with. Install the package, run **irssi** and the example config command, then connect to Freenode and join your favourite channel. Servers and networks can easily be added through further commands (type **/help** to see a list) and you can switch between them and servers and channels using Ctrl or Alt shortcuts. It's quick, powerful and easy to use. More importantly, most users download and install third-party Perl scripts to extend *Irssi* in any way

they choose, from custom highlights and showing a nick list alongside the chat view, to themes and music playback. There are already hundreds, and it's quite easy to write your own.

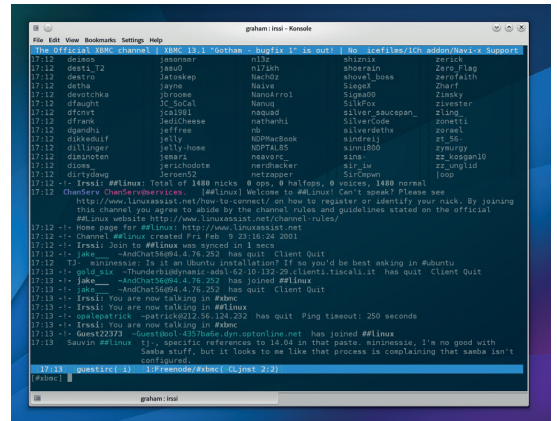
WeeChat

WeeChat has become a popular alternative to *Irssi*, as it's been able to capitalise on *Irssi*'s development doldrums over the last few years. It's what we've run on our VPS for a couple of years, and while complex to start

"Irssi has been around for a long time and is the default choice for many CLI users."

with and somewhat unforgiving, we've not found a better client.

At its core is the idea of a buffer. You can have many buffers, and each buffer can host and cache a server and session, as well as multiple sessions. You can switch between buffers and split the views horizontally or vertically many times between buffers. This means that you can configure *WeeChat* to show many channels at once, usually more efficiently than you can with a GUI application, and switch between them using the function keys.



Don't be put off my the console colours: *Irssi* is simple enough for anyone to use.

You can enable some to show the nick list, and some to not, and save multiple screen layouts and configurations with the same commands you use within the app itself. We also love the instant keybinding and the spellchecking that can highlight spelling errors on the editing line, and the Tab command completion that works for internal parameters. It all works brilliantly.

Many of the IRC clients we've looked at support scripting, but *WeeChat* has taken this to a new level. Type **/script** and the display lists hundreds of scripts that have been written and can be installed and activated in-place. Almost everything has been thought of. There's a variety of different notification systems, which is important as there's no desktop integration. You can run shell commands from within your IRC sessions, and even play *Snakes* or *Tetris*.

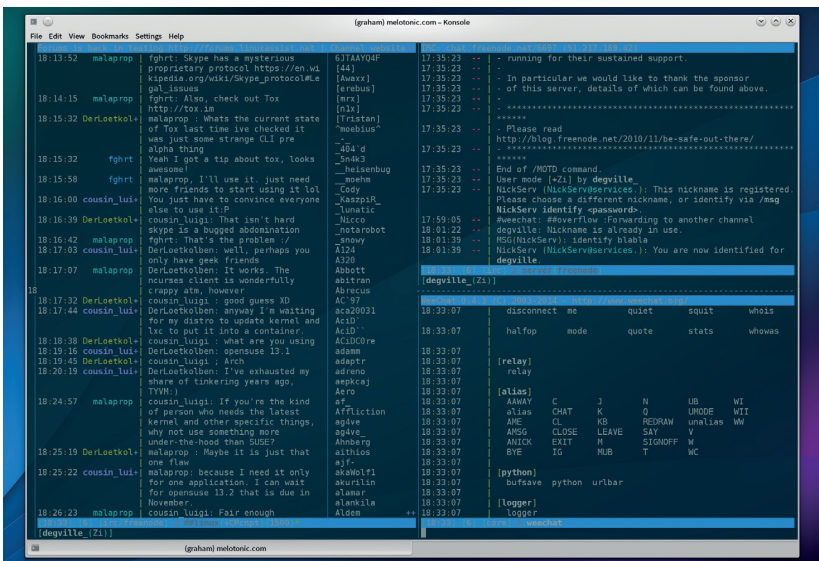
Working with both *Irssi* and *WeeChat* are a little like working with *Emacs* and *Vim* – you have to go through a considerable learning curve and use IRC regularly enough to keep what you've learnt in your local cache. But if you do, you'll find both more productive and efficient than their GUI equivalents.

These applications are always going to be a tough proposition for GUI users as beginners to IRC, but they're also a reminder of why the terminal is still so important even today, and why, in many ways, it's likely to outlast the desktop in its usefulness.

VERDICT

Irssi Smart and simple to use. If you think you might like CLI IRC, try this first... ★★★★★

WeeChat ...then if you find yourself needing more control, perhaps level-up to *WeeChat*. ★★★★★



WeeChat has some extra features, such as split views, that can become essential.

OUR VERDICT

IRC Clients

Without exception, each client we've looked at has a reason for it to be chosen as our favourite. *Quassel*, for example, has the best no-fuss separation option for client and server. It means you could run the core on a Raspberry Pi, for example, and catch up with your channels whenever you're connected. Several other clients offered similar features, but only *Quassel* combined this with what we consider a powerful GUI.

If we were to choose a GUI application, and we should to try to

activity – which is something we often want to do. The tabbed interface also makes it great for managing a large number of channel connections at once.

But we're not going with *Konversation*. We have to admit we're smitten by *WeeChat*. In our opinion, it's the Arch of IRC clients. Its forums are not friendly to newbies, and it's slightly bewildering to get started with. But we think it offers enough of an advantage on the command line that it's worth ditching the desktop for.

When you add all the advantages

“WeeChat offers enough of an advantage that it's worth ditching the desktop for.”

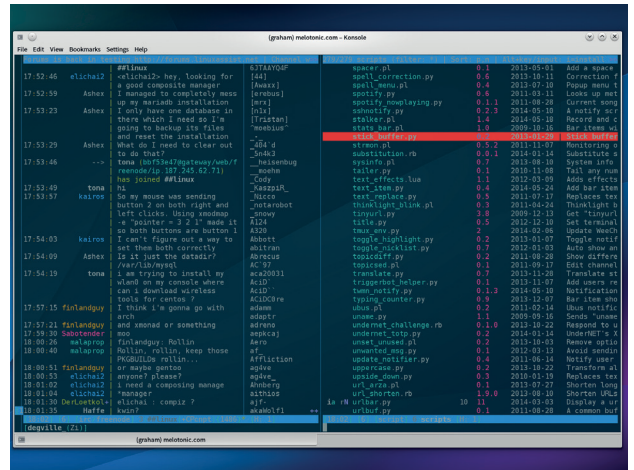
encourage new people to use IRC, we'd go with *Konversation*. Apart from a lack of scripts and extensions, we found it to be the most powerful desktop application. It did everything we asked for, and after getting our heads around the identities for networks, we found it easy to configure in even complex and bespoke IRC setups (which we use for putting the magazine together). The GUI can be subverted into almost any appearance, and there were easily accessible functions for filtering the most common chat annoyances, as well as watching nicks for

that the terminal brings for free – such as persistent *screen* sessions on a Raspberry Pi server, or low-bandwidth access from almost any SSH client, we think *WeeChat* is the best client to grow into. It's got the same feeling of liberation you get if you switch from a GUI email client to *Mutt*, or start using *Bash* more, but without sacrificing any function. Let us know if we've missed your favourite client out and we'll make sure we mention it next time. Why not let us know on our own IRC channel? You can find us as #linuxvoice on Freenode. See you there! **LV**

YOU MAY ALSO WISH TO TRY...

There are so many IRC clients, it's difficult to know where to start. You can coerce *Pidgin* into talking IRC, for example, and if you're a KDE user, *KVirc* is rather excellent. It will already be included in many KDE-centric distributions, so you won't need to do anything more to try it out. We missed this out purely

because we were already looking at two KDE-based applications and we thought three would be too many. Also worth a look is the old Mozilla client, *ChatZilla*, which works perfectly well and is very easy to use, especially alongside *Firefox*. It's still being developed and needs to be installed as a *Firefox* addon.



A splittable view, in-line spell checking, inotify and hundreds of hot-pluggable scripts – *WeeChat* is difficult to beat.

1st WeeChat

Licence GPLv3 Version 0.4.3

weechat.org

Yep, it's a terrible name. But whenever have we let that get in the way of great software?

2nd Konversation

Licence GPLv2 Version 1.5

konversation.kde.org

This is our favourite option if you're looking for the most powerful GUI client.

3rd Quassel

Licence GPLv3 Version 0.10.0

quassel-irc.org

If you want to experiment with a simple graphical client-server setup, try this option first.

4th Irssi

Licence GPL Version 0.8.16

irssi.org

If *WeeChat* is over-engineered and you need something on the terminal, *Irssi* is the best.

5th smuxi

Licence GPL Version 11.0.0

smuxi.im

It's incredible that such a brilliant app can come fourth in our list, but that's only because they're all so good.

6th XChat

Licence GPL Version 2.8.8

xchat.org

The same can be said for *XChat*. It's a great little app that works perfectly and is perhaps the best place to start with IRC.

SUBSCRIBE

shop.linuxvoice.com



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 - £57
- ROW - £60

DIGITAL
SUBSCRIPTION
ONLY £38

Get 114 pages of tutorials, features, interviews and reviews every month

Access our rapidly growing back-issues archive - all DRM-free and ready to download

Save money on the shop price and get each issue delivered to your door

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.

NEXT MONTH IN

LINUX VOICE

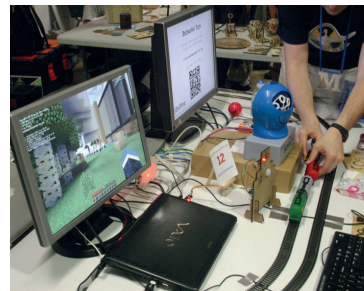
**ON SALE
THURSDAY
25 SEPTEMBER**

**BUILD YOUR OWN DISTRO**

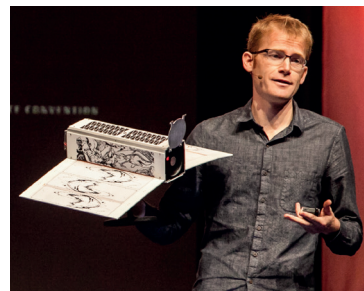
Whatever your level of technical ability, you can create a Linux distribution tailored to your exact needs. Do it – feel the power, the mastery, and build the perfect Linux for you.

EVEN MORE AWESOME!

The birth of ARM
Without Reduced Instruction Set Computing (RISC) and ARM, the smartphone enabled world would not exist. Thanks, Sophie Willson...



Get your hack on
Be inspired by the clever, creative things that people just like you are doing right now with free software. Damn, us humans are brilliant sometimes.



Space
Tiny Linux-powered satellites are in low-Earth orbit, building up an open data map of the globe. Find out who, how and why they're running Ubuntu.

LINUX VOICE IS BROUGHT TO YOU BY

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
Creative director Stacey Black
stacey@linuxvoice.com

Editorial consultant Nick Veitch
nick@linuxvoice.com

All code printed in this magazine is licensed under the GNU GPLv3

Printed in the UK by
Acorn Web Offset Ltd

Disclaimer We accept no liability for any loss of data or damage to your hardware

through the use of advice in this magazine. Experiment with Linux at your own risk! Distributed by Marketforce (UK) Ltd, Blue Fin Building, 110 Southwark Street, London, SE1 0SU
Tel: +44 (0) 20 3148 3300

Circulation Marketing by Intermedia Brand Marketing Ltd, registered office North Quay House, Sutton Harbour, Plymouth PL4 0RA
Tel: 01737 852166

Copyright Linux is a trademark of Linus Torvalds, and is used with permission. Anything in this magazine may not be reproduced without permission of the editor, until May 2015 when all content (including images) is re-licensed CC-BY-SA.
©Linux Voice Ltd 2014
ISSN 2054-3778

Subscribe: [shop.linuxvoice.com](http://shop.linuxvoice.com/subscriptions@linuxvoice.com)
subscriptions@linuxvoice.com

PCI EXPRESS LTD

LINUX BASED DIGITAL TV TUNERS



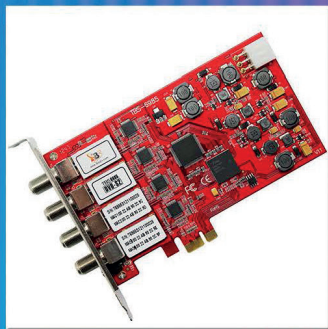
6285
DVB-T2 QUAD TUNER

£179



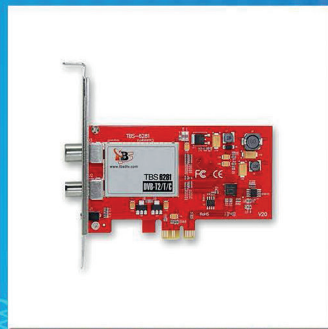
6925
DVB-S2 PROFESSIONAL

£189



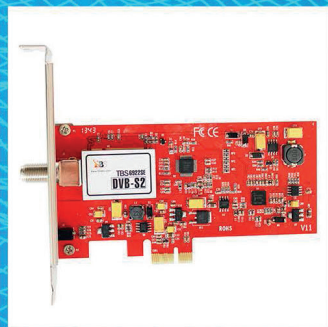
QUAD SATELLITE TV TUNER
WATCH & RECORD FTA INC. HD
WATCH 1 CHANNEL - RECORD UP TO 3 MORE

6985 £209



DUAL FREEVIEW TV TUNER
WATCH & RECORD DVB-T2
IDEAL FOR UK FREEVIEW SD & HD

6281 £109



DUAL SATELLITE TV TUNER
WATCH & RECORD FTA INC. HD
WATCH 1 CHANNEL - RECORD ANOTHER

6982 £99



Matrix



RATED
4/
5

BY YOUR VERY OWN: BEN EVERARD



xbmc

MORE INFO ON BACK PAGE

"XBMC performance is fantastic and could make the Matrix one of the best frontends you could buy"

"The hardware looks good, feels solidly made and works well"

OTHER TUNERS AVAILABLE INCLUDE
6991 DVB-S2 DUAL TUNER DUAL CI - £159
6985 DVB-S2 QUAD TUNER - £209
6928 DVB-S2 CI PCIE CARD - £99

BUY ONLINE FROM ONE OF OUR APPROVED RESELEERS

CAN ALSO BE FOUND ON **EBAY**

RED APPLE DIGITAL - REDAPPLEDIGITAL.CO.UK

CCL ONLINE - CCLONLINE.COM

VALUE AV - VALUEAV.CO.UK

QUIET PC - QUIETPC.COM

WE SUPPLY RETAIL & TRADE

TEL: 01372 377850

WWW.TBSCARDS.CO.UK

PCI EXPRESS LTD IS THE EXCLUSIVE IMPORTER AND DISTRIBUTOR IN THE UK FOR TBS
DIGITAL TUNERS AND THE MATRIX ARM MINI PC



A veteran Unix and Linux enthusiast, Chris Brown has written and delivered open source training from New Delhi to San Francisco, though not on the same day.

CORE TECHNOLOGY

Dive under the skin of your Linux system to find out what really makes it tick.

UDP: Get plugged in

Peek inside your machine to find out how it transmits data packets.

Last issue we implemented a simple server using the TCP protocol, which turned any string you typed into it into upper case letters. While the following examples should make sense on their own, we've put that article up as a PDF at www.linuxvoice.com/coretech06/ so you can read it alongside this month's.

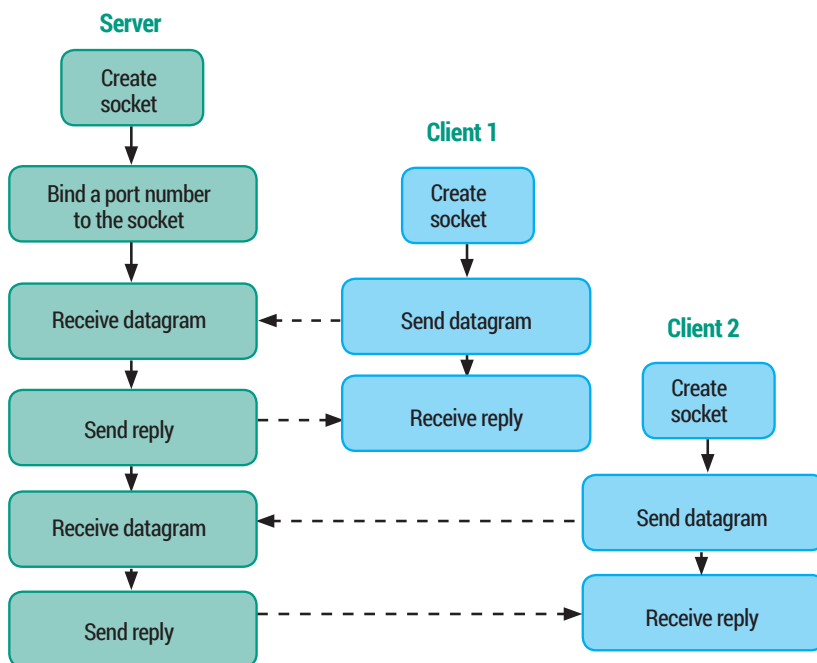
This month I want to re-cast this server to use UDP. In some ways it's simpler than TCP – there are no **connect** or **accept** operations. I've drawn a flowchart that shows the typical sequence of operations for a UDP-based service. Here we see a single server

interacting with two clients. The server has a single endpoint (UDP socket) and may well find itself retrieving datagrams from several clients in an arbitrary, interleaved order. If the server is stateless (that is, if it does not need to remember anything from one client interaction to the next) then this does not present a problem. The server simply reads a request, formulates a reply, returns it to the client, then forgets about it. Classic UDP-based services such as DNS are stateless in this sense.

Things get more complicated for servers that maintain state. One approach is to

parcel up the per-client state information into a structure, and place them into some sort of indexed data structure that uses as its search key a composite value formed from the client's IP address and port number. Another approach is for the server to create a child process for each client it finds itself dealing with. Each child can create a new UDP socket, whose port number is duly reported back to the client, and which is used by the client for the remainder of the interaction. The TFTP (Trivial File Transfer) server works this way, for example.

Peer-to-peer architecture using UDP broadcasts



A connectionless server interacts with multiple clients using a single socket.

The power of Python

Most of our code examples this month are in Python, because Python hides some of the fiddly data structures that would be exposed if we wrote them in C. So here's a UDP version of our upper-case server:

```
import socket
port = 4444
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.bind(("", port))
while 1:
    data, addr = s.recvfrom(1024)
    s.sendto(data.upper(), addr)
```

Pretty simple, huh? We create a socket and bind our "well-known" port to it. Then we enter our service loop, retrieving messages from clients, converting them to upper case, and sending them back. Last month, in showing the equivalent code for a TCP server, I briefly made the point that the server doesn't really need to know the address of the client, unless it wants to use it for logging or access control. Here it's different – we definitely need the address of the client's endpoint (**addr** in the example) so

Try It Out – Create a UDP server

To create and test the upper-case server, place its code in a file called `ucserver.py`.

To test the server, start it in one terminal window:

```
$ python ucserver.py
Now we can open a second terminal and test
server using the the jack of all trades nc (network
client) command:
$ nc -v -u localhost 4444
Connection to localhost 4444 port [udp/*] succeeded!
XXXXXThis is a test
THIS IS A TEST
```

it works!

IT WORKS!

^C

The `XXXXX` string appearing in the output above is an artifact resulting from a series of probe datagrams that `nc` apparently sends to the server (and which our server duly echoes back). The `connection succeeded` message is a little confusing; this is UDP and there is no connection as such. If you omit the `-v` (verbose) command option you won't see the `X`'s or the message. But we can

clearly see that messages we enter are returned in upper case – our server is working.

We can extend the experiment. Leave the `nc` program running in the second terminal window, open a third terminal window and run the same `nc` command there as well. You should find that you can interact with the server via both windows. That's the simplicity of a connectionless service; you don't need any multi-processing or multi-threading or other fancy tricks in the server to get concurrent operation with multiple clients.

that we know where to send the reply. Python's dynamic typing is hiding a little complexity here, because `addr` is actually a (host, port) pair. See the box above if you would like to build and test this server. We could write a little Python program to act as a client to our upper-case server, but let's switch to a different example. There is an ancient UDP-based service called `daytime`, which listens on port 13 and simply sends back a string with the current time and date. It's a sort of speaking clock but without the speaking. This service is implemented by a daemon called `xinetd`; the box below shows how to install and enable it.

Once the `daytime` service is up and running, we can write a client for it. Again using Python, it looks like this:

```
#!/usr/bin/python
# UDP daytime client
import sys
import socket
# Get server host name from command line
host = sys.argv[1]
port = 13
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
# Send an empty datagram to wake the server up
s.sendto("", (host, port))
data, addr = s.recvfrom(1024)
print "time from", addr, " is ", data
And we can run it like this:
$ chmod u+x daytimeclient.py
$ ./daytimeclient localhost
time from ('127.0.0.1', 13) is 01 JUL 2014 10:26:42
BST
```

Notice how Python automatically converts the client address into a printable (host, port) representation.

Broadcasting

One thing that you can do with UDP sockets that you cannot do with TCP is broadcasting. That is, you can send a single copy of a message and have it received by many listeners. The constraints are that all the listeners must be using the same port,

and they must all be on the same network, because routers and gateways are almost never configured to pass broadcast traffic. There isn't a lot to it really; you have to explicitly enable broadcasting on the socket, and use a special destination host address of "all ones", or 255.255.255.255 in dotted decimal notation.

The example presented here is both a client and server all rolled into one. The idea is that the client piece periodically generates an item of data, which it broadcasts. The server piece receives the broadcasts and displays the item of data. For simplicity the 'item of data' is a simple randomly generated integer, but could be something more interesting in the real world – a weather forecast or a stock price, perhaps.

The code here is in C, and it looks more complicated than the Python examples we've seen so far, but conceptually it's not really any harder. As always, the line numbers are for reference; they are not part of the code:

```
1 #include <stdlib.h>
2 #include <netdb.h>
3 #include <stdio.h>
4 #include <arpa/inet.h>
5
6 #define UPDATE_PORT 2066
7
8 void main ()
9 {
```

```
10 int sock; /* Socket descriptor */
11 struct sockaddr_in server; /* Broadcast address */
12 struct sockaddr_in client;
13 int client_len, yes = 1;
14 int value;
15
16 /* Create a datagram socket and enable
broadcasting */
17 sock = socket (AF_INET, SOCK_DGRAM, 0);
18 setsockopt (sock, SOL_SOCKET, SO_BROADCAST,
(char *) &yes, sizeof yes);
19
20 /* Bind our well-known port number */
21 server.sin_family = AF_INET;
22 server.sin_addr.s_addr = htonl (INADDR_ANY);
23 server.sin_port = htons (UPDATE_PORT);
24 bind (sock, (struct sockaddr *) &server, sizeof
server);
25
26 server.sin_family = AF_INET;
27 server.sin_addr.s_addr = 0xffffffff;
28 server.sin_port = htons (UPDATE_PORT);
29
30 /* Create an additional process. The parent acts
as the client,
31 periodically broadcasting values to anyone who
happens to be
32 listening on port 2066. The child acts as the
server,
33 receiving the broadcasts and displaying the
data.
34 */
```

Try It Out – Install the daytime service

To get the `daytime` service running we first need to install `xinetd` (this is on Ubuntu, but the story should be similar on other distros):

```
$ sudo apt-get install xinetd
Even if xinetd were already installed, the
daytime service is probably disabled. So, edit
the file /etc/xinetd.d/daytime, find the stanza
that relates to the UDP version of the service and
change the line disable = yes to read disable = no.
Now restart xinetd:
$ sudo invoke-rc.d xinetd reload
(On a Red Hat-style system you would need
```

service `xinetd` restart instead.) Now verify that the `daytime` server is listening:

```
$ sudo lsof -i | grep daytime
xinetd 27465 root 5u IPv4 165011 0t0 UDP
*:daytime
```

If you don't see an encouraging line of output here, you'll need to investigate before moving forwards. We can test this service using `nc` again:

```
$ nc -u localhost daytime
01 JUL 2014 10:16:20 BST
```

You will need to send the `daytime` server a datagram of some sort (just enter a blank line).



The Berkeley sockets library, dating from 1983, remains the standard sockets API to this day.

```

35 if (fork ())
36 { /* PARENT (client) here */
37 while (1)
38 {
39 value = rand () % 1000;
40 /* Broadcast update packet to servers */
41 sendto (sock, (char *) &value, sizeof value, 0,
42 (struct sockaddr *) &server, sizeof
server);
43 sleep (1);
44 }
45 } /* End of parent (client) code */
46
47 /* -----
*/
48
49 else
50 { /* CHILD (server) here */
51 /* Enter service loop, receiving values and
displaying them */
52 while (1)
53 {
54 /* Receive an update packet */
55 client_len = sizeof client;
56 recvfrom (sock, (char *) &value, sizeof value,
0,
57 (struct sockaddr *) &client, &client_len);
58
59 /* Display the broadcast value and where it
came from */
60 printf ("got %3d from %s\n", value, inet_ntoa
(client.sin_addr));
61 }
62 } /* End of child (server) code */
63 }
    
```

Now there's quite a bit of code here, and some of it is messy. So grab a brown paper bag (so that you can breathe into it for a bit if you start to panic) and let's work through it. First, the declarations at lines 11 and 12 refer to the endpoint addresses used for sending and receiving. (The name `sockaddr_in` means 'internet socket

address'. When I first met this years ago I thought that the "in" meant "input", and spent some time looking for a `sockaddr_out`, which my sense of symmetry told me must be there, like the Higgs Boson. But I digress.)

At line 17 we create our socket and at line 18 we set the `SO_BROADCAST` option on it. Just look at the hoops we have to jump through to pass in a Boolean TRUE value.

Lines 20–24 bind our chosen port number (2066) to the socket. At lines 26–28 we re-use the 'server' structure to hold the broadcast address. Notice the `0xffffffff` value, which is the "all ones" of the broadcast address.

Now we get cunning, rolling the client and server pieces of the application into one program by creating another process. The parent process (lines 37–45) is the client. Once a second, it generates a random integer value, and broadcasts it in a tiny 4-byte datagram. The child process (lines 52–62) is the server. It receives the broadcast packets and prints out each value, along with the IP address of the client that sent it. The important thing to keep in mind here is that this loop is not only receiving the broadcasts from its own client, it will also receive the broadcasts from all other instances of the client running elsewhere on the network.

Raw sockets

I won't inflict any more code on you this month, but I wanted to wrap up by mentioning two more socket types. First,

Get the code

A tarball of the programs used in this tutorial can be downloaded from www.linuxvoice.com/mag_code/lv07/coretech007.tar.

raw sockets enable an application program to reach right down to the IP layer and 'hand-craft' the headers of whatever the overlying protocol is. For example, the port scanner *Nmap* uses raw sockets to build non-conformant TCP headers for its own special purposes. As another example, `ping` (which sends and receives ICMP packets) also uses raw sockets. On Linux, the rule is that only processes running with root privilege can use raw sockets. This is a security precaution, because a program using raw sockets can intercept all traffic entering the system. A common way to deal with this is to have the program run "set UID to root". This enables it to create its raw socket, then drop its privilege back to a non-root user. If you look at the `ping` program for example, you'll find it runs `setuid` for this reason:

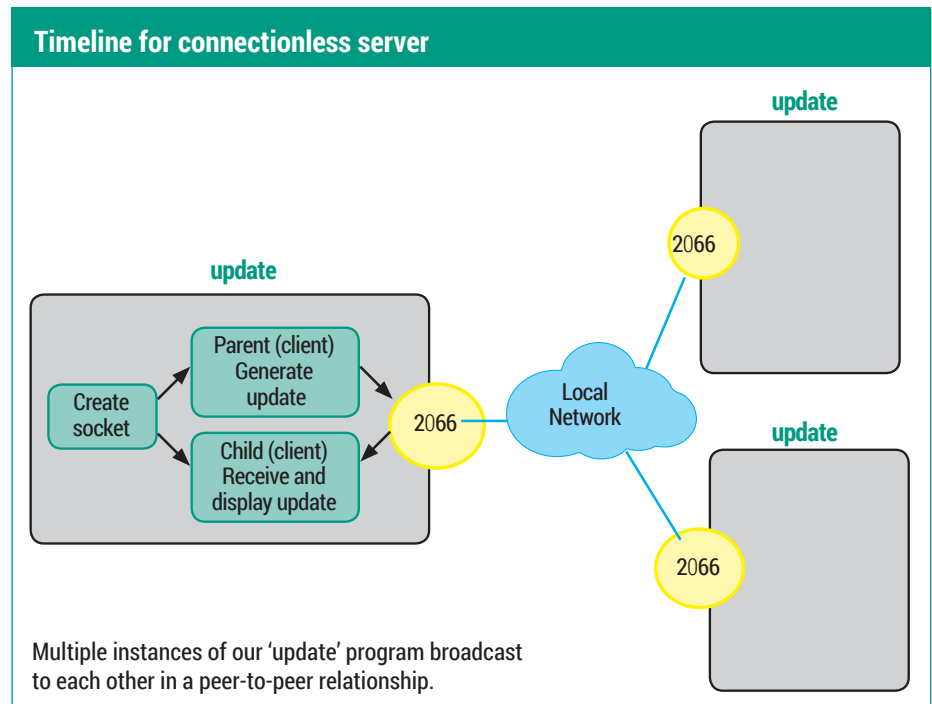
```

$ ls -l /bin/ping
-rwsr-xr-x 1 root root 44168 May 7 22:51 /bin/ping
    
```

Notice the `s` in the permissions.

Staying in the Unix domain

We've focussed here on sockets in the internet domain, which means that (among other things) the socket is identified by an IP address and a port number. But there are other naming domains for sockets; in



particular, the so-called “Unix domain sockets” are identified by a name within the filesystem. You can find all of these with the command:

```
$ sudo find / -type s
```

A classic example is `/dev/log`, which **syslog** (or **rsyslog**) uses to collect log messages from local applications. But you will probably find many others.

There's no command for creating a named socket analogous to **mkfifo** for creating named pipes. Your server creates the socket and binds the name to it, and your client needs to know that name in order to connect. Unix domain sockets only support communication between processes

Try It Out – Build a peer-to-peer update service

If necessary, install the gcc compiler:

```
$ sudo apt-get install gcc
```

Enter the code into a file called `update.c` and compile it:

```
$ gcc update.c -o update
```

Run it like this:

```
$ ./update
```

```
got 383 from 192.168.1.69
```


```
got 649 from 192.168.1.73
```

```
got 886 from 192.168.1.69
```

```
got 421 from 192.168.1.73
```

To do a meaningful test you'll need to copy the executable across onto at least one other machine on the same network, and run it there as well. (You can't run multiple instances on the same machine -- why not?) In the output above, you'll see that we're receiving interleaved broadcasts from two machines.

running on the same machine. There are also anonymous Unix domain sockets created with the **socketpair()** system call; these are somewhat similar to good ol'anonymous pipes (see Core Technologies

in LV005) but unlike pipes, which are unidirectional, a socket pair is bidirectional. Also, you can create both stream and datagram socketpairs, whereas pipes are inherently stream-oriented. 

Command of the month: dig

My command of the month is **dig**.

According to its man page it stands for “domain information groper”, though that sounds like a retrofitted acronym if ever I heard one! Anyway, **dig** is a command-line tool for performing DNS queries.

In my view, **dig** has two main uses. First, you can use it to test your DNS service. Second, you can use it as an exploration tool. It's this second use we'll focus on here. We'll use *Linux Voice's* own site as a target for our exploration. First let's just find the IP address of the website; this is the simplest type of lookup:

```
$ dig www.linuxvoice.com
```

```
;; QUESTION SECTION:
```

```
www.linuxvoice.com.      IN      A
```

```
;; ANSWER SECTION:
```

```
www.linuxvoice.com. 600 IN CNAME linuxvoice.com.
```

```
linuxvoice.com. 600 IN A 213.138.101.172
```

I've edited a lot of detail from this output but you'll see it shows that we requested an 'A' record from DNS for the name **www.linuxvoice.com**. ('A' records are the records in DNS that map machine names to IPV4 addresses.) What we actually got was a CNAME record (an alias, in effect) pointing to the name **linuxvoice.com**. **Dig** then kindly looked up the A record for **linuxvoice.com**, finally reporting the IP address.

There are command-line options for **dig** that control how much output we see. For example, **+noquestion** suppresses the **question** section from the output. You can turn off other output sections using options such as **+nocomments**, **+noauthority**, **+noadditional** and **+noanswer**, or you can turn everything off using **+noall** and then

explicitly enable the sections you want to see. For example, this shows just the **ANSWER** section:

```
$ dig ubuntu.com +noall +answer
```

```
; <<>> DiG 9.9.5-3-Ubuntu <<>> ubuntu.com +noall +answer
```

```
;; global options: +cmd
```

```
ubuntu.com. 577 IN A 91.189.94.156
```

If there are options you always want to specify, just put them into `~/digrc`. For example, if you put this line into the file:

```
+noall +answer
```

then by default your **dig** queries will only show the **ANSWER** section.

The **+short** option really cuts to the chase and shows just a bare-bones response:

```
$ dig +short linuxvoice.com
```

```
213.138.101.172
```

Dig deeper

Next let's investigate who handles mail for the **linuxvoice.com** domain. For that we need to get the MX (Mail Exchanger) record:

```
$ dig +short linuxvoice.com mx
```

```
10 smtp.linuxvoice.com.
```

```
$ dig +short smtp.linuxvoice.com
```

```
213.138.101.172
```

So... mail is handled by a machine called **smtp.linuxvoice.com**, and it turns out that this is the same machine (same IP address) as the web server. So, Linux Voice apparently hosts its own web and mail servers on a single machine. No surprises there.

Let's try a reverse lookup on that machine. That is, let's look up the PTR record for that IP address and convert it back to a machine name. The PTR records are stored under the **in-addr.arpa** domain, and because DNS names are written in a “little endian” form,

we end up with the four octets of the IP address reversed. So here's the hard way to do the lookup:

```
$ dig +short 172.101.138.213.in-addr.arpa ptr
```

```
mainsite.default.linuxvoice.uk0.bigv.io.
```

An easier way is to use the **-x** option of **dig**, which lets us enter the IP address in the usual format:

```
$ dig +short -x 213.138.101.172
```

```
mainsite.default.linuxvoice.uk0.bigv.io.
```

What's interesting is that the IP address is allocated to a machine in the **bigv.io** domain. A quick search reveals that BigV is a virtual machine hosting provider – now we know who Linux Voice uses to host its site.

A handy tool for stalkers

Let's try something a little different, by asking DNS where its root name servers are:

```
$ dig +short . ns
```

```
f.root-servers.net.
```

```
i.root-servers.net.
```

```
d.root-servers.net.
```

Here, **ns** means we're looking for name server records and **.** refers to the top level domain. It is analogous to **/** in a filename, which names the root directory (the top-level directory) in a filesystem. I've cut the output down again; there are actually 13 root name servers, I've shown only three.

By default, **dig** consults the file `/etc/resolv.conf` to figure out which name server to consult, just as normal DNS lookups do. But we can direct **dig** to a specific DNS server like this:

```
$ dig @8.8.8.8 +short jamieoliver.com
```

```
85.233.160.22
```

Here, 8.8.8.8 is the IP address of Google's public DNS service.

FOSSpicks

Sparkling gems and new releases from the world of Free and Open Source Software



Mike Saunders has spent a decade mining the internet for free software treasures. Here's the result of his latest haul...

Shiny statistics in a browser

Web VMStat

Many distros, especially those targeted at advanced users, ship with shiny system monitoring tools on the desktop. *Conky* is one such tool, while *GKrellM* was all the rage in the last decade, and they are genuinely useful for keeping tabs on your boxes, especially when you're an admin in charge of various servers.

Now, pretty much all major distros include a useful command line tool for monitoring system resource usage: **vmstat**. Enter **vmstat 1** in a terminal window and you'll see a regularly updating (once per second) bunch of statistics, showing CPU usage, free RAM, swap usage and so forth. It's all very useful, but it has one major problem: it's ugly. Very ugly. Sure, most admins don't care about fancy bells and whistles, but the information could be presented in a more readable and clean fashion.

Here's where *Web VMStat* comes in. It's a system monitor that runs an HTTP server, so you can connect to it via a web browser and see fancy CSS-driven charts. Before you install it, you'll need to get the *websocketd* utility, which you can find at <https://github.com/joewalnes/websocketd>. Helpfully, the developer has made pre-compiled executables available, so you can just grab the 32-bit or 64-bit tarball, extract it and there you have it: *websocketd*. (Of course, if you're especially security conscious, you can compile it from its source code.)

Next, clone the *Web VMStat* Git repository (or grab the Zip file and extract it). Go into the directory and

“Web VMStat is a way to keep an eye on servers on your network without having to log into them.”

```
mike@debianmike: ~/web-vmstats-master
File Edit Tabs Help
mike@debianmike:~/web-vmstats-master$ vmstat 1
procs -----memory----- --swap-- -----io----- -system- - -cpu-----
r b swpd free buff cache si so bi bo in cs us sy id wa
0 0 0 2687984 24884 207084 0 0 84 20 343 585 14 11 75 0
0 0 0 2687976 24884 207084 0 0 0 0 0 254 511 1 1 98 0
0 0 0 2687976 24884 207084 0 0 0 0 0 390 776 0 5 95 0
0 0 0 2687976 24884 207084 0 0 0 0 0 347 636 1 3 96 0
0 0 0 2684844 24884 207084 0 0 0 0 0 576 1614 5 6 89 0
0 0 0 2684892 24892 207076 0 0 0 0 0 32 299 600 0 2 98 0
1 1 0 2669880 29456 207196 0 0 4676 0 2065 12310 3 23 63 12
3 1 0 2652396 36760 207196 0 0 7300 0 3452 19744 5 42 32 20
2 0 0 2634912 43792 207248 0 0 7032 0 3386 19646 9 38 32 21
3 0 0 2615072 49944 207296 0 0 6152 0 3367 20126 6 40 33 21
0 0 0 2609616 54904 207276 0 0 4360 92 2561 14214 4 25 55 15
1 0 0 2609376 54320 207176 0 0 0 0 4084 333 632 0 2 97 1
1 0 0 2609344 54320 207192 0 0 0 0 207 406 1 1 98 0
2 0 0 2609344 54320 207192 0 0 0 0 225 436 0 3 98 0
1 0 0 2609352 54320 207192 0 0 0 0 230 450 0 2 98 0
0 0 0 2609352 54320 207192 0 0 0 0 4052 410 445 0 2 98 0
0 0 0 2609352 54320 207192 0 0 0 0 210 411 0 1 99 0
0 0 0 2609352 54320 207192 0 0 0 0 245 481 1 2 98 0
0 0 0 2609352 54320 207192 0 0 0 0 202 395 0 1 99 0
0 0 0 2609416 54320 207192 0 0 0 0 237 464 0 3 98 0
0 0 0 2609416 54328 207184 0 0 0 0 20 236 459 0 2 97 1
0 0 0 2609416 54328 207192 0 0 0 0 224 440 1 2 98 0
```

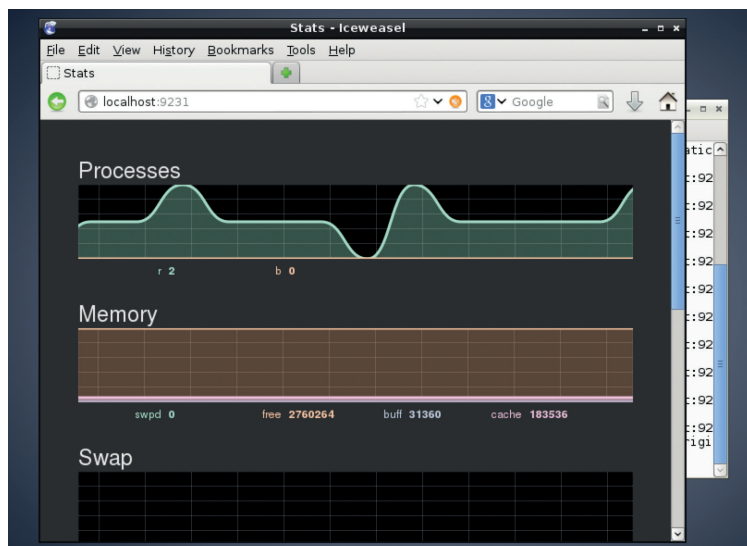
Here's the standard output for **vmstat** – not very interesting, right?

copy the aforementioned *websocketd* into the same place. Then just enter:

```
./run
```

And that's it – *Web VMStat* has started an HTTP server on port 9231, so you can access that in your browser (eg <http://localhost:9231>). Straight away, you'll see that the charts are smooth and silky, with processes, CPU, memory, IO and swap usage depicted. Hover your mouse over the numbers for more detailed descriptions.

In all, *Web VMStat* is a simple way to keep an eye on servers on your network without having to log in to them. You could create a bookmark group for a bunch of machines, for instance, and open them up in tabs for a quick glance of how they're performing. And all the information is gathered by the 'real' **vmstat** tool, so you know it's legit.



With *Web VMStat*, resource usage is shown in a much clearer and prettier manager. Look at the smooth curves!

PROJECT WEBSITE
<https://github.com/joewalnes/web-vmstats>

Operating system

NetBSD 6.1.4

Back on page 38, our FAQ looked at the BSD family of operating systems, so hopefully that has whetted your appetite sufficiently to try one. We thought we'd look at NetBSD here. It's the most ported of the BSDs, running on over 50 hardware platforms (www.netbsd.org/ports). "Big wow", you might say. "Linux runs on everything from supercomputers to wristwatches".

True – but many of these ports are in unofficial source code branches, not always remaining up-to-date with the mainline kernel. In NetBSD, everything is built from the same source tree, and this often helps with the overall stability and security of the OS.

Anyway, NetBSD for x86 and x86-64 is available in CD ISO (eg [NetBSD-6.1.4-i386.iso](#)) and USB flash drive ([NetBSD-6.1.4-i386-install.img.gz](#)) formats. For the former you can write it to a CD-R and boot it on a real machine, or try it in *VirtualBox*. The latter can be written to a flash drive with the usual **dd** command (see <http://tinyurl.com/bsdusbininstall>).

When you boot it up, you'll see a text mode installer akin to those of Debian and Slackware. It's menu driven, so it's a bit simpler to navigate than the installer in OpenBSD, and experienced Linux users won't have major troubles

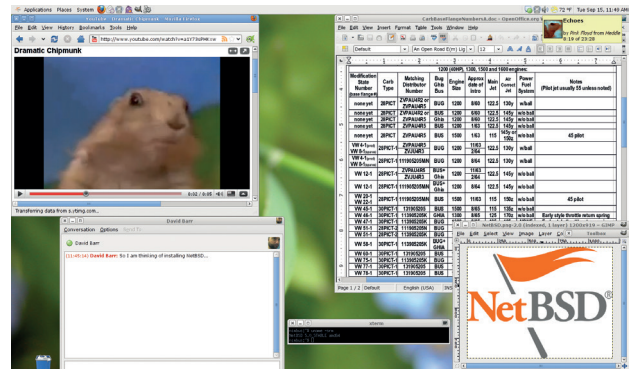
understanding it. Some terminology in the BSD world is considerably different though: for instance, hard drive devices tend to be called **wd0** (for the first), **wd1** (for the second) and so forth.

Also, NetBSD installations normally use one large primary partition on the hard drive, which is then split up into sub-partitions for root (**/**), the home directories, swap space and so forth. The installer provides plenty of help about this, but if you're unsure, run through the installation in a virtual machine before trying it on a real box!

Bare-bones setup

After the first boot from the hard drive, you can log in as root with no password. NetBSD is in a very bare state here, but fortunately there's an excellent manual page describing the next steps to take. Enter **man afterboot** and you'll see a helpful guide to changing the root password, setting up a normal user account, configuring the network (enabling DHCP on boot) and so on. Documentation in the BSD family is largely superb, and if the afterboot page doesn't help you with

"In NetBSD everything is built from the same source tree, which helps with stability and security."



Although NetBSD's base system is very minimal, you can spruce it up with most common desktop apps included in Linux distributions.

something, it's almost certainly covered in the extensive guide at www.netbsd.org/docs/guide/en/index.html.

Try entering **startx** to fire up the X Window System; if that fails, these commands (as root) should help by providing a fresh X configuration:

X -configure

```
mv xorg.conf.new /etc/X11/xorg.conf
```

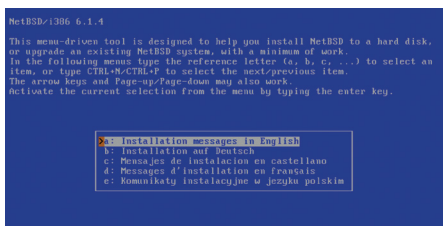
After running **startx** now, you'll land at a completely unremarkable TWM desktop. NetBSD doesn't try to second guess what you want, however, so you can start adding applications like so:

```
export PKG_PATH="http://ftp.netbsd.org/pub/pkgsrc/packages/NetBSD/i386/6.1.4/All/"
pkg_add -v xfce4 firefox24
```

Again, check out the superb online guide for more information.

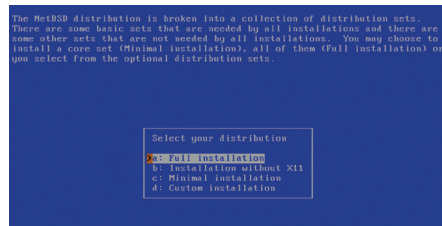
PROJECT WEBSITE
www.netbsd.org

How it works: Installing NetBSD



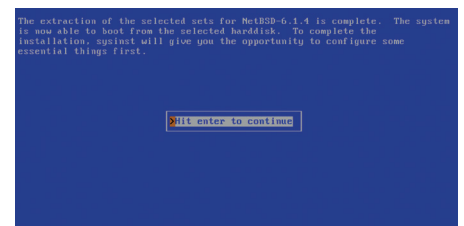
1 Boot

Boot your PC or VM from the CD or USB key, and a bunch of green NetBSD kernel messages will whizz by. After a few moments, you'll arrive at the installer; use the cursor keys and Enter to navigate.



2 Choose sets

You'll be asked which 'sets' should be installed. These are groups of software, such as development tools and the X Window System – in most cases, it's best to choose Full Installation.



3 Finish

After you've partitioned the drive (you can give NetBSD the whole disk for simplicity's sake), the OS files will be copied over, and you can reboot into your shiny new installation.

Prettified classic window manager

FVWM-Crystal 3.3.2

Rant mode activated: there's a lot of NIH (Not Invented Here) syndrome in the software development world. Too many programmers would rather write something from scratch just to say they've done it, than use or improve a mature and existing project. Nobody has the right to dictate how FOSS developers spend their time, but we like to see people working more constructively, reducing duplicated effort.

So we love *FVWM-Crystal*: it's a gorgeous, shiny and functional desktop that hasn't written everything from the ground up. No, instead it uses *FVWM*, one of the oldest (and ugliest) window managers in existence.

FVWM-Crystal takes this venerable WM and adds layers of polish. When you start it, you'll notice (in the default configuration)

that there's a program launcher bar in the top-left, workspace switcher in the top-middle, and taskbar along the bottom. Right-click on the desktop and a terminal window will appear. The program launcher has a series of icons that open submenus for different categories of applications, while the crystal button in the far top-left has a menu for tweaking *FVWM*'s settings.

If you're used to graphical dialogs for configuring every part of your desktop, you might find *FVWM-Crystal* somewhat limiting – you have to spend a lot of time poking around in text files. On the other hand, this makes it somewhat easier to back up and move your configuration across multiple machines, so once you've fine-tuned everything to perfection, you can keep your desktop pixel-perfect regardless of your distro.



FVWM-Crystal takes the ultra configurability of *FVWM* and puts a shiny layer on top.

Much of the documentation for the *FVWM* applies to *FVWM-Crystal*, and on the *FVWM-Crystal* website you'll see that the Documentation tab holds some resources such as an FAQ and tips page. (If you want to see what a standard *FVWM* setup looks like, just choose it from your login screen, it will be installed via the *FVWM-Crystal* package.)

PROJECT WEBSITE
<http://fvwm-crystal.sourceforge.net>

Command progress viewer

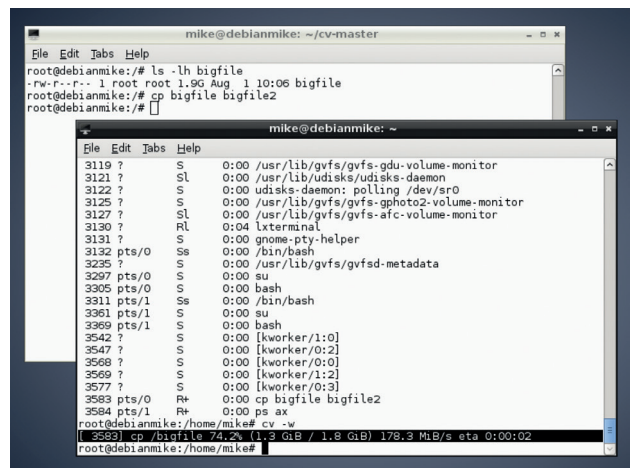
Cv 0.4

You might not have heard of *Coreutils* before, but it's an essential part of every major Linux distribution. It's a software bundle from the GNU project that provides all the little tools you use at the command line – *ls*, *rm*, *cp* and so forth. While the GNU *Coreutils* programs are regarded as the most featureful in the Unix world, they still have some limitations. It's not easy to see the progress of some commands, for instance, which can be annoying when you're performing a large file operation and want to know how much time is remaining. *Cv*, the Coreutils Viewer, is a tiny program that fixes this, showing statistics for running commands.

To install it, clone the Git repository (or just download the Zip file from the project's website) and

run **make** and **make install** (the latter as root) inside the directory. Now run a few commands that will take a while to execute, such as copying a multi-gigabyte file. In another terminal, run **cv** and you'll see the PID (process ID) of the command, along with a percentage value showing how close it is to completion.

Cv can probe running instances of these tools: *cp*, *mv*, *dd*, *tar*, *g(un)zip*, *cat*, *grep*, *cut* and *sort*. If you add the **-w** flag to the **cv** command, it will try to work out the I/O throughput of the file operation and show an estimated time for completion – but, of course, this



Copying a file in one terminal and seeing its progress with *Cv* in another – we see that it's 74.2% complete.

can vary depending on your system load. Another useful flag is **-m**, which runs **cv** in monitoring mode; this shows updated statistics every second, until the file operation completes.

PROJECT WEBSITE
<https://github.com/Xfennec/cv>

“Cv is a tiny program that shows statistics for running commands.”

Interactive filtering

Percol

Last issue, we looked at a classic tome called *The Unix Programming Environment*. At the start of this book, the authors outline the Unix philosophy, stating: “Many Unix programs do quite trivial things in isolation, but, combined with other programs, become general and useful tools.”

Percol adheres to this philosophy – it doesn’t appear to do anything useful on its own, but combined with other tools it turns out to be rather useful. It’s essentially an interactive filter for text, so you pipe some data into it, type some letters to narrow down the selection, and it spits out the resulting selection to **stdout**. You can install it with **pip install percol**, or if you grab the code from GitHub, **sudo python setup.py install**.

To get a feel for how *Percol* works, use it to view a text file, eg:

```
percol /boot/grub/grub.cfg
```

This will show the entire contents of the file, with a prompt at the top. Start typing some characters, though, and you’ll see that the display is narrowed down to lines containing the text that you’ve entered. Use the cursor keys to select a line, hit Enter, and you’ll see that the line’s text is printed at the shell prompt.

By combining *Percol* with other commands, we can provide a level of interactivity. Look at this:

```
ps aux | percol | awk '{ print $2}' | xargs kill
```

Here we’re generating a list of processes with the **ps** command, and piping them into *Percol* (as in the screenshot). In the **percol** part,

```

mike@debianmike: ~
QUERY > k
root      2  0.0  0.0  0  0 ?    S   13:02   0:00 [kthread]
root      3  0.0  0.0  0  0 ?    S   13:02   0:00 [ksfttrqd/0]
root     10  0.0  0.0  0  0 ?    S   13:02   0:00 [ksfttrqd/1]
root     11  0.0  0.0  0  0 ?    S   13:02   0:00 [kworker/0:1]
root     14  0.0  0.0  0  0 ?    S<  13:02   0:00 [khelper]
root     15  0.0  0.0  0  0 ?    S   13:02   0:00 [kdevtmpfs]
root     19  0.0  0.0  0  0 ?    S<  13:02   0:00 [kintegrityd]
root     20  0.0  0.0  0  0 ?    S<  13:02   0:00 [kblockd]
root     21  0.0  0.0  0  0 ?    S   13:02   0:00 [kworker/1:1]
root     22  0.0  0.0  0  0 ?    S   13:02   0:00 [khungtaskd]
root     23  0.0  0.0  0  0 ?    S   13:02   0:00 [kswapd0]
root     24  0.0  0.0  0  0 ?    SN  13:02   0:00 [ksmd]
root     25  0.0  0.0  0  0 ?    SN  13:02   0:00 [khugepaged]
root     26  0.0  0.0  0  0 ?    S   13:02   0:00 [fsnotify_mark]
root    106  0.0  0.0  0  0 ?    S   13:02   0:00 [khubd]
root    132  0.0  0.0  0  0 ?    S   13:02   0:00 [kworker/u:1]
root    135  0.0  0.0  0  0 ?    S   13:02   0:00 [kworker/u:2]
root    141  0.0  0.0  0  0 ?    S   13:02   0:00 [kworker/1:2]
root    436  0.0  0.0  0  0 ?    S<  13:03   0:00 [kpsmouse]
root    439  0.0  0.0  0  0 ?    S   13:03   0:00 [kworker/0:3]
avahi    2380  0.0  0.0  3156 1500 ?    S   13:03   0:00 avahi-daemon: r
root    2491  0.0  0.0  0  0 ?    S<  13:03   0:00 [krfcomm]
root    2661  0.0  0.1  27508 4868 ?    Ssl 13:03   0:00 /usr/sbin/Netw

```

Here’s the output of **ps aux** being piped through *Percol* – we’ve entered **k** here to narrow the list down a bit.

we can type the name of a process and use the cursor keys to select a specific line. After you hit Enter, **awk** pulls out the PID (process ID) part of the text, and passes it on to the **kill** command via **xargs**. So ultimately you have an interactive process killer, without any coding – just by linking tools together. If you do a lot of *Bash* scripting, you’ll find plenty of ways to add interactivity to your scripts with this.

“Combined with other tools, Percol turns out to be rather useful.”

PROJECT WEBSITE
<https://github.com/mooz/percol>

Python web framework

Bottle 0.12

If you’re new to web development and want to give it a try without all the layers of complexity – then *Bottle* is a sound choice. It’s a “micro web framework”, helping you to create small web apps with relatively few lines of code.

To install it, just run **pip install bottle** and Python’s built-in package manager will retrieve the code. Failing that, you can grab the source code from <https://pypi.python.org/pypi/bottle>. *Bottle* has no dependencies other than the standard libraries that ship with Python, so you don’t need to bloat your system with piles of extra cruft. Once you have it installed, test it out with this Python script:

```

from bottle import route, run, template
@route('/hello/<name>')
def index(name):

```

```

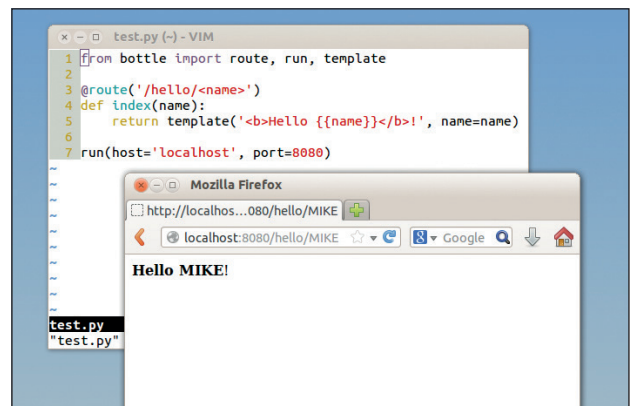
    return template('<b>Hello {{name}}</b>!',
name=name)
run(host='localhost', port=8080)

```

Now visit <http://localhost:8080/hello/you> in your browser, and you’ll see a message. There you have it – a web application in just 5 lines of code! OK, so this is totally trivial right now, but it shows you how to get started.

Bottle has a built-in template engine and can also use *Mako*, *Jinja2* and *Cheetah* templates. It provides easy ways to access and manipulate data from forms, file uploads, cookies and HTTP headers. And along with the supplied web server, it can also use other WSGI-capable servers such as *Python Paste*, *Bjoern* and *Google App Engine*.

Of course, one of the most important features of any



Web application frameworks – such as *Django* and *Zope* – can be daunting for beginners, but *Bottle* is a friendly way to get started.

framework is the documentation, and *Bottle* does a very respectable job here. The PDF at <http://bottlepy.org/docs/dev/bottle-docs.pdf> includes tutorials and reference guides, with plenty of sample code, and the text is clear and well-written. Even though *Bottle* is only at version 0.12/13, it’s showing a lot of potential.

PROJECT WEBSITE
www.bottlepy.org

Notification daemon

Dunst 1.0.0

Most desktops (and many window managers) include a notification system, allowing applications and background daemons to pop up important information. The system might show you when your laptop battery is getting low, for example, or when someone comes online in your instant messaging service.

Now, if you're happy with your desktop's notification system, great. But if not, or you're interested in dabbling in a tiling window manager (as covered in last issue's group test), then you'll need an alternative. *Dunst* is one of the best we've seen, due to its low resource requirements and customisability.

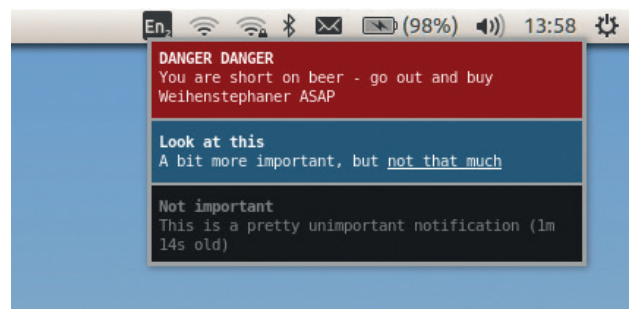
Extract the tarball, check out the **INSTALL** file for the list of dependencies, and then run **make**. All being well, you can then run **./dunst** to start the notification

daemon – or if you have one already running, get its PID with **ps aux** first, and then kill it. Now you can create notifications like so:

```
notify-send -t 0 -u low "This is a test"
```

The **-t** flag here determines for how long the notification should appear (zero = forever), while **-u** chooses the urgency level (low, normal, critical). You might find that the default *Dunst* configuration doesn't look very good; in this case, copy **dunstrc** into **~/.config/dunst/** and edit it to your liking.

Dunst includes a bag of features such as multiple monitor support, a history of previous notifications, and custom scripts that can be run when notifications match a certain



Different urgency levels have different colours, and you can see for how long a notification has been displayed.

text pattern. You can left-click to disable a single notification or right-click to disable all – and there are keyboard shortcuts too. You can even add some formatting to your notifications, using **,** for example, **<u>underline</u>** tags for a bit of variety.

One day we plan to create the Ultimate Mega Linux Voice Desktop Turbo Championship Edition, with a tiling WM and various other tools, and *Dunst* is sufficiently awesome to be included.

“Dunst is one of the best notification managers we’ve seen.”

PROJECT WEBSITE
www.knopwob.org/dunst

Machine emulator

Qemu 2.1.0

Here at Linux Voice HQ, we tend to use *VirtualBox* for testing Linux distros and other x86 operating systems, but we're big fans of *Qemu* as well. It includes emulation for a wide range of CPUs, including ARM, MIPS, SPARC and PowerPC, so you can use it to try out some very obscure and esoteric OSes.

Qemu 2.1 was released as we were finishing this issue, but we managed to find time to take it for a spin thanks to an Ubuntu PPA. While it's certainly possible to compile *Qemu* from source, the large range of CPU architectures and emulated devices make the process a long one – it could take a few hours on older machines. (And code that emulates processors is always a good stress test for the compiler.) The main dependency is

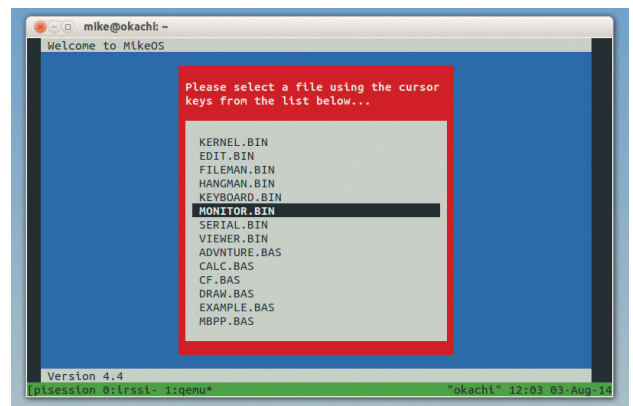
SDL to provide graphics; it's possible to run *Qemu* solely in text mode, though, if you're planning to run a text-based operating system.

To run the x86 version of *Qemu* with a hard drive image, enter:

```
qemu-system-i386 -hda drive.img -m 1024
```

This uses **drive.img** as the virtual hard drive, and provides 1024MB RAM to the emulated PC. Enter **qemu-system** at your shell prompt and hit Tab to see the other platforms that are supported. Note that on x86 systems, when emulating an x86 machine, *Qemu* can use *KVM* for a performance boost – it then uses virtualisation to pass the grunt work on to the host CPU, rather than emulating every CPU instruction in code.

New features in *Qemu 2.1* include support for memory hotplugging on



Here's MikeOS (x86) running in *Qemu* on a Raspberry Pi (ARM) in a terminal window over an SSH connection. It's the future!

x86 guests, full support for USB3 passthrough devices, and AArch64 (64-bit ARM) SHA and Crypto instruction support.

It's a major release with many changes all over the codebase; see <http://wiki.qemu.org/ChangeLog/2.1> for the full list. Now, time to dig out that old version of Coherent Unix from 1994...

PROJECT WEBSITE
www.qemu.org

FOSSPICKS Brain Relaxers

Penguin-based slippery-slider

Extreme Tux Racer 0.6

Hooray – *Tux Racer* lives! Back in the early 2000s, this was one of the flagship 3D games for Linux. It was smooth, polished and had you controlling a penguin sliding down a mountain collecting fish... what's not to love?

Things are very different today in the Linux gaming world, with Steam and GOG providing a stunning range of triple-A titles to explore. So we thought that *Tux Racer* had long been abandoned and the code was suffering from excessive bitrot, but with *Extreme Tux Racer* it's still going strong.

To compile it, you'll need version 1.2 of SDL and its related *mixer* and *image* libraries; then the standard **./configure, make** and **make install** (as root) procedure

will get it installed. After that, run **etr** to start the game. There are two main gameplay modes: event and practice. In the former, you compete in a series of cups, making progress in *Mario Kart*-style. The practice mode lets you try courses in a less competitive environment.

Left turn, Clyde

Control-wise it's simple stuff: use the arrow keys to turn left and right, press up to race faster, and down to dig your flippers into the snow (to slow down). It's also possible to jump by holding Space until the gauge in the bottom-right is full – you don't get a great deal of lift though. The 18 courses range from very tame to full-on crazy, with some hairpin turns, spiky mountains and chasmic jumps



The Path of Daggers is an especially hairy course to navigate...

livening things up. It doesn't have the depth or complexity of most commercial racing games, but it's still good fun, especially for kids.

PROJECT WEBSITE
<http://sourceforge.net/projects/extremetuxracer/>

City construction romp

Micropolis 1.6

S*imCity* has been a hugely popular franchise over the years, with each release adding more complexity and variation to the gameplay. But we still have a soft spot for the original – and especially the Super NES conversion which added cheerful music and Mario-themed add-ons. It seems quite primitive compared to the later games, but still has its charms and is enjoyable to play.

Micropolis was the original working title for *SimCity*, and today it refers to a freely GPLed version of the original source code. This has been rewritten in various languages; the version we're looking at here uses Java. Grab the **micropolisj-1.6.zip** file

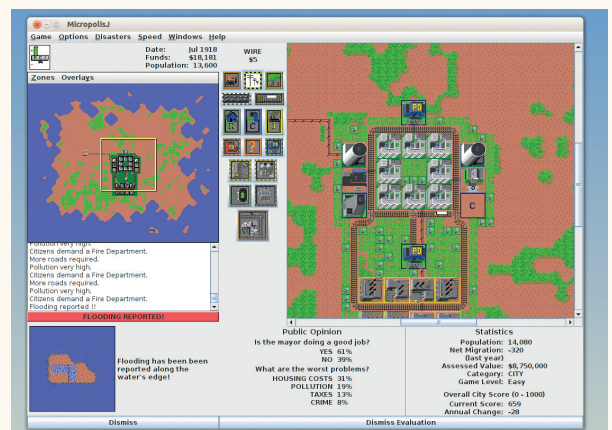
from the project's website, but note that it doesn't extract into a subdirectory – just the current one. (All Zip files should extract into their own subdirectories in our opinion – write in if you disagree!)

But anyway. Run:

```
java -jar micropolisj.jar
```

In the middle of the screen you have icons for building commercial, residential and industrial zones, along with power lines and roads/railways to connect them. Just like in the original, your goal is to keep growing the population, keeping people safe, happy and free of pollution.

You can add natural disasters for an extra challenge, and under the Windows menu you'll find Evaluation, which shows you what



Only 61% of Mikeville's residents are happy with the mayor. Let's see how they fare when we close the police stations...

your virtual inhabitants think of your city. They're always moaning about something though: even if your city is heaven to live in, they'll say it's too expensive. Humans, eh?

PROJECT WEBSITE
<https://code.google.com/p/micropolis/>

PRIVACY IS IMPOSSIBLE



WITHOUT FREE SOFTWARE

TUTORIALS

Dip your toe into a pool full of Linux knowledge with eight tutorials lovingly crafted to expand your Linux consciousness



Ben Everard

is wondering what the best way of sneaking an arcade machine into his new house is.

I'm currently packing up to move house. This is an exciting time for a geek because it means I'll have a blank canvas on which to wire my geeky thoughts. In my current place, we have cable TV, which doesn't allow much hacking, but I've already got the hardware ready for a DVB-based *TV-Headend* box to control the TV in my new place.

Of course, if I'm going to have a system set up to record TV shows, then it may as well have NAS functionality. I mean, it's basically there already isn't it?

And if I'm going to be running a machine 24/7 for a NAS/TV tuner, well, it would be sensible to get as much functionality out of it as possible wouldn't it? I've got a few adaptors to control sockets so I can turn things on and off. It may as well include a security camera – one that uploads footage to an external server obviously.

Oh, and I think I've got a servo that could control the cat flap quite nicely – the cat gets in fights if he stays out at night. Of course, the cat's a fluffy little Luddite and scared of anything technological, so he might never leave the house anyway.

If you notice my absence for a few issues, it's nothing to worry about. Moving house can take a little time, as I'm sure you can appreciate.

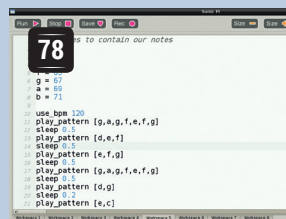
ben@linuxvoice.com

In this issue...



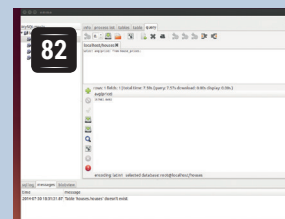
Extreme Pi

Now he's got a new model B+, **Ben Everard's** started conducting experiments on his older model B to see just how far he can push it.



Sonic Pi

Les Pounder introduces a musical programming language designed to prevent the next generation liking Justin Bieber.



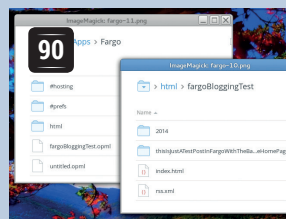
Data analysis

There are lies, damn lies, and statistics. **Ben Everard** doesn't trust anyone, so makes his own statistics from raw data.



Beginning Bash

Power up your shell, as **Mike Saunders** guides you through creating a swanky custom prompt, multiplexing and more.



Fargo 2

What's an outliner, and why do you need one? **Marco Fioretti** introduces *Fargo 2*, which helps you structure your writing better.



PyUSB

Valentine Sinityn rolls up his sleeves and gets elbow deep in some hardware as he reverse-engineers a device driver.

PROGRAMMING

Bash scripts

100 There's more to Bash than just an interactive shell. It's also a programming language in its own right. However, some of the syntax is a little archaic, and it can get confusing. Fear not! We're here to help – with this beginner's guide to Bash Scripting, you'll be writing your own programs in no time at all.

Programmer's golf

104 It's a simple game to see who can write a program in the shortest amount of code. However, completing this task can be fiendishly complex as you try even more obscure methods to shave just one or two bytes off. Read this tutorial to find out more, but be warned, it can be addictive. Don't say we didn't warn you!

Konrad Zuse

106 As Alan Turing and his colleagues were developing computers for the Allied military's code breaking effort during World War II, the Germans had their own electrical engineer. Konrad Zuse, working alone, also managed to build calculating machines and the first programmable computer in continental Europe.

BEN EVERARD

RASPBERRY PI MODEL B: VOID YOUR WARRANTY

Now we have a shiny new B+, it's time to try some dangerous experiments on our old Raspberry Pi model B.

WHY DO THIS?

- Learn the limits of your Raspberry Pi.
- Let an old model B go out in a blaze of glory.
- Add new features and personalise your Pi.

Now the Raspberry Pi B+ has come out, we've found ourselves with some of the original model B's that we're not going to use any more. These are still fully functional computers, so it seems a waste to let them rot in a drawer, or worse, throw them out. Instead, we decided to use one as a test bed for some riskier experiments.

We didn't break a Pi while researching this article, but we certainly could have done. We accept no responsibility should you slip and fry your Pi, but what better way is there to get to know a device than to push it to its limits?

Overclocking

Raspbian comes with **raspi-config**, a tool that lets you set various configuration options for your Raspberry Pi. One of which is the overclocking level. It has a series of safe levels that can give you a bit of a speed boost without damaging your Pi (though not all Pis will work at the highest speeds). This is useful for getting a bit more oomph, but it obviously raises the question of just how fast you can push your Pi.

To take things further than **raspi-config**'s menu will allow, you'll need to edit the **config.txt** file on the boot partition of the SD card. It's easiest to do this after you've set the Pi to Turbo overclocking (one of the options in the config tool) since this makes most of the options visible. You can edit this file either by putting the SD card in another computer, or from within the Pi with:

```
sudo nano /boot/config.txt
```

The de-soldering pump we used. The orange button triggers the suction and pulls the molten solder off the board.



Before going any further, we should say that there's a chance that following this tutorial will void your Pi's warranty, and there's a small chance that it'll explode in a shower of sparks (and a slightly larger – but still small – chance that it'll break in a less spectacular way). In other words, don't try this if you're not prepared to accept the risk that your Pi will stop working permanently.

If you're already in Turbo mode, you should find the following options set:

```
arm_freq=1000
```

```
core_freq=500
```

```
sdram_freq=600
```

```
over_voltage=6
```

You can mess with these to boost the performance. The three frequency settings are all in MHz, so this configuration has the main ARM processor running at 1GHz, the GPU running at 500MHz, and the SD RAM running at 600MHz. We found that we couldn't squeeze any more speed out of the GPU or the SD RAM. However, there does tend to be a little headroom in the ARM frequency.

In order to take advantage of this, though, you'll need to increase the voltage. The voltage for the core defaults to 1.2V, and each increase in the **over_voltage** setting sends an extra 0.025V. With a setting of 6, the core is running at 1.35V. Increasing the voltage enables you to increase the speed, but it can also decrease the life expectancy of the chip. Since we're seeing how much speed we can get, we whacked this up to its maximum setting of 8 (1.4V).

There's another option that you'll need to set if you want to take it beyond the normal overclocking levels:

```
force_turbo = 1
```

Just add this line to the **config.txt** file, and it'll let you push the performance up.

There are a couple of things you need to be aware of as you increase the clock speed. The most obvious is that it will become more prone to crashing, so don't use a heavily overclocked machine for any important work. The second important thing is that it will tend to run hotter than at slower speed, so you need to keep an eye on the temperature to make sure it doesn't get too hot.

You can check the temperature at any time with the command:

```
cat /sys/class/thermal/thermal_zone0/temp
```

This gives the temperature in 1000ths of a degree Celsius, so 45000 is 45°C. As a general rule of thumb,

you want to keep the temperature below 70000, but again this depends on how much you're willing to risk breaking your Pi. We gradually increased the clock speed in 50MHz increments, and performed a simple benchmark of unzipping an archive.

We found that we could run our Pi at 1.2GHz, though it wasn't very stable. At this speed, our benchmark ran about 40% faster than at non-overclocked speeds, and about 20% faster than Turbo overclocking, with a core temperature of around 60°C. This was, however, quite a simple benchmark. A more complex task may well have proved too much for the SoC at this speed. However, we did find that our Pi was reasonably stable at around 1.1GHz.

Modifying the board

You may think that, unlike desktop PCs, you can't change much on single-board computers like the Raspberry Pi. You may think that the Raspberry Pi Foundation choose what goes on the board and you just have to go with it. This isn't necessarily true. They're certainly not as flexible as desktop PCs, but with a little soldering, you can certainly tweak them to your needs. We stripped off a component we didn't need, and added one we did.

The analogue video output may be useful to some people, but not for us. It just takes up space and makes the board look cluttered. This wasn't enough for us to risk removing it before, but now we've had enough, and decided to take it off. It's only attached by three soldered points that are quite large and easy to access, so it's easy to remove.

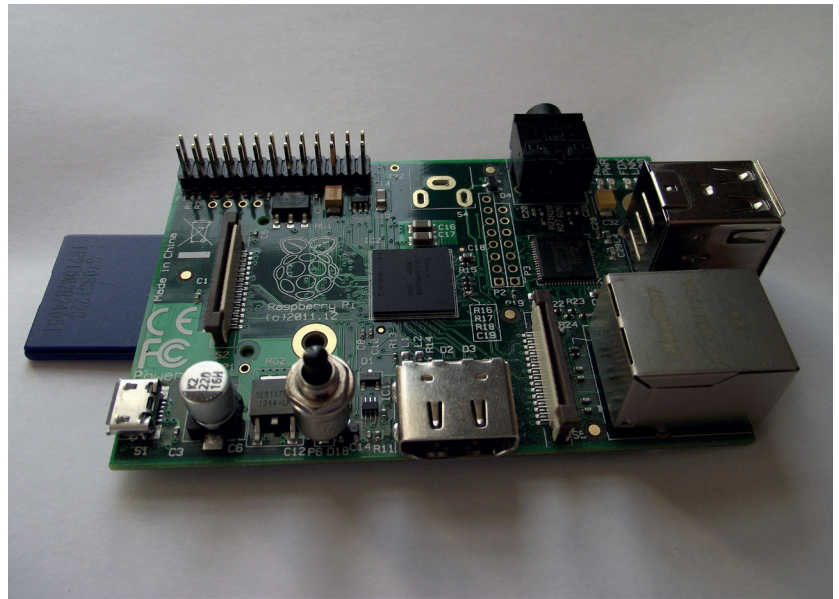
You will need either a desoldering pump or wick – we used a pump. This is a device that looks a bit like a syringe from a sci-fi film. It has a spring-loaded plunger that you press down, then a button that releases the plunger. As the plunger shoots up, it sucks air in through the nozzle. If the nozzle is placed near molten solder, it'll suck the solder off the board.

The trick of desoldering is to heat the solder up until it's molten, then use the pump or wick to remove it before it cools. If you get enough off, you should find that the component slides out. Be careful not to heat up the board too much, or pull too hard on the component, or you could damage the board. It's made of several layers stuck together, so there could be wiring you can't see.

This might seem a bit pointless, and on its own it is a bit, but for some projects where size or weight is important, removing unnecessary components can be useful.

Extra features

You may have noticed that there are loads of extra holes in model B Pis that don't seem to be used. The two larger ones are mounting holes. Some of the others are used for manufacturing and testing, but some of them allow access to extra features. One mildly annoying feature of the Raspberry Pi is the fact that there's no power button. You can halt it from




The board after our modifications. If nothing else, it now feels like it's truly our board and not just another Raspberry Pi model B that's rolled off the production line.

software, but to turn it on you have to unplug the power cable, then plug it back in. That's not the most user-friendly way of doing things, and it could be awkward in embedded settings. Fortunately, there's an alternative. Between the HDMI and power connectors, you should see two holes labelled P6. These are mounting points for a reset switch.

All you have to do is make a connection between these two points and the Pi will reset. You can test this out using a flat-head screwdriver. This will reboot your Pi whether it's switched on or off.

In order to be able to reset your Pi, you simply need to add a normally-open push switch between these two points. The best way to do this will depend on how your Pi is set up. We use ours without a case, so we simply soldered the switch straight onto the board. However, if your Pi is inside a case or some other enclosure, you may find it easier to solder wires onto the board and attach those wires to a switch in a more convenient location. On the B+, the connections for the reset button are labelled Run, and are located next to the micro SD card slot.

Not many of the holes in the model B are useful. There are a couple of extra GPIOs on the P5 header that you could find functions for. If you really want to modify your Pi, you could replace components on there with better ones. For example, it is possible to take off the linear regulator from a model B and replace it with a switching reg. In doing this, you'll gain one of the best features of the B+. It's not a particularly simple process, but there's some guidance on Dave Akerman's excellent blog at www.daveakerman.com/?page_id=1294.

Despite being superseded by the B+, there's still plenty of fun left to be had with model B's, so don't let yours rot in a drawer. Get it out and void its warranty – and learn more about it in the process. 

SONIC PI: PROGRAM ELECTRONIC MUSIC

Learn a new style of coding and get instant musical feedback with this great tool for the Raspberry Pi.

WHY DO THIS?

Programming is much more than logic and control, and creating music shows us how simple logic can be used with creativity to make music. Musicians around the world have learnt how to create music using logic and maths and to sequence their compositions for better sounding tunes.

TOOLS REQUIRED

- Raspberry Pi, any model will do.
- Keyboard, mouse and screen for your Raspberry Pi.
- Sonic Pi v2 installed, we will show you how to do that later in this tutorial.
- Headphones / Speakers if using the 3.5mm headphone socket.

In this month's tutorial we will take a break from Scratch and Python and try something new. Let's jam with Sonic Pi! Sonic Pi v1 is the creation of Sam Aaron, with the full support of the Raspberry Pi Foundation. Sonic Pi v1 comes as a pre-installed application available to all Raspbian Raspberry Pi users and enables anyone to make music using a programming language called Ruby. Ruby is a simple to learn language that has some similarities to Python, so it's handy for those already competent in Python.

For this tutorial we will be using the latest version of Sonic Pi, v2, which at the time of writing is still a release candidate but fully up to the task at hand.

Using Sonic Pi we will first create a basic song and then use programming logic to refine our work. The song chosen is the classic nursery rhyme 'London Bridge is Falling Down', but any song can be played with Sonic Pi, so feel free to experiment. During the course of this project we will learn some important programming concepts:

Sequences In order for our tune to play correctly we need to understand how we can translate the musical sequence into code, otherwise our tune would not sound very good.

Loops Using a loop introduces recursion into our programming and with it comes the art of creating the correct structure so that our loops are seamless, as a note in the wrong place can ruin our tune.

Data storage Computers have a great memory and can remember lots of things, but only if we tell them to. Variables are used to temporarily store data for use in our project.

Configuring audio

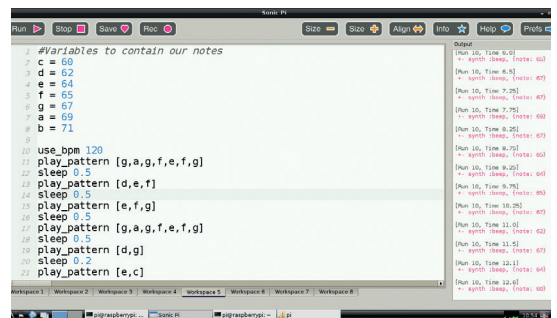
By default the Raspberry Pi will use the HDMI connection to your television for audio and video. But if you would like to use the headphone socket, say to connect to your Hi-Fi, speakers or headphones then you will need to tell your Pi that you would like to.

The best way to accomplish this is by using the **raspi-config** menu. Open a terminal and type in:

```
sudo raspi-config
```

In the menu that appears, look for 'Advanced Options', navigate to it using the cursor keys and press Enter to select it.

Inside the Advanced Options menu there will be an Audio option; select this option and a new menu will appear. This new menu enables you to choose the output method, select the analog audio output and



At the end of this project you will have created your own version of the 'London Bridge' nursery rhyme.

then exit out of **raspi-config**. For best results reboot the Raspberry Pi

Once the reboot is complete, plug in your headphones/speakers and test that they are working. If you need to fine-tune the general volume settings, open a terminal and type in the following:

alsamixer

Alsa Mixer is a terminal application that enables a user to control the volume level; you can alter the volume by pressing the up and down arrows on your keyboard. Once you're happy with the levels, press Esc to exit.

Installing Sonic Pi v2

To download, install and start *Sonic Pi*, open a terminal and type in each line followed by Enter at the end of each line:

```
wget http://sonic-pi.net/sonic-pi-RC11.tar.gz
```

```
tar -xvzf sonic-pi-RC11.tar.gz
```

```
./sonic-pi/bin/sonic-pi
```

With *Sonic Pi* started, let's take some time to familiarise ourselves with the layout.

Towards the top of the screen there's an area that contains buttons to handle the following actions.

- Run/Play our tune.
- Stop playback.
- Save our tune in the Ruby file format.
- Record the tune as a WAV file so that we can share it with others.

Moving further along we can see some more buttons in the row.

- **Size -** and **Size +** decrease and increase the size of the text in the project window.
- **Align** is a tool to automatically align any indented code, helping to format the project correctly and

minimise any potential bugs.

- **Info** opens an about window, telling us who made this great application.
- **Help** will change the bottom left of the screen and introduce a series of tabs which contain information on how to use *Sonic Pi* and its instruments.
- **Prefs** is the preferences menu, where volume levels can be adjusted.

Underneath these buttons there are three main sections of the screen. To the top-left is a project area where we write the code that makes our tune. To the top-right there is an output window, which will show the progress of our project. Finally, to the bottom-left are the workspaces, numbered 1 to 8. *Sonic Pi* can work with eight projects at once, so we can have one workspace to contain our main piece of work, and others to try out new ideas and logic.

First tune

For our first project we will create the nursery rhyme 'London Bridge Is Falling Down'. We will be using the MIDI (Musical Instrument Digital Interface) number for each of the notes. In this notation, G is 67, A is 69 and so on (see the boxout over the page for more information on MIDI numbers).

Nursery rhymes are a great way to introduce music theory and *Sonic Pi* due to their simple melodies and limited use of notes and chords. Once we understand the basics we can then tackle much larger compositions, indeed if you can find the notes for your favourite song then you can easily recreate it in *Sonic Pi*. Sam Aaron has used *Sonic Pi* to recreated 'Blue Monday' by New Order – take a look at his video <http://bit.ly/LVSonicPi>.

'London Bridge Is Falling Down' is a simple melody that starts in the key of G, and the opening motif goes as follows

London	G, A
Bridge	G
Is	F
Falling	E, F
Down	G

So how can we code this in *Sonic Pi*?

To play a note we first need to understand how we instruct the computer to do so. *Sonic Pi* can play a single note via the **play** function. So to play a G we will need to do the following in Workspace 1:

```
play 67
```

And to play the other notes we would need to add the following after **play 67**:

```
play 69
```

```
play 67
```

```
play 65
```

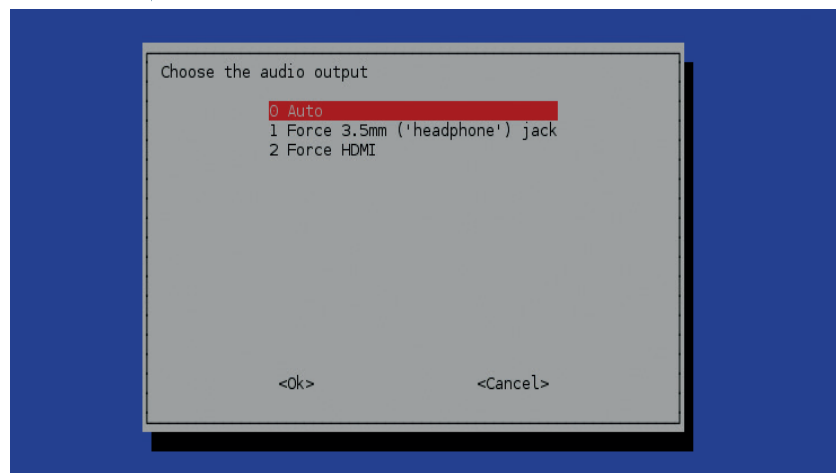
```
play 64
```

```
play 65
```

```
play 67
```

With this code in our workspace, click on the run button to play your tune.

How does your tune sound? Is the speed wrong? We didn't tell the computer to play the notes one after



another, so *Sonic Pi* will try and play all at once, leaving us with a horrible noise rather than beautiful music. To fix this we can insert a delay using the **sleep** function. This function adds an element of control to our code.

Between each of the notes that we used previously, insert the following:

```
sleep 0.3
```

This uses a float value of 0.3 seconds to delay the playback of the notes. Listen for yourself, and it should sound much better.

Now that we have our basic code, let's improve it and make it more compact.

Sonic Pi has a great feature which enables you to play a pattern of notes much more simply than playing each note individually. The function **play_pattern** can take multiple MIDI notes and play them in succession. So let us rewrite our code to use this new function:

```
play_pattern [67,69,67,65,64,65,67]
```

When it's completed, play the code. It should sound a little slow, so let's speed it up a bit using a tempo.

To introduce tempo into our project we need to use a BPM (Beats Per Minute) value. Go back to your code and make sure that the following is the first line of code, with all other lines being underneath.

```
use_bpm 120
```

Now click on the run button, and the music should sound a lot better. Congratulations: you've taken the code from a simple line-by-line sequence and using the **play_pattern** function created a more compact and robust project.

Using variables

Variables are a temporary method of storing data, and



There are three choices in the audio output menu, auto, force 3.5mm and force HDMI. If you are listening via headphones connected to your Pi choose the second option.

The simple, uncluttered layout of *Sonic Pi V2* is a credit to the team behind it.



```

Sonic Pi
Run Stop Save Rec
Size Size Align Info Help Prefs

1 play 67
2 play 69
3 play 67
4 play 65
5 play 64
6 play 65
7 play 67

Output
==> Starting run 3
[Run 3, Time 0.0]
- synth :beep, (note: 67)
- synth :beep, (note: 69)
- synth :beep, (note: 67)
- synth :beep, (note: 65)
- synth :beep, (note: 64)
- synth :beep, (note: 65)
- synth :beep, (note: 67)
==> Stopping all running code.
    
```

Our simple melody should look like this to start with, but over the course of the tutorial we will alter and re-work the code.

they can greatly improve our coding. So far we have been using the MIDI numbers that represent the notes in our tune. But it can be difficult to remember what number is for which note. Using a variable we can store the MIDI number and label the variable to match the pitch of the note, so you don't have to remember the MIDI values. At the top of your code, create the following variables:

```

c = 60
d = 62
e = 64
f = 65
g = 67
a = 69
b = 71
    
```

Now, using the variables instead of their MIDI numbers, let's rewrite our code to reflect this and write the rest of the song. Once written, try out your code.

```

play_pattern [g,a,g,f,e,f,g]
play_pattern [d,e,f]
play_pattern [e,f,g]
play_pattern [g,a,g,f,e,f,g]
    
```

Midi notes

Sonic Pi uses numbers to represent the notes played in music. These numbers are MIDI representations of those notes. MIDI (Musical Instrument Digital Interface), has long been used in the professional music community as a method of working with computers and external musical instruments, commonly keyboards. With MIDI you can easily make a change to a song without having to re-record the instrument, as the data is saved in the MIDI format.

Sonic Pi has access to the full range of MIDI numbers, but to keep things simple we're using just seven of them: C,D,E,F,G,A,B. These are more than enough for simple tunes.

To use these notes in our project, we must learn their MIDI value – below is a table of this information.

C	60
D	62
E	64
F	65
G	67
A	69
B	71

There's a great resource for MIDI notes included in the [readme file on the GitHub repository](https://github.com/lesp/LinuxVoiceSonicPi) for this project at <https://github.com/lesp/LinuxVoiceSonicPi>.

play_pattern [d,g]

play_pattern [e,c]

That sounds better, but how can we make this code even better? By adding a delay between each of our patterns. Sonic Pi uses the **sleep** function to delay a step in the sequence of code. If we use the **sleep** function with another variable we can set a universal delay to our code.

On a line below our previous variables, create the following:

delay = 1

Now insert the following in between each of the **play_pattern** lines of code, then run your code:

sleep delay

How does it sound? Perhaps a little slow in between each of the **play_patterns**? In that case, reduce the **delay** value by using a float instead of an integer. This will enable you to use fractions of a second. Try a few lower numbers and see what works for you.

Taking our music to the next level

Our tune sounds great – all of the timings and logic we used have sharpened our tune to perfection, but something is still missing. Perhaps we could add an instrument or two? As *Sonic Pi* uses MIDI, we can introduce new instruments to our project relatively easily.

Currently we use the default tone for our tune, but we can investigate some other instruments.

Sonic Pi comes with a plethora of instruments that we can use in our project. From simple pretty bell chimes to dark and melodious “fm” which at times can sound like playing a Beatles record backwards.

To introduce an instrument into our project we must first tell *Sonic Pi* that we wish to use it and the best place to do so is underneath where we said **use_bpm 120** like so:

use_bpm 120

use_synth :pretty_bell

Now play your tune – instead of the standard sound you should now hear a bell like chime.

Looping

Looping is the practice of repeating a section of code either many times or infinitely. For our tune we will use it to repeat the sequence of code that makes up our tune.

```

Sonic Pi
Run Stop Save Rec
Size Size Align Info Help Prefs

1 use_bpm 120
2 play_pattern [[67,69,67,65,64,65,67]]

Output
==> Starting run 6
[Run 6, Time 0.0]
- synth :beep, (note: 67)
- synth :beep, (note: 69)
- synth :beep, (note: 67)
- synth :beep, (note: 65)
- synth :beep, (note: 64)
- synth :beep, (note: 65)
- synth :beep, (note: 67)
[Run 6, Time 0.5]
- synth :beep, (note: 67)
- synth :beep, (note: 69)
- synth :beep, (note: 67)
- synth :beep, (note: 65)
- synth :beep, (note: 64)
- synth :beep, (note: 65)
- synth :beep, (note: 67)
[Run 6, Time 1.0]
- synth :beep, (note: 67)
- synth :beep, (note: 69)
- synth :beep, (note: 67)
- synth :beep, (note: 65)
- synth :beep, (note: 64)
- synth :beep, (note: 65)
- synth :beep, (note: 67)
[Run 6, Time 1.5]
- synth :beep, (note: 67)
- synth :beep, (note: 69)
- synth :beep, (note: 67)
- synth :beep, (note: 65)
- synth :beep, (note: 64)
- synth :beep, (note: 65)
- synth :beep, (note: 67)
[Run 6, Time 2.0]
- synth :beep, (note: 67)
- synth :beep, (note: 69)
- synth :beep, (note: 67)
- synth :beep, (note: 65)
- synth :beep, (note: 64)
- synth :beep, (note: 65)
- synth :beep, (note: 67)
[Run 6, Time 2.5]
- synth :beep, (note: 67)
- synth :beep, (note: 69)
- synth :beep, (note: 67)
- synth :beep, (note: 65)
- synth :beep, (note: 64)
- synth :beep, (note: 65)
- synth :beep, (note: 67)
[Run 6, Time 3.0]
- synth :beep, (note: 67)
- synth :beep, (note: 69)
- synth :beep, (note: 67)
- synth :beep, (note: 65)
- synth :beep, (note: 64)
- synth :beep, (note: 65)
- synth :beep, (note: 67)
    
```

play_pattern is a handy function that can considerably reduce the number of lines in our code, making it much easier to read.

To use a loop we use the following line of code

```
2.times do
```

```
#What code would you like to repeat?
```

```
end
```

You can see that the second line is indented; this shows that this is the code to be repeated, under our instruction of **2.times do**. This indentation is not as restrictive as Python, which requires 4 spaces to signify indentation. Sonic Pi will accept a single space or a tab indentation, but don't mix the two together, or you will have a headache debugging your code.

If we wanted to play a C note twice using the looping method we could approach it like this:

```
2.times do
```

```
  play 60
```

```
end
```

To use the code in our tune we must do the following:

```
2.times do
```

```
  use_synth :pretty_bell
```

```
  play_pattern [g,a,g,f,e,f,g]
```

```
  .... all of the code to play our tune.
```

```
End
```

After all of our coding, your program should look this

```
#Variables to contain our notes
```

```
c = 60
```

```
d = 62
```

```
e = 64
```

```
f = 65
```

```
g = 67
```

```
a = 69
```

```
b = 71
```

```
2.times do
```

```
  use_bpm 120
```

```
  use_synth :pretty_bell
```

```
  play_pattern [g,a,g,f,e,f,g]
```

```
  sleep 0.5
```

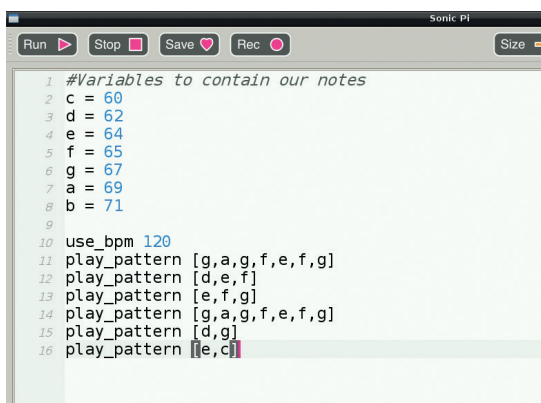
```
  play_pattern [d,e,f]
```

```
  sleep 0.5
```

```
  play_pattern [e,f,g]
```

```
  sleep 0.5
```

```
  play_pattern [g,a,g,f,e,f,g]
```



```

1 #Variables to contain our notes
2 c = 60
3 d = 62
4 e = 64
5 f = 65
6 g = 67
7 a = 69
8 b = 71
9
10 use_bpm 120
11 play_pattern [g,a,g,f,e,f,g]
12 play_pattern [d,e,f]
13 play_pattern [e,f,g]
14 play_pattern [g,a,g,f,e,f,g]
15 play_pattern [d,g]
16 play_pattern [e,c]

```

Variables enable us to store the MIDI numbers inside a container which, for ease of use, have been labelled to match the note.

What is Ruby?

Ruby was designed and developed in the mid-1990s by Yukihiro "Matz" Matsumoto in Japan to be a general-purpose programming language. Ruby can be used in both a functional capacity, where code happens in a sequence, such as our project, and in an object-oriented capacity, where code can be written using objects and classes.

Ruby is an excellent language to learn due to its very clear syntax and legibility. The programming logic learnt via Scratch and Python can be applied to Ruby, and in turn can be applied to Sonic Pi. If you would like to learn more about Ruby, there is a great interpreter called *IRB*, which can be installed via the terminal.

For Raspberry Pi- and Debian-based distros you can install as follows:

```
sudo apt-get install ruby
```

And for *yum*-based systems.

```
sudo yum install ruby
```

Using Ruby is remarkably simple, and the best way to get started is to open a terminal and type *irb* followed by Enter.

We are now in an interactive session of Ruby and can write Ruby code line by line.

First of all, let's print "Hello" on the screen.

In Ruby the function to do that is called **puts** and you would use it like this:

```
puts "Hello"
```

So now let's use a loop to print hello twice:

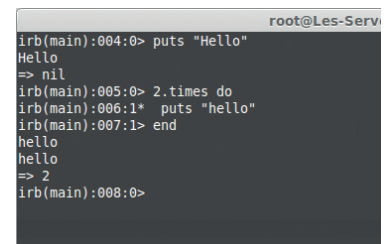
```
2.times do
```

```
  puts "Hello"
```

```
end
```

Can you see how the loop works? That's right – exactly the same way as the loop in our project does.

The official 20-minute guide to Ruby is available at www.ruby-lang.org/en/documentation/quickstart, and is a fantastic resource for learning this great language.



```

root@Les-Serv
irb(main):004:0> puts "Hello"
Hello
=> nil
irb(main):005:0> 2.times do
irb(main):006:1* puts "hello"
irb(main):007:1> end
hello
hello
=> 2
irb(main):008:0>

```

Like Python, Ruby is designed to have a simple, easy-to-read syntax.

```
sleep 0.5
```

```
play_pattern [d,g]
```

```
sleep 0.2
```

```
play_pattern [e,c]
```

```
end
```

Congratulations, you have now created your first piece of music using *Sonic Pi*. Using what you have learnt, try the following extension activities:

- 1 Add a drum beat to the London Bridge project by using another function called **in_thread**. This function will enable you to have two or more independent threads of code running at once. Code in a thread runs completely isolated from the main body of code. For example to play a G note every half a second we would write the following:

```
in_thread do
```


```
  play 67
```

```
  sleep 0.5
```

```
end
```

Have a play with this code and see what works for you.

- 2 Find a song that you like on YouTube and then use a search engine to find the sheet music to play it, then convert the tune into something that *Sonic Pi* is familiar with. Remember that any piece of music can be written using *Sonic Pi*.

Finally, a great resource of Sonic Pi material is provided by Dan Aldred's blog www.tecoed.co.uk/sonic-pi.html – head over and take a look. 

Les Pounder is a maker and hacker specialising in the Raspberry Pi and Arduino. Les travels the UK training teachers in the new computing curriculum and Raspberry Pi.

DATA ANALYSIS USING PYTHON AND MYSQL

BEN EVERARD

WHY DO THIS?

- Pull out the information that's pertinent to you from a swarming mass of numbers.
- Improve your Python and SQL skills.
- Get your computer to draw pretty pictures that make you seem smart to friends, family and co-workers.

If you're using SQL for more than a few basic queries, there are some SQL clients (such as *Emma*, shown here) that can make your life a little easier.

Graphing data makes it easier to understand, and graphing lots of data is easy with a script and a database.

In recent years, governments around the world have been opening up their information archives to the public, and now there's more data available than ever before. However, the raw data is hard to digest, and it's often analysed by people with an agenda, whether that's newspapers trying to make a story sound exciting to sell more copies, or a company trying to make their product look better than the competition. It's hard to know whether data is being properly represented, so the solution is to dive in and analyse the figures for yourself. Let's take a look at how to do this using UK house prices.

You can get a complete list of every house sold in the UK along with its location, type (eg terrace, semi-detached) and price from data.gov.uk. The data goes back to 1994, and is licensed under the Open Government Licence, which allows us to manipulate the data and publish it – so that's what we'll do.

Spreadsheets, such as *LibreOffice's Calc*, can easily handle small data sets. However, this data set is too big and needs something a little more capable. We're going to use Python and *MySQL*, though you could use most programming languages and most databases for the task.

The data comes in a CSV file, which is a text file containing the values separated by commas. These are usually used with spreadsheets, but are also fairly easy to upload into databases. Databases enable us

much better access to the data from programming environments, and can also handle much larger data sets than spreadsheets.

First you need to grab the software we'll be using. That's *MySQL* (both a client and server), and two Python modules (*MySQLDB* and *Matplotlib*). These are all quite common, and should be in your package manager. To get them in Debian-based systems, use:

```
sudo apt-get install mysql-client mysql-server python-mysqldb python-matplotlib
```

If your package manager hasn't asked you to set up a root password for *MySQL*, you can do that now with:

```
sudo mysqladmin -u root -p password newpass
```

Replace **newpass** with a password of your choice.

Get the data

Now you've got the software, you just need to grab the data. The easy way to do this is to download our database dump from www.linuxvoice.com/house-price-analysis.

This is an xzipped SQL file, so you can load it with:

```
unxz house_prices.sql.xz
```

```
mysql -u root -p < houseprices.xz
```

This will create a database called **houses**, and a table within it called **house_prices** that contains all the information we're going to work with.

That's the easy way. The hard way (which you'll need to do if you want to load data other than UK house prices), is to download the raw CSV files and load them into *MySQL*. This isn't too hard, but it can be a little fiddly.

First you need to get the CSV files. The ones we've been using are from data.gov.uk. However, there are loads of sources of open data you may wish to use (see the boxout over the page for more details). CSV files are often created with Windows encoding rather than Unix. There's a utility called **dos2unix** that can fix this, which you use with:

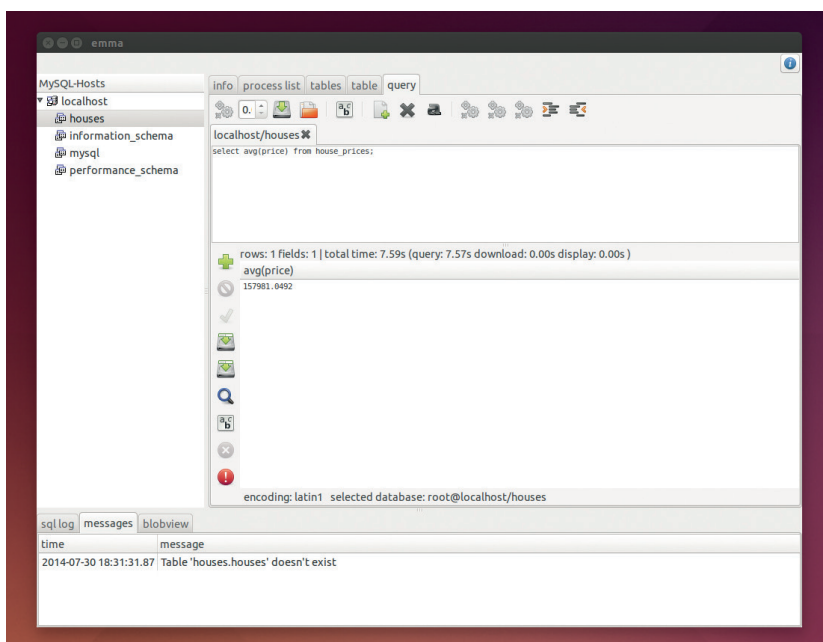
```
dos2unix <filename>
```

MySQL is really designed as a server tool, not a desktop one. This means that it has a few security features that you may not expect. One such feature is that by default, it won't usually load local files. You can get around that by starting the client with the **--in-file** flag:

```
mysql --u root -p --in-file
```

This will drop you into the *MySQL* commandline. First you need to create a new database to use:

```
create database houses;
```



use houses;

Now you need to create a new table to store the data. This has to have the same layout as the CSV files that you want to upload. For example:

```
create table house_prices (id varchar(50), price int, date
datetime, postcode varchar(10), type varchar(1), newbuild
varchar(1), leasefree varchar(1), address1 varchar(50), address2
varchar(50), address3 varchar(50), address4 varchar(50),
address5 varchar(50), address6 varchar(50), address7
varchar(50), dontknow varchar(1));
```

With all this set up, you can load the files with the following SQL statement:

```
load data local infile "file_name.csv" into table house_prices
fields terminated by ',' enclosed by '"';
```

The UK house price data comes in separate files for each year. You can use the **cat** command to join them together into one big file, or import them individually (which makes it easier to identify problems).

Getting started with SQL

Now you've got everything in the database, you can use SQL to pull out the information you want.

The basic usage of SQL to pull information out of a database is in the form:

```
select <something> from <table> where <condition>;
```

This is quite simple, but it enables you to get almost anything you need from the data store, and gives you a quick way of getting data (although complicated queries on large bodies of data can be slow).

For example, to get all of the price and house numbers for a particular postcode, you can use:

```
select price, address1 from house_prices where postcode = "XX1
1XX";
```

where **XX1 1XX** is the postcode. As well as getting specific bits of data, you can aggregate it using functions such as **avg()**, which returns the average.

For example, the following line returns the average price for houses in Bristol:

```
select avg(price) from house_prices where address6 =
"BRISTOL";
```

You'll see a few more SQL techniques as we go through the article, but they all follow this same basic process. If you're unsure of anything, *MySQL* has excellent documentation at dev.mysql.com/doc.

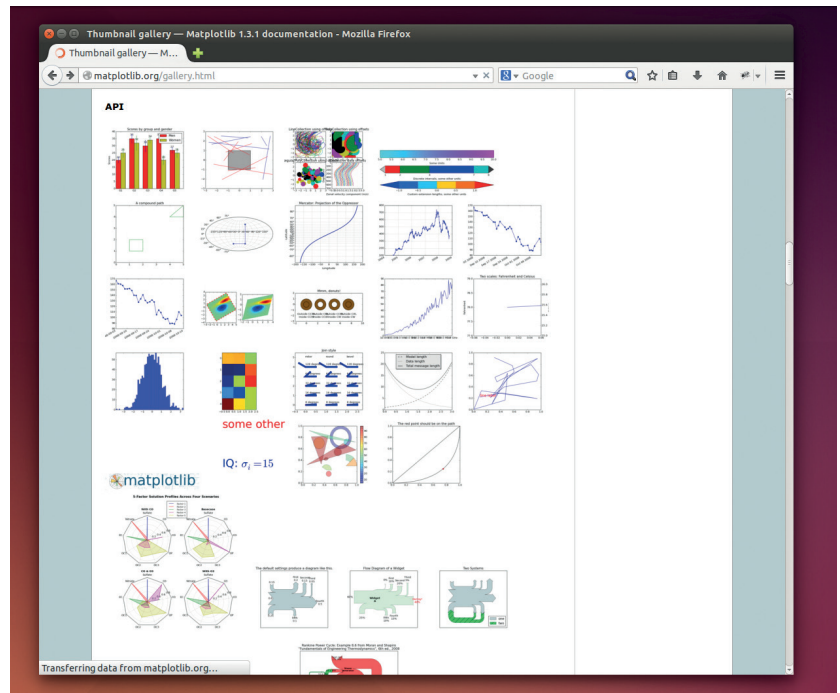
Drawing pictures with Python

SQL is great for pulling out bits of information, but it's not great at combining and comparing it. That's where Python comes in. We're going to use it to compare

MariaDB

We decided to do this tutorial using *MySQL*, because it's probably still the most widely used database for Linux. However, we know that a lot of people aren't happy with Oracle's handling of the project, and so may wish to use *MariaDB* instead, a fork of *MySQL* led by the original creator of *MySQL*, Michael "Monty" Widenius.

It should be completely compatible with *MySQL*, and so if you'd rather use this database, you should be able to follow along with this tutorial without any problems.



and graph the information we pull out of *MySQL* to make everything easy to understand.

In this case, our Python program will be acting as a glue between a module that access the database and a module that outputs graphs. Let's first look at *MySQLdb*, which we'll use to access the database.

Using the *MySQLdb* module is a fairly straightforward process. You have to connect to the database, and then create a cursor object. This cursor can then be used to execute queries and fetch the results. Take a look at the following example, which prints out the average house price in the data set.

```
import MySQLdb
```

```
db = MySQLdb.connect(host="localhost", user="root",
passwd="xxxx", db="houses")
```

```
cur = db.cursor()
```

```
cur.execute("select avg(price) from house_prices;")
```

```
result = cur.fetchone()
```

```
print str(result[0])
```

You'll need to change the password and possibly user in the **connect** command, depending on how your database is configured.

Once the connection to the database is set up, you can call **execute()** with a string containing an SQL query, and then get the result with **fetchone()**. This returns a tuple containing an entry for each column returned by the SQL (in this case, there's just one). If you expect the query to return more than one result, you can loop through them with:

```
for row in cur.fetchall():
```

```
    #do what you need to here
```

Since you just need to pass a string to **cur.execute()**, you can build this up with the usual Python tools. For example, if you want to get the average

The *MatPlotLib* project maintains a gallery of different chart types, and examples of how to use them at <http://matplotlib.org/gallery.html>.

Big data and NoSQL

Big data is one of the industry's current buzzwords. Like most tech buzzwords, there aren't any hard-and-fast rules to define it, but loosely speaking, it refers to any chunk of data that's too big to process on an ordinary computer, meaning you need some special setup to handle it efficiently. That could be a high-powered server, or a cluster of servers.

It is possible to use SQL databases to handle huge data sets, but specialist tools have sprung up to make it easier, and one common type is the so-called NoSQL variety of database. These are databases that don't use tables to hold structured information; instead they hold all the data in one non-structured mass. This means that for some processes, they can be quicker than SQL databases, and it can be easier

to share the load across many machines. They tend to process data using the **map-reduce** method, which goes through each item in turn and maps it to a value. These values can then be combined (or reduced) to a result.

The data set we've used here is 19 million items big. We've certainly heard people calling much more mundane analyses than this big data, but in our view, it doesn't qualify. *MySQL* handles the task perfectly well, and it's a technology that's far more useful in most circumstances than NoSQL.

However, if you happen to be in the job market at the moment, NoSQL is one of the hottest skills around (according to www.indeed.com/jobtrends, *MongoDB* – a NoSQL database – is the second hottest skill to have after HTML5).

prices for a few different counties, you could use:

```
for county in ['GREATER MANCHESTER','GLOUCESTERSHIRE']:
    query = "select avg(price) from house_prices where
address7 = " + county + ";"
    cur.execute(query)
    result = cur.fetchone()
    print "Average house price in " + county + " : " +
str(result[0])
```

Alternatively, you could see how the house prices have changed over the 20 years we have data for using the following. You'll need to include the previous code to connect to the database as well.

```
years = range(1995, 2015)
data = []

for year in years:
    query = 'select avg(price) from house_prices where
data between "' + str(year) + '-01-01" and "' + str(year) +
'-12-31";'
    cur.execute(query)
    result = cur.fetchone()
    print str(year) + " : " + str(result[0])
    data.append(int(result[0]))
```

If you're an SQL user, you'll probably notice that this could be done in a single query. We've done it this way to make the code a bit easier to follow.

This code stores the data in a list as well as printing it on a screen. This list (rounded to whole numbers), can be used to create graphs. One option is to output it to a file in CSV format. CSVs can be loaded into most spreadsheets (such as *LibreOffice Calc*), and from there you can generate any graphics you need. This can be a good way to experiment with different types of graph, because it enables you to quickly try various visualisations on the data. However, it's bad if you need to produce lots of graphs based on the data, because it requires quite a bit of manual intervention. For this, it's much easier to use the *Matplotlib* module to automatically draw any charts you want.

Get Matplotlib

To use this, you'll need to import it. We'll pull it in with *pylab*, which provides some other functions as well as chart drawing. You'll need to add the following to the start of your program:

```
from pylab import *
```

The following two lines can then be added to the end of the previous program to plot the data, and show the chart:

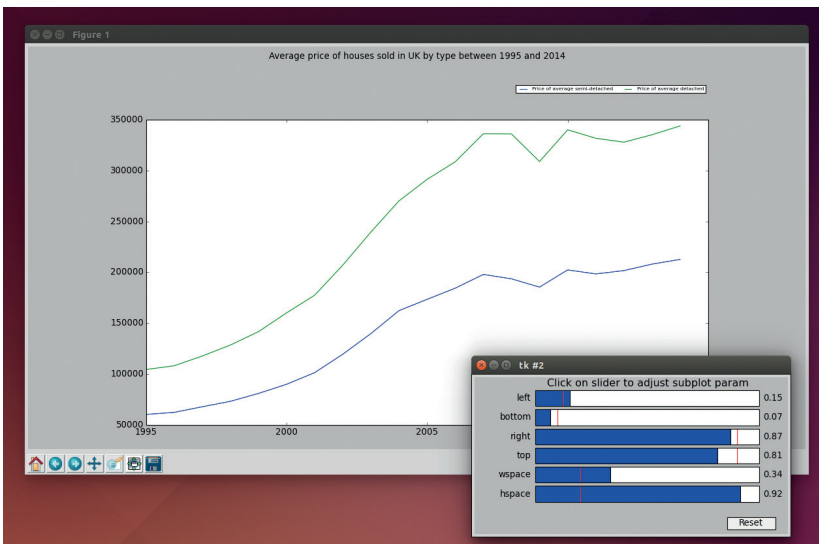
```
plot(years, data)
show()
```

This is the most basic use of the plotting module, and it can do far more than this. Let's take a look at a slightly more complicated example. This time, we'll see how the average price of houses has changed for detached and semi-detached houses. First we need to pull the appropriate information from the database with the following code (this will also need the code to connect to the database):

```
def get_value(cur, query):
    cur.execute(query)
    row = cur.fetchone()
    return int(row[0])

val_of_semi = []
val_of_detached = []
years = range(1995, 2015)
for year in years:
    query = 'select avg(price) from house_prices where
data between "' + str(year) + '-01-01" and "' + str(year) + '-12-31"
and type="S";'
    val_of_semi.append(get_value(cur, query))
```

You can change some parameters of the figure after it's created using the **Configure Subplots** button (second from the right).



```
query = 'select avg(price) from house_prices where
data between "' + str(year) + '-01-01' and "' + str(year) + '-12-31'
and type="D";'
```

```
val_of_detached.append(get_value(cur, query))
```

Now you have two lists; you just need to put them in the plot. The following code does this:

```
fig = figure()
```

```
fig.set_size_inches(10,4,forward=True)
```

```
ax = subplot(111)
```

```
box = ax.get_position()
```

```
ax.set_position([box.x0, box.y0, box.width, box.height*0.80])
```

```
semi_line = ax.plot(years, val_of_semi, label="Price of average
semi-detached")
```

```
detached_line = ax.plot(years, val_of_detached, label="Price of
average detached")
```

```
ax.legend(bbox_to_anchor=(0., 1.02, 1., .102), ncol=2,
prop=("size":7))
```

```
suptitle('Average price of houses sold in UK by type between
1995 and 2014')
```

```
show()
```

First, this code creates a figure, and resizes it to 1000 pixels by 400 pixels (it defaults to 100 pixels per inch). The parameter **forward=True** allows you to re-size the window.

Instead of just calling **plot()** like we did in the previous example, this time we create a subplot and shrink it down to 80% of its original height. This gives us space to put a title and legend above it.

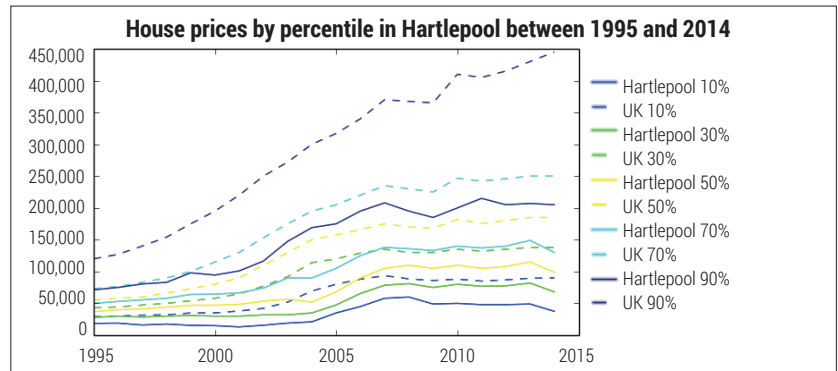
The value returned by **plot()** is a line object that we can manipulate to alter the way the line will be displayed. Although we don't do it in this example, you can use this object to alter the way they're displayed.

For example the following (placed before **show()**) would make the lines red and green (by (r,g,b) values),

Data sources

There are loads of other sources of data that are crying out for analysis. Here are a few places to start looking:

- **Data.gov.uk** The official source of all UK government data (this is where the housing data for this article comes from).
- **www.data.gov** The US government's data sets.
- **bitly.com/bundles/bigmlcom/i** A bundle of links to the data websites for many governments from around the world.
- **data.worldbank.org** The world bank publishes financial data on the state of the world economy.
- **epp.eurostat.ec.europa.eu** Eurostat is the directorate general of the European Commission, and is responsible for compiling and publishing statistics about the European Union.
- **www.eea.europa.eu/data-and-maps** The European Environment Agency publishes a lot of data about the state of Europe.
- **aws.amazon.com/datasets** A list of some of the most popular data sets from around the world.
- **www.reddit.com/r/datasets** A subreddit dedicated to seeking out data on all topics.



and dashed (**linestyle "--"**).

```
setp(semi_line, "color", (1,0,0))
```

```
setp(detached_line, "color", (0,1,0))
```

```
setp((semi_line, detached_line), "linestyle", "--")
```

Other line styles are "-" (solid line), "." (dotted), and "-." (dash-dot). You can also use **setp** to change the alpha (transparency) settings. In fact, there is a mind-boggling set of different options you can set to make the graph look exactly how you want. If you want to create your own graphs, it's best to spend a little time perusing the set of examples at <http://matplotlib.org/gallery.html> to see what's available.

Once you've got everything for the subplot organised, you need to make sure your graph is labelled properly. Adding a title is easy, as you can see in the above call to **suptitle()**. Adding a legend is a bit more complex, because positioning in *Matplotlib* is something of a dark art.

If you want to save figures rather than just displaying them, you can use:

```
savefig('filename')
```

There are loads of ways you can drill down to almost any level of detail, and pull out whatever you want. Of course, this does require an ability to program, and the time to do it.

The end goal, of course, isn't to draw pretty pictures, but to get a better understanding of what the data means. In this case, we've been looking at how the prices of houses have changed over the past 20 years. We won't tell you exactly how to do this because it would defeat the point of this tutorial (which is to learn how to analyse the data for yourself), but we looked into how the house prices changed across different locations and different values of house.

You can see our results at www.linuxvoice.com/house-price-analysis. This challenges the view that house prices are rising in the UK. In fact, our analysis shows that in most places house prices are quite static, but that rapid rises in London are pushing the average price up across the UK, distorting the picture. Don't take our word for it though. Dive into the data and see what it tells you. 📊

Ben Everard is the co-author of the best-selling *Learn Python With Raspberry Pi*, and is working on a best-selling follow-up called *Learning Computer Architecture With Raspberry Pi*.

Hartlepool (among other towns and cities) hasn't seen the same rise in house prices as south-eastern England. See www.linuxvoice.com/house-price-analysis for the rest of our analysis.

LINUX 101: POWER UP YOUR SHELL

MIKE SAUNDERS

Get a more versatile, featureful and colourful command line interface with our guide to shell basics.

WHY DO THIS?

- Make life at the shell prompt easier and faster.
- Resume sessions after losing a connection.
- Stop pushing around that fiddly rodent!

As a Linux user, you're probably familiar with the shell (aka command line). You may pop up the occasional terminal now and then for some essential jobs that you can't do at the GUI, or perhaps you live in a tiling window manager environment and the shell is your main way of interacting with your Linux box.

In either case, you're probably using the stock *Bash* configuration that came with your distro – and while

it's powerful enough for most jobs, it could still be a lot better. In this tutorial we'll show you how to pimp up your shell to make it more informative, useful and pleasant to work in. We'll customise the prompt to make it provide better feedback than the defaults, and we'll show you how to manage sessions and run multiple programs together with the incredibly cool *tmux* tool. And for a bit of eye candy, we'll look at colour schemes as well. So, onwards!

1 MAKE YOUR PROMPT SING

Most distributions ship with very plain prompts – they show a bit of information, and generally get you by, but the prompt can do so much more. Take the default prompt on a Debian 7 installation, for instance:

```
mike@somebox:~$
```

This shows the user, hostname, current directory and account type symbol (if you switch to root, the **\$** changes to **#**). But where is this information stored? The answer is in the **PS1** environment variable. If you enter **echo \$PS1** you'll see this at the end of the text string that appears:

```
\u@\h:\w\$
```

This looks a bit ugly, and at first glance you might start screaming, assuming it to be a dreaded regular expression, but we're not going to fry our brains with the complexity of those. No, the slashes here are escape sequences, telling the prompt to do special

things. The **\u** part, for instance, tells the prompt to show the username, while **\w** means the working directory.

Here's a list of things you can use in the prompt:

- **\d** The current date.
- **\h** The hostname.
- **\n** A newline character.
- **\A** The current time (HH:MM).
- **\u** The current user.
- **\w** (lowercase) The whole working directory.
- **\W** (uppercase) The basename of the working directory.
- **\\$** A prompt symbol that changes to **#** for root.
- **!** The shell history number of this command.

To clarify the difference in the **\w** and **\W** options: with the former, you'll see the whole path for the directory in which you're working (eg **/usr/local/bin**), whereas for the latter it will just show the **bin** part.

Here's our souped-up prompt on steroids. It's a bit long for this small terminal window, but you can tweak it to your liking.

```
mike@debianmike: ~
File Edit Tabs Help
mike@debianmike:~$ # This prompt is rather boring, isn't it?
mike@debianmike:~$ # Let's spice it up by modifying the PS1 variable...
mike@debianmike:~$ export PS1="(?!)\[\e[31m\][\A] \[\e[32m\]\u@\h \[\e[34m\]\w \[\e[30m\]\$ "
(140) [11:00] mike@debianmike ~ $ cd /usr/local/include
(141) [11:00] mike@debianmike /usr/local/include $ # Now that's a lot better!
(142) [11:01] mike@debianmike /usr/local/include $ # Now our prompt has colour
(143) [11:01] mike@debianmike /usr/local/include $ # Along with history nums
(144) [11:01] mike@debianmike /usr/local/include $ # A clock
(145) [11:01] mike@debianmike /usr/local/include $ # And better spacing :-)
```

Get customising

Now, how do you go about changing the prompt? You need to modify the contents of the **PS1** environment variable. Try this:

```
export PS1="I am \u and it is \A \$"
```

Now your prompt will look something like:

```
I am mike and it is 11:26 $
```

From here you can experiment with the other escape sequences shown above to create the prompt of your dreams. But wait a second – when you log out, all of your hard work will be lost, because the value of the **PS1** environment variable is reset each time you start a terminal. The simplest way to fix this is to open the **.bashrc** configuration file (in your home directory) and add the complete export command to the bottom. This **.bashrc** file will be read by *Bash* every time you start a new shell session, so your beefed-up

prompt will always appear. You can also spruce up your prompt with extra colour. This is a bit tricky at first, as you have to use some rather odd-looking escape sequences, but the results can be great. Add this to a point in your **PS1** string and it will change the text to red:

```
\[\e[31m\]
```

You can change 31 here to other numbers for different colours:

- 30 Black
- 32 Green
- 33 Yellow
- 34 Blue
- 35 Magenta
- 36 Cyan
- 37 White

So, let's finish off this section by creating the mother of all prompts, using the escape sequences and colours we've already looked at. Take a deep breath, flex your fingers, and then type this beast:

```
export PS1="(❗) \[\e[31m\][\A] \[\e[32m\]\u@\h \[\e[34m\]\w \[\e[30m\]\$ "
```

This provides a *Bash* command history number, current time, and colours for the user/hostname

Shell essentials

If you're totally new to Linux and have just picked up this magazine for the first time, you might find the tutorial a bit heavy going. So here are the basics to get you familiar with the shell. It's usually found as *Terminal*, *XTerm* or *Konsole* in your menus, and when you start it the most useful commands are:

ls (list files); **cp one.txt two.txt** (copy file); **rm file.txt** (remove file); **mv old.txt new.txt** (move or rename); **cd /some/directory** (change directory); **cd ..** (change to directory above); **./program** (run program in current directory); **ls > list.txt** (redirect output to a file).

Almost every command has a manual page explaining options (eg **man ls** – press Q to quit the viewer). There you can learn about command options, so you can see that **ls -la** shows a detailed list including hidden files. Use the up and down cursor keys to cycle through previous commands, and use Tab after entering part of a file or directory name to auto-complete it.

combination and working directory. If you're feeling especially ambitious, you can change the background colours as well as the foreground ones, for really striking combinations. The ever useful Arch wiki has a full list of colour codes: <http://tinyurl.com/3gvz4ec>.

2 TMUX: A WINDOW MANAGER FOR YOUR SHELL

A window manager inside a text mode environment – it sounds crazy, right? Well, do you remember when web browsers first implemented tabbed browsing? It was a major step forward in usability at the time, and reduced clutter in desktop taskbars and window lists enormously. Instead of having taskbar or pager icons for every single site you had open, you just had the one button for your browser, and then the ability to switch sites inside the browser itself. It made an awful lot of sense.

If you end up running several terminals at the same time, a similar situation occurs; you might find it annoying to keep jumping between them, and finding the right one in your taskbar or window list each time. With a text-mode window manager you can not only run multiple shell sessions simultaneously inside the same terminal window, but you can even arrange them side-by-side.

And there's another benefit too: detaching and reattaching. The best way to see how this works is to try it yourself. In a terminal window, enter **screen** (it's installed by default on most distros, or will be available in your package repositories). Some welcome text appears – just hit Enter to dismiss it. Now run an interactive text mode program, such as **nano**, and close the terminal window.

In a normal shell session, the act of closing the window would terminate every process running inside it – so your *Nano* editing session would be a goner. But not with *screen*. Open a new terminal and enter:

```
screen -r
```

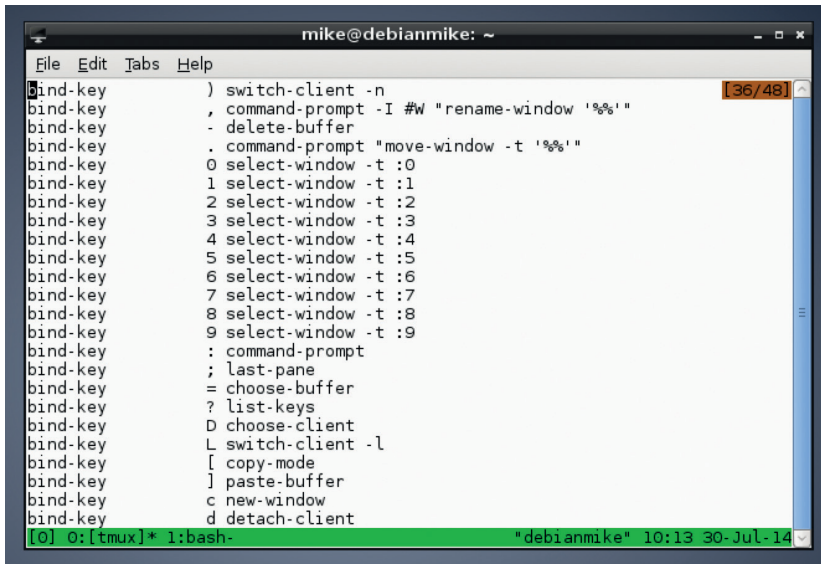
And *voilà*: the *Nano* session you started before is back!

When you originally ran **screen**, it created a new shell session that was independent and not tied to a specific terminal window, so it could be detached and reattached (hence the **-r** option) later.

This is especially useful if you're using SSH to connect to another machine, doing some work, and don't want a flaky connection to ruin all your progress. If you do your work inside a **screen** session and your connection goes down (or your laptop battery dies, or your computer explodes), you can simply reconnect/recharge/buy a new computer, then SSH back in to the remote box, run **screen -r** to reattach and carry on from where you left off.

Here's **tmux** with two panes open: the left has Vim editing a configuration file, while the right shows a manual page.

```
mike@debianmike: ~
File Edit Tabs Help
1 Package generated configuration file
2 # See the sshd_config(5) manpage for details
3
4 # What ports, IPs and protocols we listen for
5 Port 22
6 # Use these options to restrict which interfaces/prot
ocols sshd will bind to
7 #ListenAddress ::
8 #ListenAddress 0.0.0.0
9 Protocol 2
10 # HostKeys for protocol version 2
11 HostKey /etc/ssh/ssh_host_rsa_key
12 HostKey /etc/ssh/ssh_host_dsa_key
13 HostKey /etc/ssh/ssh_host_ecdsa_key
14 #Privilege Separation is turned on for security
15 UsePrivilegeSeparation yes
16
17 # Lifetime and size of ephemeral version 1 server key
18 KeyRegenerationInterval 3600
19 ServerKeyBits 768
20
21 # Logging
22 SyslogFacility AUTH
23 LogLevel INFO
24
25 # Authentication:
26 LoginGraceTime 120
27 PermitRootLogin yes
28 StrictModes yes
29
30 RSAAuthentication yes
31 PubkeyAuthentication yes
32 #AuthorizedKeysFile  %h/.ssh/authorized_keys
33
sshd config 1.1 Top
:syntax on
[0] 0:vim* 1:bash-
d_config(5) line 1 (press h for help or q to quit)
"debianmike" 10:11 30-Jul-14
```



In **tmux**, hit **Ctrl+B** followed by **?** to get a list of the default key bindings.

Now, we've been talking about GNU *screen* here, but the title of this section mentions *tmux*. Essentially, *tmux* (terminal multiplexer) is like a beefed up version of *screen* with lots of useful extra features, so we're going to focus on it here. Some distros include *tmux* by default; in others it's usually just an **apt-get**, **yum install** or **pacman -S** command away.

Multiplexing magic

Once you have it installed, enter **tmux** to start it. You'll notice right away that there's a green line of information along the bottom. This is very much like a

taskbar from a traditional window manager: there's a list of running programs, the hostname of the machine, a clock and the date. Now run a

program, eg *Nano* again, and hit **Ctrl+B** followed by **C**. This creates a new window inside the *tmux* session, and you can see this in the taskbar at the bottom:

```
O:nano- 1:bash*
```

Each window has a number, and the currently displayed program is marked with an asterisk symbol. **Ctrl+B** is the standard way of interacting with *tmux*, so if you hit that key combo followed by a window number, you'll switch to that window. You can also use **Ctrl+B** followed by **N** and **P** to switch to the next and previous windows respectively – or use **Ctrl+B** followed by **L** to switch between the two most recently used windows (a bit like the classic **Alt+Tab** behaviour on the desktop). To get a window list, use **Ctrl+B** followed by **W**.

So far, so good: you can now have multiple programs running inside a single terminal window, reducing clutter (especially if you often have multiple SSH logins active on the same remote machine). But what about seeing two programs at the same time?

For this, *tmux* uses "panes". Hit **Ctrl+B** followed by **%** and the current window will be split into two sections,

one on the left and one on the right. You can switch between them Using **Ctrl+B** followed by **O**. This is especially useful if you want to see two things at the same time – eg a manual page in one pane, and an editor with a configuration file in another.

Sometimes you'll want to resize the individual panes, and this is a bit trickier. First you have to hit **Ctrl+B** followed by **:** (colon), which turns the *tmux* bar along the bottom into a dark orange colour. You're now in command mode, where you can type in commands to operate *tmux*. Enter **resize-pane -R** to resize the current pane one character to the right, or use **-L** to resize in a leftward direction. These may seem like long commands for a relatively simple operation, but note that the *tmux* command mode (started with the aforementioned colon) has tab completion. So you don't have to type the whole command – just enter **"resi"** and hit **Tab** to complete. Also note that the *tmux* command mode also has a history, so if you want to repeat the resize operation, hit **Ctrl+B** followed by colon and then use the up cursor key to retrieve the command that you entered previously.

Finally, let's look at detaching and reattaching – the awesome feature of *screen* we demonstrated earlier. Inside *tmux*, hit **Ctrl+B** followed by **D** to detach the current *tmux* session from the terminal window, which leaves everything running in the background. To reattach to the session use **tmux a**. But what happens if you have multiple *tmux* sessions running? Use this command to list them:

tmux ls

This shows a number for each session; if you want to reattach to session 1, use **tmux a -t 1**. *tmux* is hugely configurable, with the ability to add custom keybindings and change colour schemes, so once you're comfortable with the main features, delve into the manual page to learn more.

Zsh: an alternative shell

Choice is good, but standardisation is also important as well. So it makes sense that almost every mainstream Linux distribution uses the *Bash* shell by default – although there are others. *Bash* provides pretty much everything you need from a shell, including command history, filename completion and lots of scripting ability. It's mature, reliable and well documented – but it's not the only shell in town.

Many advanced users swear by *Zsh*, the Z Shell. This is a replacement for *Bash* that offers almost all of the same functionality, with some extra features on top. For instance, in *Zsh* you can enter **ls -** and hit **Tab** to get quick descriptions of the various options available for **ls**. No need to open the manual page!

Zsh sports other great auto-completion features: type **cd /u/lo/bi** and hit **Tab**, for instance, and the full path of **/usr/local/bin** will appear (providing there aren't other paths containing **u**, **lo** and **bi**). Or try **cd** on its own followed by **Tab**, and you'll see nicely coloured directory listings – much better than the plain ones used by *Bash*.

Zsh is available in the package repositories of all major distros; install it and enter **zsh** to start it. To change your default shell from *Bash* to *Zsh*, use the **chsh** command. And for more information visit www.zsh.org.

Fine-tune your colour scheme

We're not obsessed with eye-candy at Linux Voice, but we do recognise the importance of aesthetics when you're staring at something for several hours every day. Many of us love to tweak our desktops and window managers to perfection, crafting pixel-perfect drop shadows and fiddling with colour schemes until we're 100% happy. (And then fiddling some more out of habit.)

But then we tend to ignore the terminal window. Well, that deserves some love too, and at <http://ciembor.github.io/4bit> you'll find a highly awesome colour scheme designer that can export settings for all of the popular terminal emulators (*XTerm*, *Gnome Terminal*, *Konsole* and *Xfce4 Terminal* are among the apps supported.) Move the sliders until you attain colour scheme nirvana, then click on the Get Scheme button at the top-right of the page.

Similarly, if you spend a lot of time in a text editor such as *Vim* or *Emacs*, it's worth using a well-crafted palette there as well. **Solarized** at <http://ethanschoonover.com/solarized> is an excellent scheme that's not just pretty, but designed for maximum usability, with plenty of research and testing behind it.

```
1 #!/bin/bash~
2 ~
3 cd $ROOT_DIR
4 DOT_FILES="lastpass weechat ssh Xauthority"~
5 for dotfile in $DOT_FILES; do conform_link "$DATA_DIR/$dotfile" ".$dotfile"; dor
6 ~
7 # } } }~
8 # crontab update from file { { {~
9 # TODO: refactor with suffix variables (or common cron values)~
10 ~
11 case "$PLATFORM" in
12     linux)~
13         #conform_link "$CONF_DIR/shell/zshenv" ".zshenv"~
14         crontab -l > $ROOT_DIR/tmp/crontab-conflict-arch~
15         cd $ROOT_DIR/$CONF_DIR/cron~
16         if [[ "$(diff -/tmp/crontab-conflict-arch crontab-current-arch)" == "" ]~
17             ]];~
18         then # no difference with current backup~
19             logger "$LOG_PREFIX: crontab live settings match stored "\
20                 "settings; no restore required"~
21             rm -/tmp/crontab-conflict-arch~
```

The Solarized colour scheme might not look so swish on paper, but it works brilliantly on the screen to reduce eye strain during long coding sessions.

3 THE TERMINALS OF THE FUTURE


You might be wondering why the application that contains your command prompt is called a terminal. Back in the early days of Unix, people tended to work on multi-user machines, with a giant mainframe computer occupying a room somewhere in a building, and people connected to it using screen and keyboard combinations at the end of some wires. These terminal machines were often called “dumb”, because they didn't do any important processing themselves – they just displayed whatever was sent down the wire from the mainframe, and sent keyboard presses back to it.

Today, almost all of us do the actual processing on our own machines, so our computers are not terminals in a traditional sense. This is why programs like *XTerm*, *Gnome Terminal*, *Konsole* etc. are called “terminal emulators” – they provide the same facilities as the physical terminals of yesteryear. And indeed, in many respects they haven't moved on much. Sure, we

have anti-aliased fonts now, better colours and the ability to click on URLs, but by and large they've been working in the same way for decades.

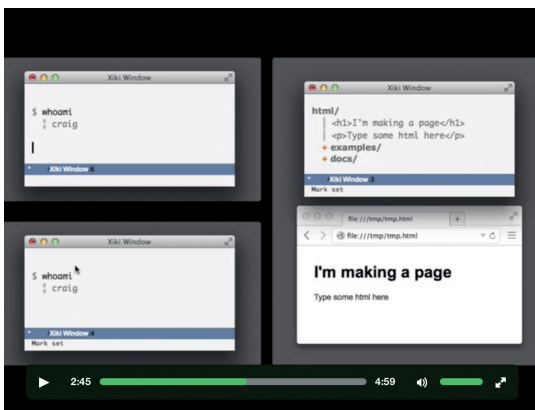
Some programmers are trying to change this though. *Terminology* (<http://tinyurl.com/osopjv9>), from the team behind the ultra-snazzy *Enlightenment* window manager, aims to bring terminals into the 21st century with features such as inline media display. You can enter **ls** in a directory full of images and see thumbnails, or even play videos from directly inside your terminal. This makes the terminal work a bit more like a file manager, and means that you can quickly check the contents of media files without having to open them in a separate application.

Then there's *Xiki* (www.xiki.org), which describes itself as “the command revolution”. It's like a cross between a traditional shell, a GUI and a wiki; you can type commands anywhere, store their output as notes for reference later, and create very powerful custom commands. It's hard to describe it in mere words, so the authors have made a video (see the Screencasts section of the *Xiki* site) which shows how much potential it has.

And *Xiki* is definitely not a flash in the pan project that will die of bitrot in a few months. The authors ran a successful Kickstarter campaign to fund its development, netting over \$84,000 at the end of July. Yes, you read that correctly – \$84K for a terminal emulator. It might be the most unusual crowdfunding campaign since some crazy guys decided to start their own Linux magazine... 

LV PRO TIP

Many command line and text-based programs match their GUI equivalents for feature parity, and are often much faster and more efficient to use. Our recommendations: *Irssi* (IRC client); *Mutt* (mail client); *rTorrent* (BitTorrent); *Ranger* (file manager); *htop* (process monitor). *ELinks* does a decent job for web browsing, given the limitations of the terminal, and it's useful for reading text-heavy websites such as Wikipedia.



Xiki aims to be both a more welcoming shell for new users, and a step-up for experienced CLIs.

Mike Saunders remembers using a mouse once. On the Amiga. Now he just wants kids to get off his damn lawn.

FARGO: WRITE AND PUBLISH OUTLINES IN OPEN FORMATS

MARCO FIORETTI

Turn the web upside down with this text outliner – without installing a single piece of software.

WHY DO THIS?

- Prepare yourself for the open, distributed web of tomorrow, in an easy and fun way.
- Publish a nice-looking personal blog for free, in five minutes, without installing anything.
- Get familiar with OPML. You may need it again some day. Trust us.

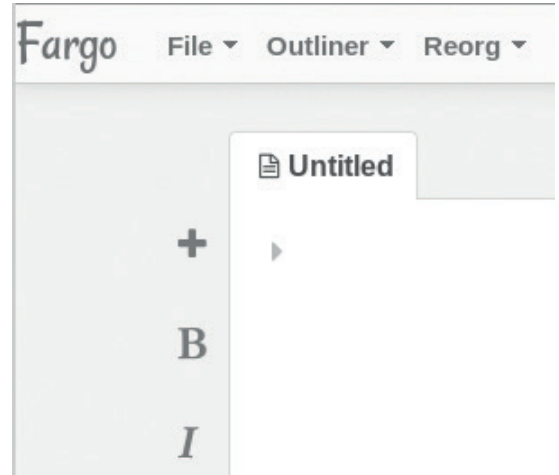
When you look at it closely, much written text has the same basic structure. Newspaper articles, philosophy essays, novel summaries, courseware, recipes... are all outlines – that is, hierarchical trees of topics and sub topics. If this is true, the more a software editor takes it into account, the more efficient it is, right?

Fargo is an outliner – that is a text editor designed to handle outlines in the most efficient way. Any outliner program provides tools to quickly navigate the elements of an outline and rearrange them at will, with the smallest possible effort. Above all, outliners can instantly hide certain levels or branches of an outline, so that at any moment you only see the exact amount of content and level of detail that you want to see.

Outliners are nothing new. In fact, the real value of *Fargo* is not in what it does, but in how and where it does it. This tutorial explains how *Fargo* works and how to use it, mainly from the point of view of a Linux user who would like to integrate it with their other online and desktop activities.

Now, the *Fargo* user interface is deceptively simple. It's easy to find the menus and buttons that perform an action, but to work with this tool (rather than against it) you must first understand the *Fargo* philosophy and what it does under the hood. We may even say that assimilating where the hood is is the hardest part here. Consequently, we will devote more space to explaining what *Fargo* provides, and how, than to explain how to actually use single menus or panels.

There are three points that were the origin, and still are at the core, of the *Fargo* proposal. The first is the observation that modern JavaScript-capable browsers are very powerful and run on hardware, even including mobile devices, much more powerful than



Eye candy and formatting functions in *Fargo* are limited to the bare minimum, and there are two powerful menus for viewing and managing outlines.

10 or even just five years ago. No question about that, but the other points deserve more reflection.

Fargo also works on the assumption that today “the cloud is ubiquitous and reliable” (not to mention, we may add, affordable). Residents in rural areas of Western countries, plus almost everybody else, may disagree on this. The final point is about lock in and... let's discuss it at the end of the tutorial.

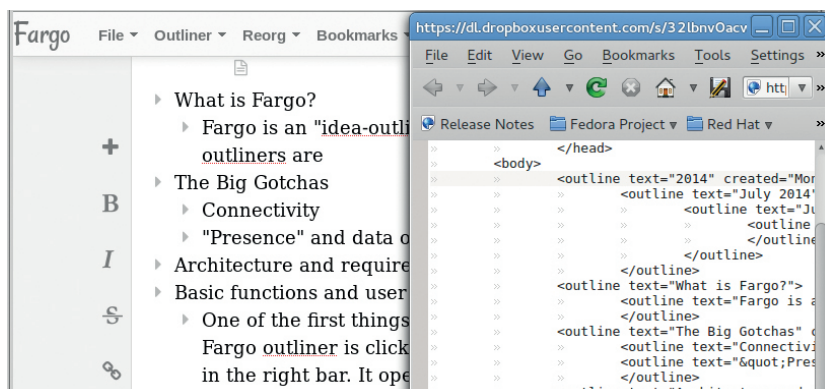
Fargo architecture and requirements

Installing *Fargo* is really simple: there is nothing to install! The only requirements are a browser that can handle JavaScript and HTML5 and a Dropbox account. Log in to Dropbox, point your browser to <http://fargo.io> and accept the request to let the *Fargo* app work in a dedicated subfolder of your account. If you don't see that request, it is because you've already been there. Tell your browser to erase all the cookies from the **fargo.io** domain and reload.

Thirty seconds later, you will be able to start writing outlines and publishing them online using an interface, and with a final result, already close to what you could get at **Tumblr.com** or **WordPress.com**, but without the lock-in.

This happens because, while *Fargo* is a static JavaScript app that runs entirely in the browser of your own computer or smartphone, it behaves as if it were a traditional CMS engine and produces the same results: you can always write and archive outlines in the same way from any device and location. From the

We see two very important things here: first, text written in *Fargo* looks very clean and easy on one's eyes. Second, that all your works remain available in open source formats.



viewer's point of view it's the same too: everybody can access all the outlines that you made public as if they were a traditional website. *Fargo* can also generate static HTML versions of your outlines and upload them to a web server whenever you want.

This is why *Fargo* has the potential to turn the web upside down. The current model of web self publishing and working "in the cloud" is based on central CMS servers doing all the really heavy work, from database queries to page rendering, for many thousands of authors and visitors simultaneously. This architecture demands servers and data centres with very high costs and environmental impacts.

In the *Fargo* model, as much as possible is decentralised. Only sensible data such as passwords are stored in your device. Raw outlines are still stored on servers; that is, in a private folder of your Dropbox account, but all the processing happens in the browsers that run *Fargo*, or in those that display its static HTML pages.

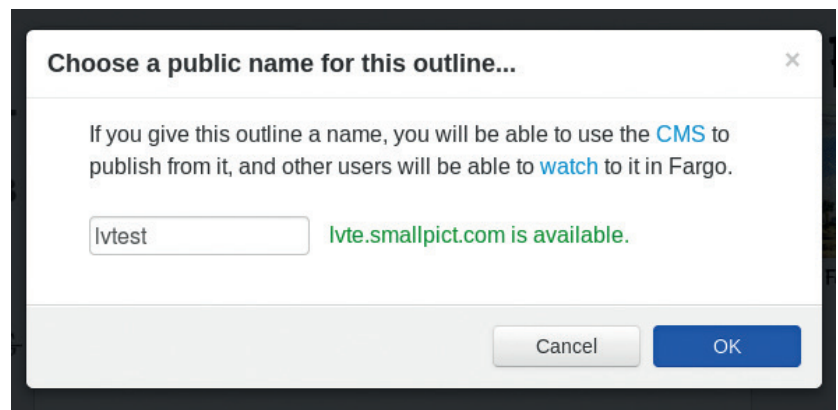
For authors, *Fargo* has another big advantage on server-based publishing systems like WordPress: since all the CMS logic runs in a browser, it can have a much more flexible and responsive interface, and provide a structure that naturally matches the structure of most writing.

Structure of Fargo content

At the low level, each *Fargo* outline is a separate OPML file stored in your Dropbox account (see the OPML box below to understand what OPML is, and why it is great regardless of *Fargo*). Using Dropbox as filesystem also provides automated backups and versioning for free, even if you still have to backup everything outside Dropbox regularly.

The single elements of each outline, which can be nested at will, are called headlines (or even summits, if at the top level). We would have preferred terms like node or paragraph, because each *Fargo* headline can be as long as you want, and each time you press Enter, you create a new one, but headline it is.

Besides its unique position in the hierarchy, which of course you can change as you want, each headline can have attributes like identification code, creation



Fargo would be very useful even as a purely private editor, but it couldn't be easier to transform content in to a blog.

date or author-defined data, or be commented out. In the latter case, the headline will remain in the OPML file, but out of sight, and it will never be included in the HTML versions. We will explain how to comment or assign attributes in a moment.

All your Dropbox files are private, until you ask *Fargo* to create public, but read-only links to them. An outline can even embed content from external websites, if you pass them to *Fargo* with the browser Bookmarklet linked from the right bar.

When an outline grows unwieldy, you can archive all its headlines that you don't need to edit anymore, and still make them show up (and render) in the outline. To do that, you have to archive those headlines as "includes". Do do this, place the cursor on them, select File > Archive Cursor, and they will be moved to the **archive** sub-folder of your *Fargo* folder in Dropbox.

Images and interactive content? Of course!

In case you were worrying that a system optimised for outlines doesn't support anything but static text, relax! You can tell *Fargo* to keep an eye on a Dropbox subfolder for generic media (images, audio, PDFs, whatever). Then, any time you upload something there, *Fargo* will notice it and give you a URL for it in a pop-up window. You can add as much interactivity as you want to your *Fargo* outlines... as long as you write

LV PRO TIP

If you have a lot of texts on your hard drive that you would like to import in *Fargo*, don't worry. It's very likely, you can automate much of that work. One of the best tools for the is *Pandoc* (<http://johnmacfarlane.net/pandoc>): a very versatile converter that can transform any of dozens of formats into any of the others.

What is OPML?

Really open file formats and communication standards are arguably even more important than free software. If somebody else sends you files in one of those formats, you can merrily ignore if they use proprietary software, and open those files with whatever application you prefer, directly on Linux.

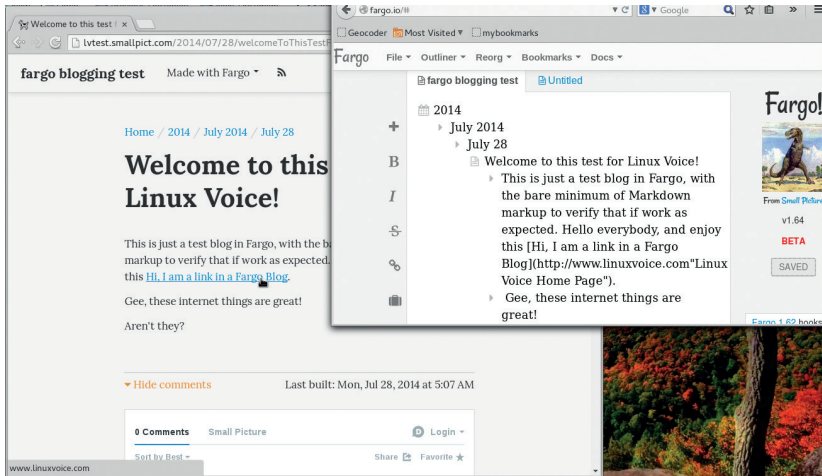
In the open formats family, the Outline Processor Markup Language (OPML – <http://dev.opml.org/spec2.html>), was developed specifically to process and exchange outlines. You have probably already seen it, or at least one of its applications: the list of links on the side of many websites known as "blogrolls" are just that: outlines that under the hood are most probably OPML files.

The most frequent application of OPML, at least on the web, is the automatic exchange of lists of RSS feeds between the

websites and software programs that generate, process and syndicate such feeds.

From a technical point of view, OPML is nothing other than another application of XML. In practice, this means that an OPML file, while terribly verbose, is just plain text that you can generate, parse and process automatically with many free software tools, from custom scripts to specialised editors.

It is equally important to realise that there's nothing to limit OPML to handling lists of headlines and relative links and abstracts. Formally speaking, OPML can handle anything whose structure is a hierarchic tree of nodes, each containing named attributes in text format. If you think about it for just a moment, you will realise that even your family tree, or your company's organisation chart, match this description.



What looks just like very well structured text, automatically becomes a simple blog, complete with Disqus comments, with just a few clicks.

it in JavaScript, as *Fargo* itself. In general, the developers have already started to think about JavaScript “verbs” for *Fargo* that would make such tasks easier. See <http://docs.fargo.io/fargoScripting/> for details.

You can already add snippets of JavaScript to a headline (including calls to internal *Fargo* functions) and run them by pressing `Ctrl+.` It is also possible to run some JavaScript code automatically, every time you reload *Fargo* or publish an outline.

The Fargo user interface

The first thing you want to do in your *Fargo* outliner is click on ‘Cribsheet’ in the right sidebar, to get a cheatsheet with all the main commands. Next, you should take a look at the many resources in the Docs top menu. Just remember that whenever those pages say “Cmd”, (as in the Command key on OS X) what they mean on Linux is the Control key.

Now, let’s talk configuration. To access the *Fargo* configuration tabs, click on your name in the top-right corner of the browser window and select “Settings”. Besides a multimedia folder here, you can set a password to encrypt all your private outlines, the

Integration with WordPress

Many bloggers simply cannot give up their WordPress accounts for *Fargo*, because they need some special plugin or, much more simply, they are just (co-)authors, not the owners of those blogs. What should they do, if they find the *Fargo* authoring interface much better than the WordPress one? Post to WordPress from *Fargo*, of course (only one blog per *Fargo* account, sorry!). The “Blog” tab of the *Fargo* settings interface is there just for that purpose: enter the URL of your blog, your username and password.

If you need to format your blog posts in ways that *Fargo* doesn’t support, just check the Markdown box, and all the markup you add in your headlines will automatically be converted to HTML before sending it to the blog. After this initial configuration, every time you want to post create a new headline for the title, another right below it with the content, and click on the WordPress icon in the left sidebar.

autosave behaviour and your contact information (profile page, email, Twitter and Facebook accounts). In the same place, you can define separate CSS styles for each level of your outlines, or a background image.

Fargo can handle multiple outlines simultaneously, each in a different tab. The editor marks each headline with a wedge on the left, which will be black if there is unexpanded test underneath it, or grey otherwise. The actual content of a headline can be formatted with HTML or Markdown syntax.

Setting the standard attributes of a headline, or giving it custom ones, is easy: select the headline, click on the suitcase icon (or select Outliner > Edit Attributes) and enter the attribute name on the left and its value on the right. Click on the + button if you also need to add custom attributes, and repeat. Headlines can also be individually commented by pressing `Ctrl+\`. When you do that, their wedges will become chevrons. To uncomment them, press `Cmd+\` again.

Working with Fargo

Looking at *Fargo* as just an editor, its two main features are the ‘Outliner’ and ‘Reorg’ top menus. The first is used to control how much you see of the current outline and toggle between Non-Render and Render mode: use the first mode to write or edit raw markup inside an outline, and the other to see what the results looks like.

As the name suggests, the ‘Reorg’ menu helps to reorganise your writings. The entries to move one or more headlines up or down the outline they are in, or to change their indentation levels, are all there.

The main functions found in both those top menus are also available in ‘Pad’ format, to work faster on touchscreen devices. The *Fargo* ‘Arrow Pad’ (Outliner > Show Arrow Pad) has two buttons, one to collapse or expand parts of the hierarchy, and the other to toggle Navigate and Reorg mode. Depending on the mode the four arrow icons in the pad will let you move headlines around, or navigate from one to another.

On devices with real keyboards, you can use shortcuts for almost all menu entries. Tab and Shift+Tab, for example, increase and decrease the indentation level of a headline. Remember that in *Fargo* pressing the Enter key does not enter a newline, or split the current text in two. It just add one more empty headline below the current one, regardless of where the cursor was when you typed.

Once you have acquired familiarity with the outline-oriented interface of *Fargo*, you will also be able to use it to build a public, simple blog. The post shown in the image above-left was created in four main steps (the extra, really simple details are all at <http://fargo.io/docs/blogging/firstPost.html>):

- 1 Create a new outline (File > New).
- 2 Give it a name (File > Name Outline), let’s say ‘goinuxvoice’.
- 3 Write some content in the usual way.
- 4 When you are done, put the cursor on the top

headline, and click the Eye icon in the left bar.

The last action will create a new subdomain, **glinuxvoice.smallpict.com** (Small Pict is the company that develops Fargo). All visitors of that domain will be transparently redirected to static HTML copies, organised like a blog, of all the posts that you add to that named outline. The documentation also explain how to add WordPress-like categories or generate RSS feeds.

If you plan to use *Fargo* just for private outlines, but occasionally want to share one of them with others, in read-only mode, select File > View In Reader: this will produce a public URL of your outline that you can distribute to your friends, students or colleagues.

Desktop integration and automation

What you have learned so far is enough to make most aspiring authors of outliners and personal blogs happy, but we Linux users are more demanding than the average bear.

Writing outlines and optionally publishing them online with *Fargo* is easy and efficient, but could we do more? For example, would it be possible to reuse *Fargo* content in other publishing systems, with as little manual work as possible?

Or what about writing outlines locally (even when there is no connectivity), and uploading them automatically when you connect to the internet?

The first activity – re-use – is pretty easy. Set up the Linux client for Dropbox to automatically copy all the raw outlines onto your computer in OPML format, then play with tools like *Pandoc* to convert them to other formats, as in these two examples:

```
#> pandoc -f opml -t html outline.opml > outline.html
```

```
#> pandoc -f opml -t markdown outline.opml > outline.md
```

In other words, it's easy to avoid being locked into *Fargo* as an outline-based editor.

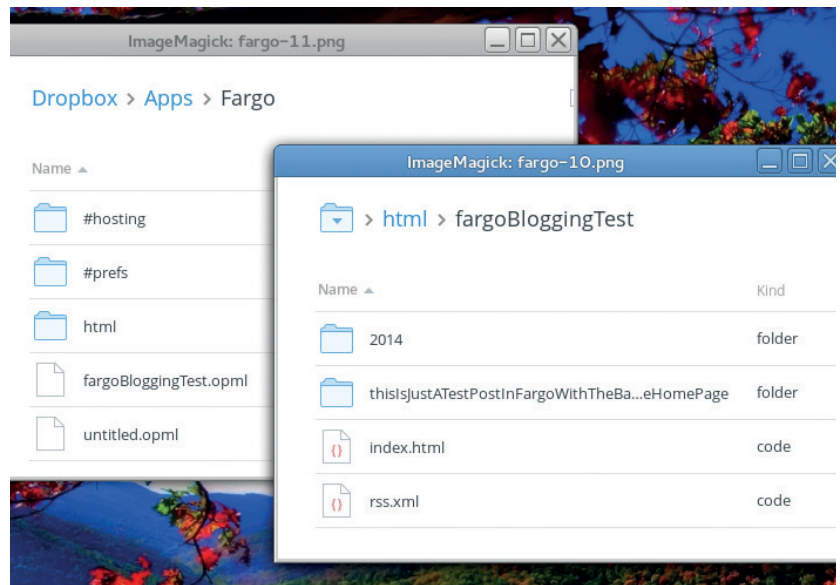
The reverse path – that is, generating OPML outlines on your computer and using them in *Fargo* – is not possible yet. Not directly, at least. If you put OPML files in your *Fargo* folder at **Dropbox.com** nothing will happen. The only available workaround so far seems to be uploading those files somewhere else, and then to tell *Fargo* to include them. This location can even be another subfolder of your Dropbox account, as long as you share it to get a publicly accessible URL usable by *Fargo*.

Control, and alternatives

All this finally leads us back to the final basic point of *Fargo*, the one that we only hinted at in the beginning, and to the future developments of this technology.

One of the official announcements of *Fargo* proudly points out that using it “you have a lot more freedom about where you host your website”. In reality, as you should have already noticed, things are quite different from that, at least now and for average users.

On one hand, you have to have a Dropbox account and let them “see” your private documents, which is not all that comfortable in this post-Snowden era. On



This is where all your raw content, obviously in open formats, ends up in *Fargo*: inside dedicated folders and subfolders of the *Fargo* app space of your Dropbox account.

the other, if you want to use *Fargo* for blogging, your online presence will only be as stable as the **smallpict.com** domain name, and the willingness of its owners to let you use it for free.

Wouldn't be great if all the servers *Fargo* needed were a Raspberry Pi under your desk, and it could use any domain name of your choice?

Truth be told, Dave Winer and the other developers of *Fargo* do see all the limitations, and are more than willing to overcome them. In fact, we already have some alternatives today, and a road ahead to solve the problem for good.

The already existing, but radical solution to the problem just mentioned is to not use *Fargo*. If you think about it, a desktop-based outline editor coupled with a static blog engine like *Mynt* or *Jekyll* already provides most of what you may get from *Fargo*. Especially on Linux, which gives you the ability to couple it with the right set of shell scripts.

At the same time, it is hard to beat the ease of use and device independence of *Fargo*. And the companion free software of *Fargo* called *Fargo Publisher* (<https://github.com/scripting/fargoPublisher>) can already transfer HTML versions of *Fargo* outlines to any server of your choice, solving the domain name problem for good. The process is quite complex, but Chris Dadswell, who is already using it, made a great job of documenting all the steps at <http://scriven.chrisdadswell.co.uk/articles/howtofargoselfpublishingstorageoptions.html> and <http://scriven.chrisdadswell.co.uk/articles/howToSelfPublishingFargoBlog.html>.

The Dropbox dependency remains, but with any luck we'll also get over it. Stay tuned for another tutorial when that day comes! 📖

Marco Fioretti is a Free Software and open data campaigner who has advocated FOSS all over the world.

LV PRO TIP

Markdown (<http://daringfireball.net/projects/markdown/>) may be the most popular, if not the most versatile, markup system for plain text available today. Learning to write and convert text with Markdown, regardless of *Fargo*, would be a very smart move if you want to publish lots of text regularly.

LV PRO TIP

You can transform your outline in online presentations as explained at <http://fargo.io/docs/presentations.html>.

WHY DO THIS?

- Get to know USB.
- Earn some geek points with reverse engineering.
- Practice with the PyUSB library.

DRIVE IT YOURSELF: A USB CAR

Ever wondered how device drivers are reverse engineered? We'll show you with a simple yet complete example.

Have you ever been enticed into a Windows versus Linux flame war? If not, you are lucky. Otherwise, you probably know that Windows fanboys often talk as though support for peripherals in Linux is non-existent. While this argument loses ground every year (the situation is incomparably better than it was in around 2005), you can still occasionally come across a device that is not recognised by your favourite distribution. Most of the time, this will be some sort of a USB peripheral.

The beauty of free software is that you can fix this situation yourself. The effort required is obviously dependent on how sophisticated the peripheral is, and with a shiny new 3D web camera you may be out of luck. However, some USB devices are quite simple, and with Linux, you don't even need to delve into the kernel and C to write a working driver program for it. In this tutorial, we'll show you how it's done step by step, using a high-level interpreted language (Python, you guessed it) and a toy USB radio controlled car I happen to have lying around.

What we are going to do is a basic variant of a process generally known as reverse engineering. You start examining the device with common tools (USB is quite descriptive itself). Then you capture the data that the device exchanges with its existing (Windows) driver, and try to guess what it means. This is the toughest part, and you'll need some experience and a

bit of luck to reverse engineer a non-trivial protocol. This is legal under most jurisdictions, but as usual, contact a lawyer if in doubt.

Get to know USB

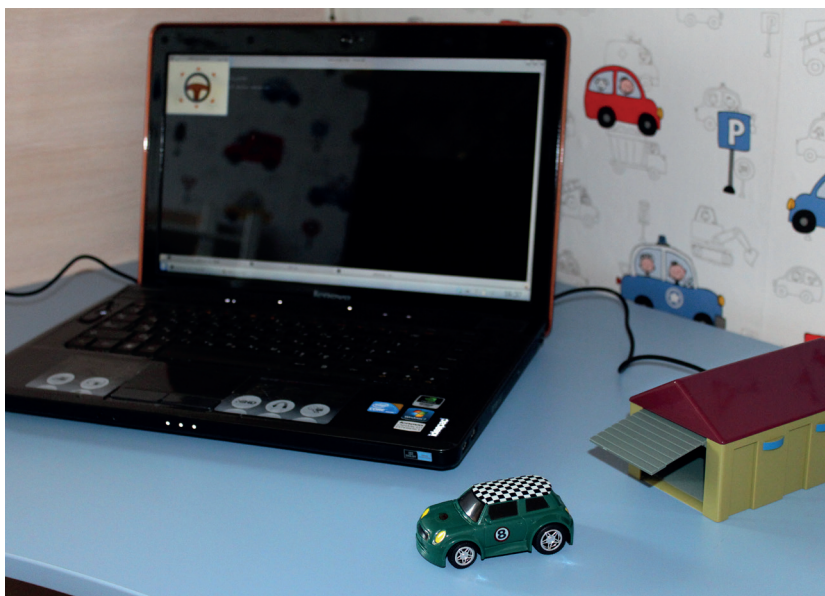
Before you start reversing, you'll need to know what exactly USB is. First, USB is a host-controlled bus. This means that the host (your PC) decides which device sends data over the wire, and when it happens. Even an asynchronous event (like a user pressing a button on a USB keyboard) is not sent to the host immediately. Given that each bus may have up to 127 USB devices connected (and even more if hubs are concerned), this design simplifies the management.

USB is also a layered set of protocols somewhat like the internet. Its lowest layer (an Ethernet counterpart) is usually implemented in silicon, and you don't have to think about it. The USB transport layer (occupied by TCP and UDP in the internet – see page 64 for Dr Brown's exploration of the UDP protocol) is represented by 'pipes'. There are stream pipes that convey arbitrary data, and message pipes for well-defined messages used to control USB devices. Each device supports at least one message pipe. At the highest layer there are the application-level (or class-level, in USB terms) protocols, like the ubiquitous USB Mass Storage (pen drives) or Human Interface Devices (HID).

Inside a wire

A USB device can be seen as a set of endpoints; or, simply put, input/output buffers. Each endpoint has an associated direction (in or out) and a transfer type. The USB specification defines several transfer types: interrupt, isochronous, bulk, and control, which differ in characteristics and purpose.

Interrupt transfers are for short periodic real-time data exchanges. Remember that a host, not the USB device, decides when to send data, so if (say) a user presses the button, the device must wait until the host asks: "Were there any buttons pressed?". You certainly don't want the host to keep silent for too long (to preserve an illusion that the device has notified the host as soon as you pressed a button), and you don't want these events to be lost. Isochronous transfers are somewhat similar but less strict; they allow for larger data blocks and are used by web cameras and similar devices, where delays or even losses of a single frame are not crucial.



Fun to play and also simple: this is the device we will write a driver for.

Fixing permissions

By default, only root is able to work with USB devices in Linux. It's not a good idea to run our example program as a superuser, so add a following udev rule to fix the permissions:

```
SUBSYSTEM=="usb", ATTRS{idVendor}=="0a81",
ATTRS{idProduct}=="0702", GROUP="INSERT_HERE",
MODE="0660"
```

Just insert the name of a group your user belongs to and put this in `/lib/udev/rules.d/99-usbcar.rules`.

Bulk transfers are for large amounts of data. Since they can easily hog the bus, they are not allocated the bandwidth, but rather given what's left after other transfers. Finally, the control transfer type is the only one that has a standardised request (and response) format, and is used to manage devices, as we'll see in a second. A set of endpoints with associated metadata is also known as an interface.

Any USB device has at least one endpoint (number zero) that is the end for the default pipe and is used for control transfers. But how does the host know how many other endpoints the device has, and which type they are? It uses various descriptors available on specific requests sent over the default pipe. They can be standard (and available for all USB devices), class-specific (available only for HID, Mass Storage or other devices), or vendor-specific (read "proprietary").

Descriptors form a hierarchy that you can view with tools like **lsusb**. On top of it is a Device descriptor, which contains information like device Vendor ID (VID) and Product ID (PID). This pair of numbers uniquely identifies the device, so a system can find and load the appropriate driver for it. USB devices are often rebranded, but a **VID:PID** pair quickly reveals their origin. A USB device may have many configurations (a typical example is a printer, scanner or both for a multifunction device), each with several interfaces. However, a single configuration with a single interface is usually defined. These are represented by Configuration and Interface descriptors. Each endpoint also has an Endpoint descriptor that contains its address (a number), direction (in or out), and a transfer type, among other things.

Finally, USB class specifications define their own descriptor types. For example, the USB HID (human interface device) specification, which is implemented by keyboards, mice and similar devices, expects all data to be exchanged in the form of 'reports' that are sent/received to and from the control or interrupt endpoint. Class-level HID descriptors define the report format (such as "1 field 8 bits long") and the intended usage ("offset in the X direction"). This way, a HID device is self-descriptive, and can be supported by a generic driver (*usbhid* on Linux). Without this, we would need a custom driver for each individual USB mouse we buy.

It's not too easy to summarise several hundred pages of specifications in a few passages of the



tutorial text, but I hope you didn't get bored. For a more complete overview of how USB operates, I highly recommend O'Reilly's *USB in a Nutshell*, available freely at www.beyondlogic.org/usbnutshell. And now, let's do some real work.

No, you can't control this car from a PC – it's a mouse and misses Output reports.

Under the hood

For starters, let's take a look at how the car looks as a USB device. **lsusb** is a common Linux tool to enumerate USB devices, and (optionally) decode and print their descriptors. It usually comes as part of the **usbutils** package.

```
[val@y550p ~]$ lsusb
```

```
Bus 002 Device 036: ID 0a81:0702 Chesen Electronics Corp.
```

```
Bus 002 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

```
...
```

The car is the Device 036 here (unplug it and run **lsusb** again to be sure). The ID field is a **VID:PID** pair. To read the descriptors, run **lsusb -v** for the device in question:

```
[val@y550p ~]$ lsusb -vd 0a81:0702
```

```
Bus 002 Device 036: ID 0a81:0702 Chesen Electronics Corp.
```

```
Device Descriptor:
```

```
...
```

```
idVendor      0x0a81 Chesen Electronics Corp.
```

```
idProduct     0x0702
```

```
...
```

```
bNumConfigurations 1
```

```
Configuration Descriptor:
```

```
...
```

```
Interface Descriptor:
```

```
...
```

```
bInterfaceClass 3 Human Interface Device
```

```
...
```

```
iInterface     0
```

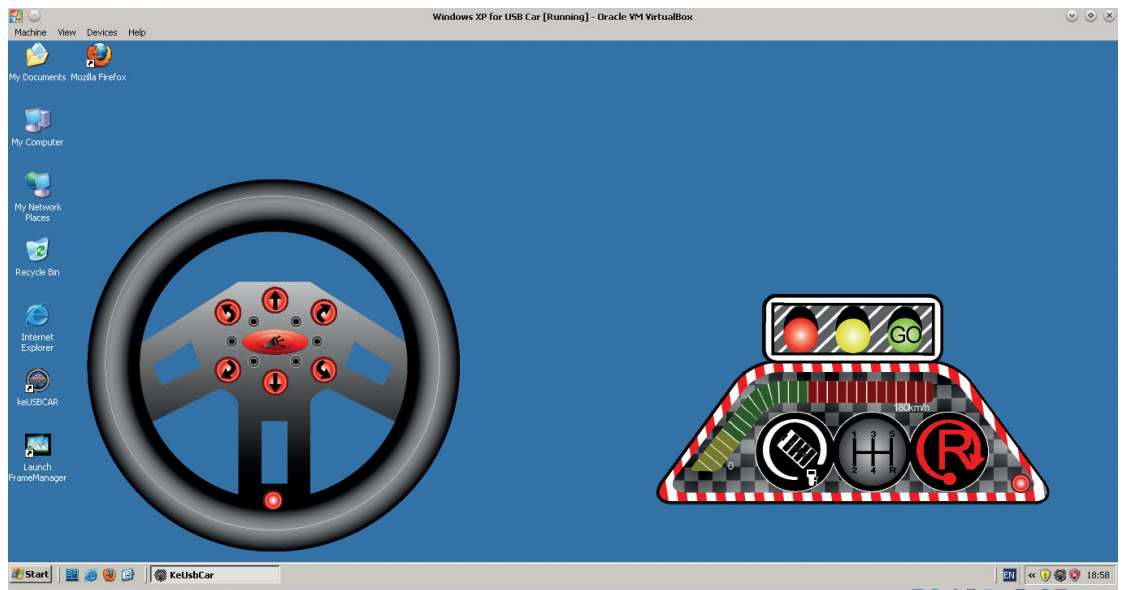
```
HID Device Descriptor:
```

```
...
```

```
Report Descriptors:
```

```
** UNAVAILABLE **
```

The original KeUsbCar application under Windows XP.



Endpoint Descriptor:
...
bEndpointAddress 0x81 EP 1 IN
bmAttributes 3
Transfer Type Interrupt
...

Here you can see a standard descriptors hierarchy; as with the majority of USB devices, the car has only one configuration and interface. You can also spot a single interrupt-in endpoint (besides the default endpoint zero that is always present and thus not shown). The **bInterfaceClass** field suggests that the car is a HID device. This is a good sign, since the HID communication protocol is open. You might think that we just need to read the Report descriptor to understand report format and usage, and we are done. However, this is marked **** UNAVAILABLE ****. What's the matter? Since the car is a HID device, the *usbhid* driver has claimed ownership over it (although it doesn't know how to handle it). We need to 'unbind' the driver to control the device ourselves.

First, we need to find a bus address for the device. Unplug the car and plug it again, run **dmesg | grep usb**, and look for the last line that starts with **usb X-Y.Z: X, Y** and **Z** are integers that uniquely identify USB ports on a host. Then run:

```
[root@y550p ~]# echo -n X-Y.Z:1.0 > /sys/bus/usb/drivers/usbhid/unbind
```

1.0 is the configuration and the interface that we want the *usbhid* driver to release. To bind the driver again, simply write the same into **/sys/bus/usb/drivers/usbhid/bind**.

Now, Report descriptor becomes readable:

Report Descriptor: (length is 52)
Item(Global): Usage Page, data= [0xa0 0xff] 65440
(null)
Item(Local): Usage, data= [0x01] 1
(null)
...
Item(Global): Report Size, data= [0x08] 8

Item(Global): Report Count, data= [0x01] 1
Item(Main): Input, data= [0x02] 2
...
Item(Global): Report Size, data= [0x08] 8
Item(Global): Report Count, data= [0x01] 1
Item(Main): Output, data= [0x02] 2
...

Here, two reports are defined; one that is read from the device (Input), and the other that can be written back to it (Output). Both are one byte long. However, their intended usage is unclear (Usage Page is in the vendor-specific region), and it is probably why the *usbhid* driver can't do anything useful with the device. For comparison, this is how a USB mouse Report descriptor looks (with some lines removed):

Report Descriptor: (length is 75)
Item(Global): Usage Page, data= [0x01] 1
Generic Desktop Controls
Item(Local): Usage, data= [0x02] 2
Mouse
Item(Local): Usage, data= [0x01] 1
Pointer
Item(Global): Usage Page, data= [0x09] 9
Buttons
Item(Local): Usage Minimum, data= [0x01] 1
Button 1 (Primary)
Item(Local): Usage Maximum, data= [0x05] 5
Button 5
Item(Global): Report Count, data= [0x05] 5
Item(Global): Report Size, data= [0x01] 1

A bonus value

Most RC toys are quite simple and use stock receivers and other circuits that operate at the same frequencies. This means our car driver program can be used to control toys other than the car that comes bundled. I've just discovered that I can play with my son's tractor from my laptop. With some background in amateur radio, you'll certainly find more interesting applications for this.

Item(Main): Input, data= [0x02] 2

This is crystal clear both for us and for the OS. With the car, it's not the case, and we need to deduce the meaning of the bits in the reports ourselves by looking at raw USB traffic.

Detective work

If you were to analyse network traffic, you'd use a sniffer. Given that USB is somewhat similar, it comes as no surprise that you can use a sniffer to monitor USB traffic as well. There are dedicated commercial USB monitors that may come in handy if you are doing reverse engineering professionally, but for our purposes, the venerable *Wireshark* will do just fine.

Here's how to set up USB capture with *Wireshark* (you can find more instructions at). First, we'll need to enable USB monitoring in the kernel. The **usbmon** module is responsible for that, so load it now:

```
root@y550p ~]# modprobe usbmon
```

Then, mount the special **debugfs** filesystem, if it's not already mounted:

```
root@y550p ~]# mount -t debugfs none /sys/kernel/debug
```

This will create a **/sys/kernel/debug/usb/usbmon** directory that you can already use to capture USB traffic with nothing more than standard shell tools:

```
root@y550p ~]# ls /sys/kernel/debug/usb/usbmon
```

```
0s 0u    1s 1t    1u 2s    2t 2u
```

There are some files here with cryptic names. An integer is the bus number (the first part of the USB bus address); **0** means all buses on the host. **s** stands for 'statistics' **t** is for 'transfers' (ie what's going over the wire) and **u** means URBs (USB Request Blocks, logical entities that represents a USB transaction). So, to capture all transfers on Bus 2, just run:

```
root@y550p ~]# cat /sys/kernel/debug/usb/usbmon/2t
```

```
ffff88007d57cb40 296194404 S li:036:01 -115 1 <
```

```
ffff88007d57cb40 296195649 C li:036:01 0 1 = 05
```

```
ffff8800446d4840 298081925 S Co:036:00 s 21 09 0200 0000 0001 1 = 01
```

```
ffff8800446d4840 298082240 C Co:036:00 0 1 >
```

```
ffff880114fd1780 298214432 S Co:036:00 s 21 09 0200 0000 0001 1 = 00
```

Unless you have a trained eye, this feedback is unreadable. Luckily, *Wireshark* will decode many protocol fields for us.

Now, we'll need a Windows instance that runs the original driver for our device. The recommended way is to install everything in *VirtualBox* (the Oracle Extension Pack is required, since we need USB support). Make sure *VirtualBox* can use the device, and run the Windows program (*KeUsbCar*) that controls the car. Now, start *Wireshark* to see what commands the driver sends over the wire. At the initial screen, select the 'usbmonX' interface, where X is the bus that the car is attached to. If you plan to run *Wireshark* as a non-root user (which is recommended), make sure that the **/dev/usbmon*** device nodes have the appropriate permissions.

Suppose we pressed a "Forward" button in *KeUsbCar*. *Wireshark* will catch several output control

The screenshot shows the Wireshark interface with a list of captured USB packets. The selected packet (No. 4186) is a USB_URB_CONTROL packet. The details pane shows the following fields:

- USB_URB
- USB_URB setup
 - bmRequestType: 0x21
 - bRequest: 9
 - wValue: 0x0200
 - wIndex: 0
 - wLength: 1
 - Leftover Capture Data: 01

The packet bytes pane shows the raw data: 0010 64 0c 93 53 00 00 00 00 65 f8 04 00 8d ff ff ff d..S....e.....

Wireshark captures Windows driver-originated commands.

transfers, as shown on the screenshot above. The one we are interested in is highlighted. The parameters indicate it is a **SET_REPORT HID class-specific request (bmRequestType = 0x21, bRequest = 0x09)** conventionally used to set a device status such as keyboard LEDs. According to the Report Descriptor we saw earlier, the data length is 1 byte, and the data (which is the report itself) is 0x01 (also highlighted).

Pressing another button (say, "Right") results in similar request; however, the report will be 0x02 this time. One can easily deduce that the report value encodes a movement direction. Pressing the remaining buttons in turn, we discover that 0x04 is reverse right, 0x08 is reverse, and so on. The rule is simple: the direction code is a binary 1 shifted left by the button position in *KeUsbCar* interface (if you count them clockwise).

We can also spot periodic interrupt input requests for Endpoint 1 (0x81, 0x80 means it's an input endpoint; 0x01 is its address). What are they for? Except buttons, *KeUsbCar* has a battery level indicator, so these requests are probably charge level reads. However, their values remain the same (0x05) unless the car is out of the garage. In this case, there are no interrupt requests, but they resume if we put the car back. We can suppose that 0x05 means "charging" (the toy is simple, and no charge level is really returned, only a flag). If we give the car enough time, the battery will fully charge, and interrupt reads will start to return 0x85 (0x05 with bit 7 set). It looks like the bit 7 is a "charged" flag; however, the exact meaning of other two flags (bit 0 and bit 2 that form 0x05) remains unclear. Nevertheless, what we have

This may not look as good as *KeUsbCar*, but it runs under Linux.



figured out so far is already enough to recreate a functional driver.

Get to code

The program we are going to create is quite similar to its Windows counterpart, as you can easily see from the screenshot above. It has six arrow buttons and a charge level indicator that bounces when the car is in the garage (charging). You can download the code from GitHub (<https://github.com/vsinityn/usbcar.py>); the steering wheel image comes from www.openclipart.org.

The main question is, how do we work with USB in Linux? It is possible to do it from userspace (subject to permission checks, of course; see the boxout below), and the *libusb* library facilitates this process. This library is written for use with the C language and requires the user to have a solid knowledge of USB. A simpler alternative would be *PyUSB*, which is a simpler alternative: it strives to “guess” sensible defaults to hide the details from you, and it is pure Python, not C. Internally, *PyUSB* can use *libusb* or some other backend, but you generally don’t need to think about it. You could argue that *libusb* is more capable and flexible, but *PyUSB* is a good fit for cases like ours, when you need a working prototype with minimum

effort. We also use *PyGame* for the user interface, but won’t discuss this code here – though we’ll briefly visit it at the end of this section.

Download the *PyUSB* sources from <https://github.com/walac/pyusb>, unpack them and install with `python setup.py install` (possibly in a virtualenv). You will also need the *libusb* library, which should be available in your package manager. Now, let’s wrap the functionality we need to control a car in a class imaginatively named **USBCar**.

```
import usb.core
import usb.util

class USBCar(object):
    VID = 0x0a81
    PID = 0x0702

    FORWARD = 1
    RIGHT = 2
    REVERSE_RIGHT = 4
    REVERSE = 8
    REVERSE_LEFT = 16
    LEFT = 32
    STOP = 0
```

We import two main *PyUSB* modules and define the direction values we’ve deduced from the USB traffic. VID and PID are the car ID taken from the output of **lsusb**.

```
def __init__(self):
    self._had_driver = False
    self._dev = usb.core.find(idVendor=USBCar.VID,
                              idProduct=USBCar.PID)
    if self._dev is None:
        raise ValueError("Device not found")
```

In the constructor, we use the `usb.core.find()` function to look up the device by ID. If it is not found, we raise an error. The `usb.core.find()` function is very powerful and can locate or enumerate USB devices by other properties as well; consult <https://github.com/walac/pyusb/blob/master/docs/tutorial.rst> for the full details.

```
if self._dev.is_kernel_driver_active(0):
    self._dev.detach_kernel_driver(0)
    self._had_driver = True
```

Next, we detach (unbind) the kernel driver, as we did previously for **lsusb**. Zero is the interface number. We should re-attach the driver on program exit (see the `release()` method below) if it was active, so we remember the initial state in `self._had_driver`.

```
self._dev.set_configuration()
```

Finally, we activate the configuration. This call is one of a few nifty shortcuts *PyUSB* has for us. The code above is equivalent to the following, however it doesn’t require you to know the interface number and the configuration value:

```
self._dev.set_configuration(1)
usb.util.claim_interface(0)
```

```
def release(self):
    usb.util.release_interface(self._dev, 0)
```

No more toys: writing a real driver (almost)

Having a custom program to work with a previously unsupported device is certainly a step forward, but sometimes you also need it to integrate with the rest of the system. Generally it implies writing a driver, which requires coding at kernel level (see our tutorial from LV002 at www.linuxvoice.com/be-a-kernel-hacker/) and is probably not what you want. However, with USB the chances are that you can stay in userspace.

If you have a USB network card, you can use TUN/TAP to hook your *PyUSB* program into Linux networking stack. TUN/TAP interfaces look like regular network interfaces (with names like `tun0` or `tap1`) in Linux, but they make all packets received or

transmitted available through the `/dev/net/tun` device node. The `pytun` module makes working with TUN/TAP devices in Python a breeze. Performance may suffer in this case, but you can rewrite your program in C with *libusb* and see if this helps.

Other good candidates are USB displays. Linux comes with the `vfb` module, which makes a framebuffer accessible as `/dev/fbX` device. Then you can use `ioctl`s to redirect Linux console to that framebuffer, and continuously pump the contents of `/dev/fbX` into a USB device using the protocol you reversed. This won’t be very speedy either, but unless you are going to play 3D shooters over USB, it could be a viable solution.

```
if self._had_driver:
```

```
self._dev.attach_kernel_driver(0)
```

This method should be called before the program exits. Here, we release the interface we claimed and attach the kernel driver back.

Moving the car is also simple:

```
def move(self, direction):
```

```
ret = self._dev.ctrl_transfer(0x21, 0x09, 0x0200, 0, [direction])
```

```
return ret == 1
```

The direction is supposed to be one of the values defined at the beginning of the class. The `ctrl_transfer()` method does control transfer, and you can easily recognise `bmRequestType` (0x21, a class-specific out request targeted at the endpoint), `bRequest` (0x09, `Set_Report`) as defined in the USB HID specification), report type (0x0200, Output) and the interface (0) we saw in *Wireshark*. The data to be sent is passed to `ctrl_transfer()` as a string or a list; the method returns the number of bytes written. Since we expect it to write one byte, we return True in this case and False otherwise.

The method that determines battery status spans a few more lines:

```
def battery_status(self):
```

```
try:
```

```
ret = self._dev.read(0x81, 1, timeout=self.READ_TIMEOUT)
```

```
if ret:
```

```
res = ret.tolist()
```

```
if res[0] == 0x05:
```

```
return 'charging'
```

```
elif res[0] == 0x85:
```

```
return 'charged'
```

```
return 'unknown'
```

```
except usb.core.USBError:
```

```
return 'out of the garage'
```

At its core is the `read()` method, which accepts an endpoint address and the number of bytes to read. A transfer type is determined by the endpoint and is stored in its descriptor. We also use a non-default (smaller) timeout value to make the application more responsive (you won't do it in a real program: a non-blocking call or a separate thread should be used instead). `Device.read()` returns an array (see the 'array' module) which we convert to list with the `tolist()` method. Then we check its first (and the only) byte to determine charge status. Remember that this it is not reported when the car is out of the garage. In this case, the `read()` call will run out of time and throw a `usb.core.USBError` exception, as most *PyUSB* methods do. We (fondly) assume that the timeout is



With *PyUSB* we could also control this toy digger, so you may find that the drivers you write will have more uses than you imagined.

the only possible reason for the exception here. In all other cases we report the status as 'unknown'.

Another class, creatively named `UI`, encapsulates the user interface – let's do a short overview of the most important bits. The main loop is encapsulated in the `UI.main_loop()` method. Here, we set up a background (steering wheel image taken from OpenClipart.org), display the battery level indicator if the car is in the garage, and draw arrow buttons (`UI.generate_arrows()` is responsible for calculating their vertices' coordinates). Then we wait for the event, and if it is a mouse click, move the car in the specified direction with the `USBCar.move()` method described earlier.

One tricky part is how we associate directions with arrow buttons. There is more than one way to do it, but in this program we draw two sets of arrows with identical shapes. A first one, with red buttons you see on the screenshot, is shown to the user, while the second one is kept off-screen. Each arrow in that hidden set has a different colour, whose R component is set to a direction value. Outside the arrows, the background is filled with 0 (the `USBCar.STOP` command). When a user clicks somewhere in the window, we just check the R component of the pixel underneath the cursor in that hidden canvas, and action appropriately.

The complete program with a GUI takes little more than 200 lines. Not bad for the device we didn't even had the documentation for!

That's all folks!

This concludes our short journey into the world of reverse engineering and USB protocols. The device for which we've developed a driver (or more accurately, a support program) was intentionally simple. However, there are many devices similar to this USB car out there, and many of them use a protocol that is close to the one we've reversed (USB missile launchers are good example). Reversing a sophisticated device isn't easy, but now you can already add Linux support for something like a desktop mail notifier. While it may not seem immediately useful, it's a lot of fun. 📺

Dr Valentine Sinitsyn edited the Russian edition of O'Reilly's *Understanding the Linux Kernel*, has a PhD in physics, and is currently doing clever things with Python.

Resources

- *USB in a Nutshell*: www.beyondlogic.org/usbnutshell
- USB Capture Setup at the *Wireshark* wiki: <http://wiki.wireshark.org/CaptureSetup/USB>
- Tutorial code: <https://github.com/vsinityn/usbcar.py>
- PyUSB homepage: <https://github.com/walac/pyusb>
- "Programming with PyUSB 1.0" tutorial: <https://github.com/walac/pyusb/blob/master/docs/tutorial.rst>

BASH: BEYOND THE COMMAND PROMPT

Speed up repetitive tasks, get more power out of the command line or just make life easier – welcome to the world of Bash scripting.

WHY DO THIS?

- Chain commands together to create flexible scripts.
- Get more from the command line.
- Learn a new way of working.

Most Linux users will know *Bash* as the command line prompt. But it is also a powerful programming language – a lot of the code that glues the various parts of your system together is written in *Bash*. You may have looked at some of it and seen seas of parentheses, braces and brackets. This less-than-obvious syntax helps make other languages, such as Python, more attractive to beginners. But *Bash* is ubiquitous in the Linux world, and it's worth taking the time to learn how to go beyond the prompt.

A good introduction to *Bash* programming is to put frequently issued command sequences into a file so that you can replay them at the command prompt instead of typing each one. Such a file is called a script, and we often hear “scripting” instead of “programming”. *Bash* is nonetheless a language with its own syntax, capabilities and limitations.

The basics

Bash programs, like Python and Ruby, are not compiled into binary executables, but need to be parsed by an interpreter. For *Bash*, this is an executable called **bash** that interprets commands read interactively from its command prompt or from a script. If you're at a *Bash* prompt, it'll be provided by a running **bash** process, and you can feed a script straight to it:

```
$ source myscript
```

But you may not be at such a prompt (you might use another shell, such as *csh* or *ksh*, or you may be at the Run dialog of your desktop). If you set the execute bit on your script:

```
$ chmod +x myscript
```

then you can execute it:

```
$ myscript
```

which causes your shell to ask the operating system's

program loader to start it. This creates, or forks, a child process of your shell.

But the script isn't a binary executable, so the program loader needs to be told how to execute it. You do this by including a special directive as the first line of your script, which is why most **bash** scripts begin with a line this:

```
#!/bin/bash
```

The first two characters, **#!**, known as a shebang, are detected by the program loader as a magic number that tells it that the file is a script and that it should interpret the remainder of the line as the executable to load – plus, optionally, any arguments to pass to it along with the script itself. The program loader starts `\bin\bash` in a new process, and this runs the script. It needs the absolute path to the executable because the kernel has no concept of a search path (that is itself a feature of the shell).

Scripts that perform specific tasks are usually executed so they run in a predictable environment. Every process has an environment that it inherits from its parent, and contains so-called environment variables that offer its parent a way to pass information into it. A process can alter its own environment and prepare that of its children, but it cannot affect its parent.

Scripts specifically written to alter the current environment (like **rc** files) are sourced and usually don't have their execute bit set.

One line at a time

Bash reads input one line at a time, whether from a command prompt or a script. Comments are discarded; they start with a hash **#** character and continue to the end of the line (**bash** sees the shebang as a comment). It applies quoting rules and parameter expansion to what remains and ends up with words – commands, operators and keywords that make up the language. Commands are executed and return an exit status, which is stored in a special variable for use by subsequent commands.

Words are separated by metacharacters: a space or one of **|**, **&**, **;**, **(**, **)**, **<** or **>**. Operators are character sequences containing one or more metacharacters.

Metacharacters can have their special meaning removed by quoting. The first form of quoting removes special meaning from all characters enclosed by single quotes. It is not possible to enclose a single quote within single quotes. Double quotes are

POSIX

An IEEE standard for a portable operating system interface, POSIX is frequently mentioned in texts about shell scripting. It means being compatible with something called the Shell Command Language, which is defined by an IEEE standard and implemented as the shell on all Unix-like systems by the `/bin/sh` command. These days `/bin/sh` is usually a symlink to a shell that

can run in a POSIX-compliant mode. The **bash** command does this when launched in this way or if given the `--posix` command-line option.

In POSIX mode, Bash only supports the features defined by the POSIX standard. Anything else is commonly called a bashism. See <http://bit.ly/bashposix> for what's different in *Bash*'s POSIX mode.

Chain of command

Bash expects one command per line, but this can be a chain: a sequence of commands connected together with one of four special operators. Commands chained with **&&** only execute if the exit status of the preceding one was 0, indicating success. Conversely, commands chained with **||** execute only if the preceding one failed. Commands chained with a semicolon (;) execute irrespective of how the prior command exited. Lastly, the single-ampersand **&** operator chains commands, placing the preceding command into the background:

```
command1 && command2 # perform command2 only if
command1 succeeded
```

```
command1 || command2 # perform command2 only if
command1 failed
```

```
command1 ; command2 # perform command1 and then
command2
```

```
command1 & command2 # perform command2 after starting
command1 in the background
```

Chains can be combined, giving a succinct if-then-else construct:

```
command1 && command2 || command3
```

The exit status of a chain is the exit status of the last command to execute.

similar, except some metacharacters still work, most notably the Dollar sign, which performs parameter expansion, and the escape ****, which is the third form of quoting and removes special meaning from the following character only.

Parameters pass information into the script. Positional parameters contain the script's argument list, and special variables provide ways to access them en-masse and also provide other information like the script's filesystem path, its process ID and the last command's exit status.

Variables are parameters that you can define by assigning a value to a name. Names can be any string of alphanumeric characters, plus the underscore (**_**) but cannot begin with a numeric character, and all values are character strings, even numbers. Variables don't need to be declared before use, but doing so enables additional attributes to be set such as making them read-only (effectively a constant) or defining them as an integer or array (they're still string values though!). Assignment is performed with the **=** operator and must be written without spaces between the name and value. Here are some examples that you might see:

```
var1=hello
```

```
var2=1234
```

```
declare -i int=100 # integer
```

```
declare -r CON=123 # constant
```

```
declare -a arr=(foo bar baz) # array
```

Variables default to being shell variables; they aren't part of the environment passed to child processes. For that to happen, the variable must be exported as an environment variable:

```
export $MYVAR
```

Names can use upper- and lower-case characters and are case-sensitive. It's good practice to use lower

case names for your own variables and use upper case names for constants and environment variables.

Parameter expansion happens when a parameter's name is preceded by the dollar sign, and it replaces the parameter with its value:

```
echo $1
```

which outputs the script's first argument. These so-called positional parameters are numbered upwards from 1 and 0 contains the filesystem path to the script. Parameter names can be enclosed by { and } if their names would otherwise be unclear. Consider this:

```
$ var=1234
```

```
$ echo $var5678
```

```
$ echo ${var}5678
```

```
12345678
```

The first **echo** receives the value of a non-existent variable **var5678** whereas the second gets the value of **var**, followed by **5678**. The other thing to understand about parameters is that **bash** expands them before any command receives them as arguments. If this expansion includes argument separators, then the expanded value will become multiple arguments. You'll encounter this when values contain spaces, and the solution to this problem is quoting:

```
$ file='big data'
```

```
$ touch "$file"
```

```
$ ls $file
```

```
ls: cannot access big: No such file or directory
```

```
ls: cannot access data: No such file or directory
```

Here, **touch** creates a file called **big data** because the **file** variable is quoted, but **ls** fails to list it because it is unquoted and therefore receives two arguments instead of one.

For these two reasons, it is common to quote and delimit parameters when expanding them; many scripts refer to variables like this:

```
"${myvar}"
```

Braces are also required to expand array variables. These are defined using parentheses and expanded with braces:

```
$ myarr=(foo bar baz)
```

```
$ echo "${myarr[@]}" # values
```

```
foo bar baz
```

```
$ echo "${!myarr[@]}" # indices
```

Special Variables

- 0** The name of the shell (if interactive) or script.
- 1 .. n** The positional parameters numbered from 1 to the number of arguments **n**. Braces must be used when expanding arguments greater than 9 (eg **\${10}**).
- *** All the positional parameters. Expanding within quotes gives a single word containing all parameters separated by spaces (eg **"\$*"** is equivalent to **"\$1 \$2 ... \$n"**).
- @** All the positional parameters. Expanding within quotes gives all parameters, each as a separate word (eg **"\$@"** is equivalent to **"\$1 \$2 ... \$n"**).
- ?** The exit status of the most recent command.
- \$** The process ID of the shell.
- !** The PID of the last backgrounded command.

LV PRO TIP

You can use a **."** instead of **"source"** to run a script in the current environment.

```
0 1 2
$ echo "${#myarr[@]}" # count
3
```

Arrays are indexed by default and do not need to be declared. You can also create associative arrays if you have **bash** version 4, but you need to declare them:

```
$ declare -A hash=( [key1]=value1 [key2]=value2 )
$ hash[key3]=value3
$ echo ${hash[@]}
value3 value2 value1
$ echo ${!hash[@]}
key3 key2 key1
$ echo ${hash[key1]}
value1
```

Braces are also used for inline expansion, where `///:a,bV///1` becomes `a1 b1` and `///:1..5V///` becomes `1 2 3 4 5`. Braces also define a command group: a sequence of commands that are treated as one so that their input and output can be redirected:

```
(date; ls;) > output.log
```

A similar construct is the subshell. Commands written in parentheses are launched in a child process. Expanding them enables us to capture their output:

```
now=$(date +%T)
```

Although our example used a child process, the parent blocked; it waited for the child to finish before continuing. Child processes can also be used to run tasks in parallel by backgrounding them:

```
(command)&
```

This enables your script to continue while the 'command' runs in a separate process. You can **wait**, perhaps later on in your script, for it to finish.

Unlike the subshell, the command group does not fork a child process and, therefore, affects the current environment. They cannot be used in a pipeline and they cannot be expanded to capture their output. Subshells can do these things and are also useful for running parallel processes in separate environments.

Do the maths

You'll also encounter double parentheses; these are one way to do integer arithmetic (**bash** doesn't have floating-point numbers); **let** and **expr** are others:

```
profit=$((income - expenses))
profit=$((income - expenses))
let profit=income-expenses
profit=$(expr $income - $expenses)
```

The double parentheses form allows spaces to be inserted and the dollar signs to be omitted from the expression to aid readability. Also note that the use of **expr** is less efficient, because it's an external command. Arithmetic expansion also allows operators similar to those found in the C programming language, as in this common idiom to increment a variable:

```
$ x=4
$ let x++
$ echo $x
5
```

Finally, we have square brackets, which evaluate expressions and expand to their exit status. They're

used to test and compare parameters, variables and file types. There are single- and double-bracket variants; the single bracket expression is an alias for the **test** command – these are equivalent:

```
"$myvar" == hello
test "$myvar" == hello
```

The double bracket expression is a more versatile extended test command (see **help [[**), which is a keyword and part of the language syntax. **test** is just a command that has the opening bracket as an alias and, when used that way, expects its last argument to be a closing bracket. This is an important difference to understand, because it affects how the expression is expanded. **test** is expanded like arguments to any other command, whereas an extended **test** expression is not expanded but parsed in its entirety as an expression with its own syntax, in a way that's more in line with other programming languages.

It supports the same constructs as **test** (see **help test** or **man test**), performs command substitution and expands parameters. Values don't need to be quoted, and comparison operators (**=**, **&&**, **||**, **>** and **<**) work as expected, plus the **=~** operator compares with a regular expression:

```
$ [[ hello =~ ^he ]] && echo match
match
```

Like any command, both single- and double-bracket expressions expand to their exit status and can be used in conditionals that use it to choose the path of execution:

```
if c; then c; fi
if c; then c; else c; fi
if c; then c; elif c; then c; else c; fi
```

where **c** is a command. The semicolons can be omitted if the following word appears on a new line. Each command can be multiple commands but it is the exit status of the final conditional command that determines the execution path. Conditionals can be nested too:

Internal and external commands

Some commands are implemented within *Bash* and are known as builtins. They are more efficient than other external commands because they don't have the overhead of forking a child process. Some builtins have equivalent external commands that pre-date them being implemented within **bash**. Keywords are similar to builtins but are reserved words that form part of the language syntax. You can use **type** to see what a word means in **bash**:

```
$ type cat
cat is /usr/bin/cat
$ type echo
echo is a shell builtin
$ type /usr/bin/echo
/usr/bin/echo is /usr/bin/echo
$ type if
if is a shell keyword
```

You can get help on builtin commands and keywords:

```
$ help {
{ ... }: { COMMANDS ; }
Group commands as a unit.
```

LV PRO TIP

The *Advanced Bash Scripting Guide* contains an unofficial style guide <http://bit.ly/bashstyle>.

A question of truth

A Boolean expression is either true or false. In *Bash*, true and false are shell builtins (you may also find equivalent external commands in `/usr/bin`) and, like all commands, they return an exit status where zero indicates success and a non-zero value indicates failure. So, 'true' returns 0 and 'false' returns 1.

You may be tempted to write something like this:

```
var=true
```

This assigns a variable called `var` with the value of the four-character string `true`, and has nothing to do with the `true` command. Similarly,

```
if [[ $var == true ]]; then...
```

compares the value of `var` with the four-character string `true`, whereas

`if true; then...`

always succeeds. Here `true` is the command and its exit status is 0, indicating success.

To confuse things further, arithmetic expansion sees 1 as true and 0 as false, and sees the words "true" and "false" as (potentially undefined) variables rather than the builtins described above.

```
$ echo $((true == false))
```

```
1
```

That happens because both `true` and `false` are undefined variables that expand to the same value (nothing) and are therefore equal. This makes the expression true which, arithmetically, is 1.

```
if condition
```

```
then
```

```
if nested-condition
```

```
command
```

```
else
```

```
command
```

```
fi
```

```
fi
```

`while` and `until` loops are also controlled by exit status:

```
while c; do c; done
```

```
until c; do c; done
```

The `for` loop is different – it iterates over a series of words:

```
for i in foo bar baz
```

```
do
```

```
something
```

```
done
```

but you can use brace expansion to simulate a counting loop:

```
for i in {1..10}
```

Function definition

No programming language would be complete without some way to group and reuse code, and `bash` has functions. A function is easy to define, either:

```
function myfunc {
```

```
}
```

or (preferably, and POSIX compliant):

```
myfunc () {
```

```
}
```

Functions have the same naming rules as variables but it's conventional to use lower-case words separated by underscores. They can be used wherever commands can, and are given arguments in the same way, although the function definition doesn't define any (the parentheses must be empty). The function sees its arguments as positional parameters.

Variables defined outside a function are visible inside, and variables defined inside are accessible outside, unless declared as local:

```
function f() {
```

```
in1=456
```

```
local in2=789
```

```
echo $out$in1$in2
```

```
}
```

```
out=123
```

```
f # 123456789
```

```
echo $out$in1$in2 # 123456
```

You can be caught out by local variables. Here's an example: if a function `f1` defines a local, then calls another function `f2`, that local is also visible inside `f2`. When a function defines local variables, they are visible to any functions that it calls. Also, you can define one function inside another but you might not get what you expect. All functions are names and have similar scope. Function definitions are executed – that means that a function defined inside another function will be redefined every time that function is called.

Functions return an exit status, which is either the exit status of the last command to be executed within the function, or explicitly set with "return". Exit status is a number between 0 (meaning success) and 255. You can't return anything more complex than that.

There are, however, tricks that you can use to return more complex data from a function. Using global variables is a simple solution, but another common one is to pass the name of a variable as a parameter and use `eval` to populate it:

```
myfunc() {
```

```
local resultvar=$1
```

```
local result='a value'
```


```
eval $resultvar="$result"
```

```
}
```

```
myfunc foo
```

```
echo $foo # a value
```

`eval` enables you to build a command in a string and then execute it; so, in the example above, the function passes in `foo` and this gets assigned to the local `resultvar`. So, when `eval` is called, its argument is a string containing `foo='a value'` that it executes to set the variable `foo`. The single quotes ensure that the value of `result` is treated as one word.

These are the main parts of the language, and should be sufficient for any *Bash* script to make sense, but there are many nuances and techniques that you can still learn. Your journey beyond the prompt has just begun... 

John Lane is a technology consultant. He doesn't know where our jetpacks are, but he does help businesses use Linux.

LV PRO TIP

`test` is both a built-in and external command (`/usr/bin/test`).

LV PRO TIP

Try to always use double bracket expressions unless POSIX compliance is important.

BEN EVERARD

CODE NINJA: PROGRAMMER'S GOLF

Sometimes you just have to prove, without a doubt, that you're the best programmer in the room.

WHY DO THIS?

- Show off your 1337 programming skillz.
- Push yourself to learn more about your language of choice.
- Save minuscule amounts of disk space.

The source code to the main engine of this simplified game of Tetris is: `function(a,b,c,d,e){return d+=c,e=a|b<<d,d<0|a&b<<d&&(a=e=parseInt((a|b<<c).toString(d=32).replace(/v/, ""),d),b=new Date%2?1:3),[a,b,d,e]}`

People are naturally competitive. There's just something in human nature that makes us want to find out who's the best at something, whether it's who's the fastest runner, who can jump the furthest or who's the best at kicking balls between goalposts. Sometimes we geeks like to think we've transcended this base desire. Perhaps you have, but many of us have just transferred this competitive instinct from physical exploits to mental ones.

Linguists have crosswords, mathematicians have number puzzles, and techies have programmer's golf. Programmer's golf in case you're wondering, is the challenge of taking a particular problem and coding it in as small a number of characters as possible.

There aren't any fixed rules for this other than the result must be accepted by the interpreter or compiler as a valid program, and sometimes there are restrictions on the modules or libraries that can be used. Beyond that, anything is permissible.

A good understanding of the language being used is essential, especially as it's often the language's more esoteric features that can result in saved space.

Let's take a look at a simple example, printing the numbers 1 to 6 at one per line in Python. Done normally, this might look something like this:

```
for number in range(1,7):
```

```
    print number
```

This is 40 characters. It's easy to see we've wasted quite a bit on the variable name, so we can shrink this down to 30 characters by simply replacing it with a single letter:

```
for i in range(1,7):
```

```
    print i
```

If you're familiar with Python, you might know that there are two extra characters that we can get rid of quite easily.

```
for i in range(1,7):print i
```

It's clear that we need the **print** statement, because there's no shorter way of outputting text onto the screen. Of the remaining code, the **range** call takes up 8 characters, so it seems like a good place to look for further shrinking. We need something that Python can iterate over that returns the 1 to 6. It's important to realise that in this case, it doesn't matter if it's the numbers 1 to 6 or the characters 1 to 6, because Python's **print** statement can work with either.

How short is a piece of string?

Once you've realised that it can be the characters 1 to 6, it's fairly obvious that we can iterate the **for** loop over a string instead:

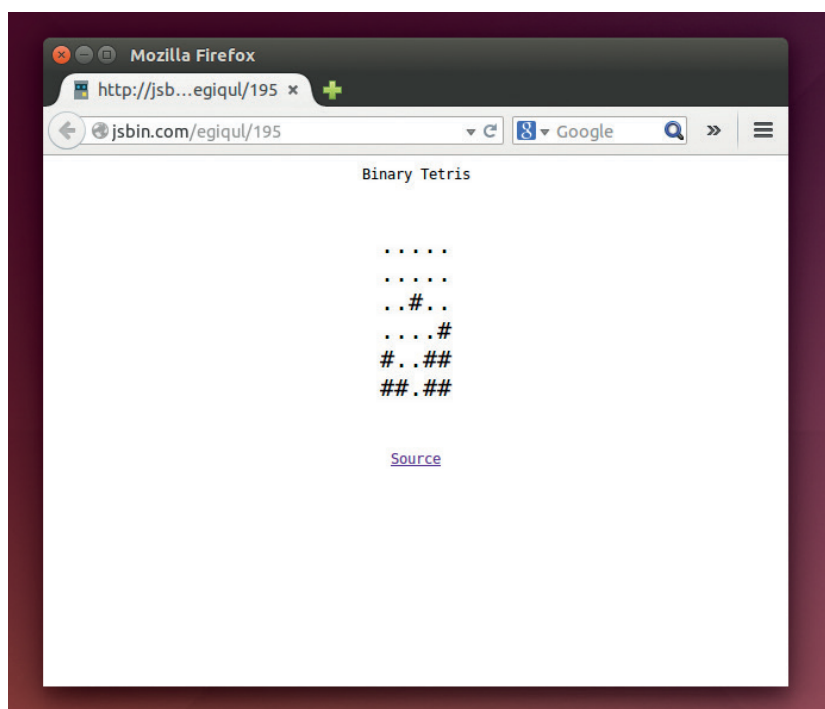
```
for i in '123456':print i
```

This has managed to claw back another couple of characters. What's more, it now uses the string type, which has quite a few powerful methods that perhaps we can make use of.

We're confident that **print** is the shortest way of outputting something to the screen, and we think that the string is the shortest way of encoding the numbers we need. The only place left to look is the **for** loop. Here, we need to think back to what the original challenge was: print the characters 1 to 6 with each character on a separate line. So far, we've been using a separate **print** statement for each line, and this has required us to use a loop to call the **print** statements on each number in turn. However, we could get rid of the loop if we printed them all with the same **print** statement, but put a new line character in-between each number.

```
print'\n'.join('123456')
```

This uses Python's **join** method on the string `'\n'`. This iterates through the argument and outputs every item in the argument with the original string between it. Since strings are iterated through on individual characters, this outputs:




```
1\n2\n3\n4\n5\n6
```

Since `\n` is the new line character, printing this results in each number being printed on a separate line.

There is another way of getting the code this short. In Python, you don't need to separate bits of text with spaces if the interpreter can distinguish between them, so you can remove the space between `in` and the start of the string. In other words, with:

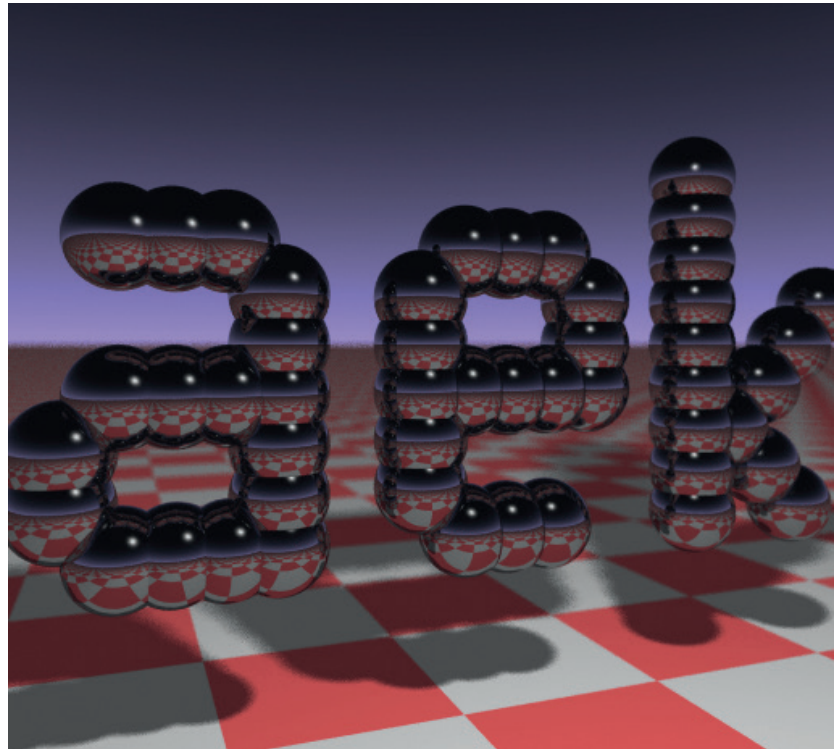
```
for i in'123456':print i
```

Is this as short as it can go? Possibly not. There's no way to know for sure that there definitely isn't a shorter way of doing something. If you find a way to remove a few characters, drop me an email at ben@linuxvoice.com. I'd love to hear it.

Some people may wonder why bother with this at all. After all, the resulting code is almost always an unreadable, unmaintainable mess. Wouldn't it be better to focus our competitive instincts on more useful aspects of programming like readability or performance? No decent programmer would focus their efforts on squeezing every last byte out of their code without a very good reason.

However, aside from the competitive aspect of the challenge, there are some skills to be learned in shrinking file sizes. For one, it forces you to learn more about your chosen language. For example, it's perfectly possible to program in Python for years, yet never really get to grips with lambda functions. However, if you're looking to squeeze a few characters out, they can be a fertile source of reductions. The features you learn may well help you program better in ways other than file size.

Many times, the tricks that you use to remove unnecessary bloat are quirks and edge cases of the



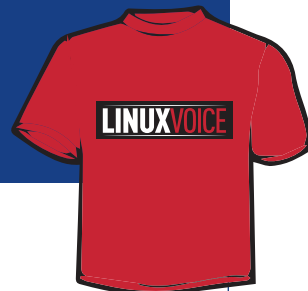
The code used to create this 3D render is small enough to fit on a business card, and perhaps more impressively, is 1337 bytes long. For more details see www.fabiensanglard.net/rayTracing_back_of_business_card

language, and these can sometimes lead to bugs or other unexpected behaviour. Learning to exploit these means learning to understand them, and this means a better understanding of the language.

Now, let's see how good you are with a little competition. Fore! 🍀

COMPETITION

Write ridiculously small code, win an attractive garment!



This month, the Linux Voice challenge is a game of Python programmer's golf. Your challenge is to write the smallest possible Python program that takes a number as input, and prints the value in Roman numerals. To get you started, here's a sample program that does just this:

```
symbols = [('M', 1000), ('C M', 900), ('D', 500),
           ('C D', 400), ('C', 100), ('X C', 90), ('L', 50),
           ('X L', 40), ('X', 10), ('I X', 9), ('V', 5),
           ('I V', 4), ('I', 1)]
```

```
def romannumeral(number):
    while number > 0:
        for symbol, value in symbols:
            if number - value >= 0:
                print symbol,
                number = number - value
                continue
```

```
number_in = raw_input("Enter a number: ")
romannumeral(int(number_in))
```

This is obviously not optimised for size, so you shouldn't have too much trouble stripping some fat off it. The question is, how much?

There are a couple of things to point out about this code. It puts a space in between each character. This is for simplicity, and any spacing between characters other than new lines is acceptable as long as it's consistent.

There is also some contention about what the Roman numerals for certain numbers are. For example, should 1999 be MIM or MCMXCIX? Without wanting to get into a historical argument about how people would have written numbers thousands of years ago, we'll simply say that your program should match the form of Roman numerals given by our program.

Beyond this, there are just a few rules:

- The length of the code will be the total length of the submitted code in characters, and the person who submits the shortest code will win an exclusive Linux Voice winner's T-shirt.
- No modules can be imported. That would just make it too easy.
- Either Python 2 or 3 is acceptable.
- Email your entries to ben@linuxvoice.com by the end of the day on 15th October 2014.
- In the event that more than one person has an entry the same length, they will both be considered winners, but the first entry received will win the T-shirt.
- All code must be released under an OSI approved open source licence, and GPL v3 is preferred.

JULIET KEMP

KONRAD ZUSE: (NEARLY) THE GERMAN TURING

Try a programming language designed amid the rubble of post-war Germany before there were any computers on which to run it.

WHY DO THIS?

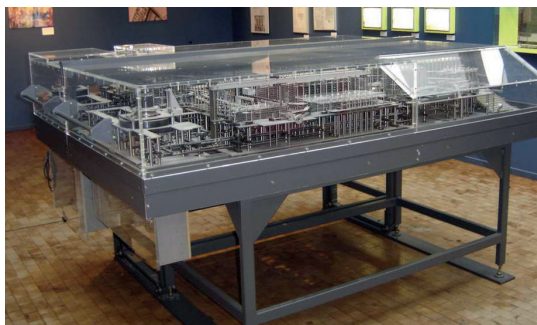
- Discover an under-appreciated pioneer of computer science.
- Plan your next trip to the technical museums of Germany.

If you have any interest in computer history, and possibly even if you haven't, you'll have heard of two of the early computer pioneers: Alan Turing and John von Neumann, who were involved with the machines being developed during World War II. But there's a fair chance that you haven't heard of Konrad Zuse, in Germany — despite the fact that he was achieving very similar things over four years earlier.

Unlike both Turing and von Neumann, Zuse was working in isolation — he had no similarly able colleagues in Germany, and did not of course have any contact with the leading computer scientists and mathematicians working for the Allies. Nevertheless, in the Z3 he built the world's first fully operational electromagnetic programmable computer, in 1941; and came up with the theory of stored-program computation in 1937, several years before von Neumann proposed it.

Z1 and Z2

In 1935, a young Zuse was working as a design engineer at an aircraft factory near Berlin. Much of his time was spent in doing large numbers of calculations by hand, and Zuse, understandably, found this massively tedious. He began to wonder whether he could construct a machine to calculate for him. Working in his parents' flat, he began building the Z1 in 1936, from bits of metal plate and pins. The Z1 wasn't a computer, but a floating-point binary mechanical calculator. It had some programming capacity, and read instructions from holes punched in 35mm film. Zuse filed two patents in 1937, which most importantly included the idea of stored-program computation and what has become referred to as "von Neumann architecture", years before von Neumann himself proposed it. The Z1 was finished in 1938, but it never worked particularly well, as its 30,000 metal parts were not precise enough. It was



Replica of the Z1 in the German Museum of Technology in Berlin. Image: CC-SA, ComputerGeek.



Konrad Zuse in 1992 (he died in 1995). Photo: CC-SA, Wolfgang Hunscher, Dortmund.

destroyed in an air raid in 1944, although a replica is now in the German Museum of Technology in Berlin.

Zuse's next attempt was the Z2, which he built in 1939–40. He had been called into military service, and so had a research subsidy, but initially at least was still working in his parents' flat. The Z2 took up several rooms of the flat when he presented it to the Deutsche Versuchsanstalt für Luftfahrt (DVL, the German Research Institute for Aviation), which rather makes you wonder how big the flat was and how tolerant Zuse's parents were! T

The Z2 was basically an improved version of the Z1, but using 600 telephone relays rather than the metal plates of the Z1. It had a 64-word mechanical memory, and electrical relay circuits for the arithmetic and control logic. It weighed 300kg. It worked better than the Z1, but was still very unreliable — though it worked well for the presentation to the DVL and impressed them enough that they coughed up further funding.

Z3

In 1941, with subsidies from the DVL, Zuse was able to start a company and (finally!) hire a lab to work on his next machine, the Z3. This was a programmable calculator with a memory, which had loops but no conditional jumps (so no if/then logic). Like the Z2, it was relay-based, using 2,000 relays and 22-bit words,

but it was far more reliable. Zuse's co-worker Helmut Schreyer had suggested vacuum tubes to Zuse, as were used in Colossus in 1943, but he dismissed them as a crazy idea. (IBM's Harvard Mark II, built in 1947, used relays, so they were by no means obsolete.) As with the previous machines, the Z3 used punched film for code and data input; it also had a terminal and lamps for input and output. It was Turing-complete (see boxout, right), and as such was the world's first fully operational electromechanical computer. However, Turing-completeness was not of interest to Zuse or his backers the DVL, who were interested only in automating calculations. (It was a similar story with the ENIAC in the US, which was originally intended to calculate artillery firing tables; but the wider possibilities were quickly realised by US mathematicians and scientists. ENIAC wasn't ready until 1945, though, several years after the Z3.)

Like the Atanasoft-Berry Computer in the US (tested in 1942, but not programmable, being designed to solve linear equations), but unlike ENIAC and IBM's early machines (which were decimal), the Z3 was binary. The punched tape system was also ahead of other early computers — Colossus and ENIAC were both programmed with plugs and switches. It was an eminently practical machine, for the time, thanks undoubtedly to Zuse's engineering background. His main aim was to automate engineering calculations, and the Z3 did this admirably. Its primary use at the DVL was analysing wing flutter (vibration in certain flying conditions, which can damage or destroy aircraft). Zuse did ask for funding to replace the relays with electronic switches, but this was considered "not war-important" and denied.

Meanwhile, Zuse was also working on the S1 and S2, which were special-purpose computing machines to calculate corrections to the wings of radio-controlled flying bombs — the precursors to the modern cruise missile.

Z4 and afterwards

The Z3, along with Zuse's workshop, was destroyed in an air raid in 1943, but the successor Z4 (also relay-based) was in a different workshop, and was not affected. It was eventually packed up and moved, half-finished, to Berlin in February 1945, then evacuated to Göttingen where it was completed, after which it was moved again to Bad Hindelang in Bavaria, near the Austrian border, where it was hidden in a shed to avoid its capture by the Allies.

For the next couple of years, Zuse's priority was survival — he sold woodcuts to farmers and US troops to earn money. He began working on the Z4 again in 1948, but electricity was only intermittently available and there was only rarely enough of it to run the Z4. A visit from Prof Stiefel from Zurich led to the Z4 eventually being delivered to the Swiss Federal Institute of Technology in Zurich in July 1950. At the time, it was the only working computer in continental Europe. Zuse formed the company Zuse KG, which

Turing-completeness

A Turing-complete machine is one that can simulate any single-taped Turing machine. In practice this basically means that it can (in theory and approximately) simulate any other general-purpose computer; so it can do anything you expect a "computer" to be able to do. It might, however, take a very long time!

Since the Z3 had no conditional branching (if/then), it is not straightforwardly obvious that it is Turing-complete. In 1998 Raúl Rojas

proved that it was, by proposing a program that instead of branching, would compute both sides of every branch. It would therefore calculate all possibilities, and cancel out the unnecessary ones. In an abstract theoretical sense, then, the Z3 was Turing-complete. In practice, this doesn't mean that it was in any real sense the same as a modern computer, or even a 1940s/50s computer with branching capability. However, the Z4 did have conditional branching.

went on to build a further 250 computers before being sold to Siemens in 1967.



IBM bought an option on his patents in 1946 (Zuse, it seems, might have preferred to work for them directly, but they weren't interested).

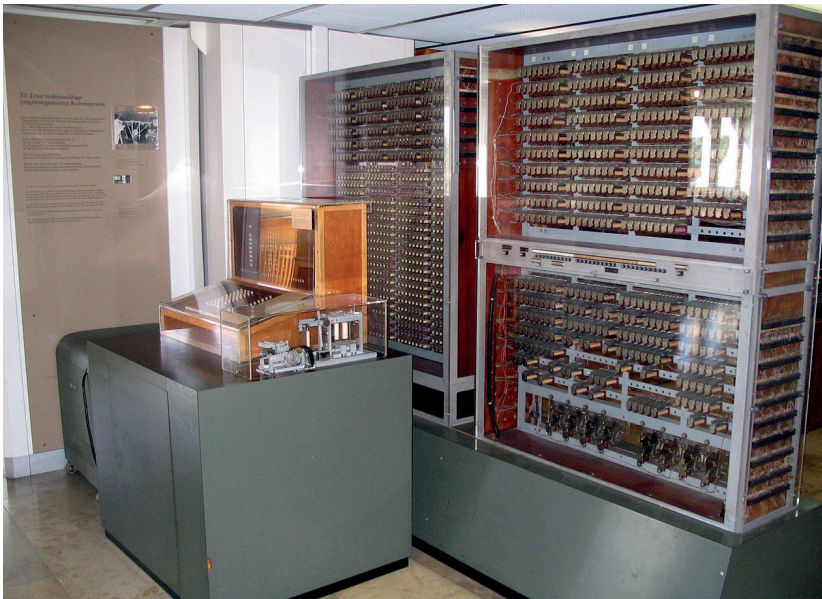
The exact influence of this on IBM's work is unknown, but it is possible that information from Zuse's binary machines were part of IBM's move from decimal and analog to binary and digital computers.

Simulators

There's a really nice Z3 simulator available online (note that the site is in German). It runs there as an in-browser applet, which I wasn't able to get running on my Linux browser. (I could run it on Mac, which is usually pickier about Java, so the applet definitely does work; a different hardware and software setup may be all that's needed.) Alternatively, I was able to run it on Linux by downloading the file **Z3.zip** from the simulation overview page, unzipping it, and running **appletviewer simulation.html** from the resulting folder. I couldn't initially see the film tape part of the main window, but it did reappear after I resized the window, choose Programm > Neu, and hit Ende. (And in fact you can program the simulation without seeing the film tape, although it's nice to see your instructions appear!). If you only have a big purple box in the middle of the top and no 'film' picture, this is the 'Speicherauswahl' box referred to below. Enter your memory locations in here and use the right-hand buttons for operations just as detailed below.

The Z3 is labelled in German. Some of the labels are immediately obvious, but here's a quick translation of some of the others:

 German	 English
Vorzeichen	Sign (positive/negative)
Ziffern	Numbers
Komma	Comma
Wurzel	Root
Einlesen	Read in
Ausgeben	Output
Eingabe	Input
Mantisse	Mantissa (significand)
Speicher	Memory
Rücksetzen	Reset
Fortsetzen	Resume/continue



Replica of the Z3 at the German Museum in Munich. Image: CC-SA, Venusianer.

The registers R1 and R2 are the working registers, and the memory (Speicher) has 64 words available. You can set this manually by clicking the circles. I found the mantissa/exponent setup a little confusing but each line has a decimal translation at the end so you can play around until you have the idea.

The mantissa (or significand)/exponent is a way of describing floating point numbers. For example, a significand of 1234 and an exponent of -1 would describe the decimal number 123.4.

You can either enter a calculation directly, using it in effect as a desk-top calculator, or enter a program. (Sadly, you can't save programs.) As with other computers of a similar age, to run a calculation, you first enter two numbers. These will be loaded into the two working registers R1 and R2; the next instruction is then applied to those registers, and the output stored in R1, ready for the next calculation.

Here's an example of manually adding two numbers, 11 and 2:

- Enter 11 with the top set of buttons (Eingabe).
- Hit the Einlesen button, and watch the circuitry change. Notice that the R1 circle on the bottom left will now be lit.
- Enter 2 with the top set of buttons (Eingabe).
- Hit the Einlesen button again. Both R1 and R2 are now lit.
- Hit Addition, then Ausgeben. The output, 13.0, will appear in the very bottom left.

You can also use the 'film' to enter a program. When you first load the applet, there's a program provided on the film. To run this, go to the applet's Programm menu and choose Start. To enter your own new program, go to Programm > Neu (new). You'll then get an extra three buttons: Laden (load/read from memory); Speichern (store to memory); and Ende (end). You use the purple Speicherauswahl box to enter a memory location, and the buttons to enter an operation code (such as addition, multiplication, read from storage, etc).

Here's how to enter a program to add two numbers:

- Enter 0 in the purple Speicherauswahl box, and hit Laden. This reads from memory location 0.
- Enter 1 in the Speicherauswahl box, and hit Laden, to read from memory location 1.
- Hit Addition. This will add the last two numbers that were read in.
- Hit Ausgeben. This will output the result.
- Hit Ende to finish the program.
- Go to the Speicher window and enter a number in the 0 location and in the 1 location.
- Choose Start from the Programm menu. Your program will run, and you'll see the result (the sum of your two numbers) at the bottom-left.
- To start again, you'll need to hit Fortsetzen (Reset).
- To store the result in a specific memory location, say location 6, you can replace the Ausgeben instruction with

Speicherauswahl 6, Speichern.

Run this (you'll have to re-enter the whole thing), and keep an eye on the Speicher window. You'll see your result show up in memory location 6.

Here's a program to calculate 4! (4*3*2*1):

- In the Speicher box, enter values 1, 2, 3, 4 in memory locations 0, 1, 2, 3.
- In the main window, start a new program.
- Speicherauswahl 0, Laden.
- Speicherauswahl 1, Laden.
- Multiplikation.
- Speicherauswahl 2, Laden.
- Multiplikation.
- Speicherauswahl 3, Laden.
- Multiplikation.
- Ausgeben.
- Ende.

Run the program to get the output 24. Note that multiplication steps take a while! You'll see here the advantage of having the output of each calculation stored in R1 ready to be used. By rewriting memory addresses it should be possible to construct a loop; have a go and see what you can manage.

If you want more information about the simulation, there is an article by Raúl Rojas which discusses the construction of the simulation and includes the instruction set. There are also instructions for using the simulator (in German, but Google Translate does a reasonable enough job) on the Zuse project webpage.

While building the Z4, Zuse concluded that an alternative was needed to programming in machine

Zuse and Turing

Zuse and Turing may have met briefly after the war, in 1947, at a colloquium in Göttingen which included a few other British and German researchers. ('Colloquium' is a polite way of describing a discussion which has also been described as "an interrogation". The participation of the German scientists was almost certainly not optional.) However,

this meeting is only described in Heinz Billing's memoirs, and no details survive. The historical detail is discussed in a paper by Herbert Bruderer. If Zuse and Turing did meet it is likely, due to the secrecy of the war and post-war period, that neither of them was familiar with the achievements of the other, which seems more than a little sad.

code, to make programming more straightforward. In 1945/6, when he was living in the rural Allgäu and couldn't work on hardware, he designed Plankalkül ("Plan Calculus"), which was the first high-level programming language. However, this only existed in theoretical form during his lifetime; a team finally implemented a compiler in the year 2000, five years after his death. Plankalkül has been compared to APL and relational algebra, but it did not in practice have an impact on future languages, since it wasn't implemented at the time. It is, however, the first theoretical description of high-level programming.

Programming in Plankalkül

Zuse's original notation for Plankalkül was two-dimensional, although a linear notation was devised when implementing it in the 1990s. The full report from the Free University of Berlin team is a fascinating read, but here are a few of the basics:

There are three basic types of variables:

- V variables (V0, V1...), read-only, used to pass parameters into programs.
- Z variables (Z0, Z1...), read/write, used for intermediate results.
- R variables (R0, R1...), write-only, used to pass the final results of a program.

Loop variables are also used, written i0, i1, i2, etc. Variables have one of the following types:

- One bit, written 0.
- n bits, written n.0.

Tuples of other types, written (n.0, m.0, ...). So (3.0, 4.0) would be a tuple with two members, one 3-bit variable and one 4-bit variable. Tuples can have two or more elements.

Vectors of a single type: so m.n.0 is a vector (or array) with **m** members each of which has **n** bits. Vectors are used for arrays of the same type, tuples for arrays of different types.

Here's a quick example that adds two numbers:

```
P1 (V0[:8.0], V1[:8.0]) => R0[:8.0]
```

```
V0[:8.0] + V1[:8.0] => R0[:8.0]
```

```
END
```

Note that the report would have **R(V0[:8.0]...)** in that first line, but the online compiler at the Zuse Project website doesn't like that.

After Zuse KG was bought, Zuse wrote the book *Calculating Space*, in which he suggested that the

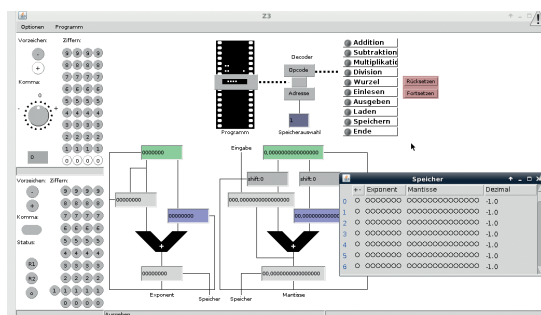


Z4 (the real thing!) on display in the German Museum in Munich. Image: Clemens Pfeiffer, CC-G.

universe itself is running on a cellular automaton. (Von Neumann had an interest in cellular automata, too.) There's no physical evidence against this thesis, and other scientists have expanded on it since. After retirement, he spent his time painting; he died in 1995.

Looking at Zuse's history, it's hard not to make comparisons with Turing, von Neumann, or Hopper, working at the same time in other countries; and to wonder what might have happened if Zuse had been taken more seriously in his own country. Or, more cheerfully, if all of them had been truly able to collaborate in a peaceful world across international boundaries. What would programming languages look like today if Plankalkül had been implemented before COBOL? Would things have moved faster if the Z3 hadn't been destroyed (or if Colossus, in the UK, had been an open project)? Or, on the other hand, did the war drive developments that would otherwise have been much slower? The ethics on all sides are difficult, too; all the pioneers of this time were working on war projects. Zuse, while he was working for the Nazi regime, was never a member of the Nazi Party (unlike many other German scientists of the time). In later life he suggested that scientists and engineers usually have to choose between working for questionable interests (commercial or military), or not working at all.

What is clear is that Zuse was working at the very top of his field, even if he wasn't able to work alongside the others doing the same. His machines were at least two–three years ahead of the teams in the UK and US. Although the Z4, his 'final' version, was finished at roughly the same time as ENIAC and a little after Colossus, it was more programmable than both and genuinely general-purpose. Zuse was an immensely talented scientist whose contribution to computing has gone unfairly unnoticed.



The Z3 window and memory window, in the middle of entering a program.

Juliet Kemp is a programming polyglot, and the author of O'Reilly's *Linux System Administration Recipes*.

Work with Windows users, using the Samba and the Samba Web Administration Tool.

SAMBA: SHARE WINDOWS FILES

Set up file sharing and co-exist in harmony with Windows users.

JOHN LANE

Even the most die-hard Linux fan will at some point find themselves on a network alongside users of other operating systems and will want to share files with them.

Samba is an open-source implementation of the file and print sharing protocol that Windows computers use. It was originally part of the networking suite that Microsoft implemented before they adopted TCP/IP, the networking standard that we all use today, and this legacy brings a certain quiriness to the interaction between Linux and Windows, one of which is having to deal with two name resolution services.

There are two ways that you can exchange files with a Windows system. You can, as a client, connect to another resource on the network to access files or you can set up a server to allow others to connect to you. You'll hear network-accessible filesystems being called shares and, in the Linux world, Samba shares.

Connecting to one of these as a client is very easy these days, because the drivers that you need are now part of the Linux kernel, but you may still need to install the command-line tools:

```
$ apt-get install cifs-utils
```

CIFS is the Common Internet File System, and is what Microsoft calls Samba. It was originally called Server Message Block, or SMB, which led to the Linux implementation being called Samba.

So, if all you want to do is connect to a Windows server to read and write files, it's a simple **mount**:

```
$ mount -t cifs -o username=myuser,password=mypass //myserver/myshare /mnt
```

LV PRO TIP

Apple's OS X uses the same Samba as Linux and can therefore interoperate in the same way.

The 'guest' user

When you connect to a Samba share, you do so as a specific user that, unless you specify otherwise, will be the same as your local username. The server can be configured, like our example is, to provide a guest user and to map unrecognised users to it. This allows access to permitted shares without authenticating. Shares are accessible to guests when their configuration includes: **guest ok = yes**

A quirk of the protocol requires recognised users to authenticate even when accessing shares that are accessible to guests without doing so. You can get around this by mounting with the **guest** option:

```
$ mount -t cifs -o guest //myserver/public_share /mnt
```

Our example configuration sets the ownership of files written by guests to the **nobody** user and **nogroup** group.

You'll need to be logged in as root to use **mount** like that, or you can add an entry to **/etc/fstab** to mount automatically upon boot:

```
//myserver/myshare /mnt cifs username=myuser,password=mypass,users 0 0
```

We use the **username** and **password** options to specify the credentials needed to connect to the remote share. You can omit these if you're connecting to a publicly-accessible guest share. The **users** option allows members of the **users** group to mount and unmount the share without needing root privileges.

smbclient

While you may prefer to mount shares that you frequently use, there is another way to access them that may suit for occasional use or in situations where you aren't permitted to mount. This is the **smbclient** tool, and you'll need to install it from your repository:

```
$ apt-get install smbclient
```

It works a bit like an FTP client; you connect to a host and then use **put** and **get** to send and receive files. You can give the **help** command to see the list of commands available. Here's an example session

```
$ smbclient //myhost/public
```

```
Enter john's password:
```

```
Domain=[WORKGROUP] OS=[Unix] Server=[Samba 3.6.4]
```

```
smb: \> ls
```

```
testfile N 6 Wed May 15 19:32:07 2013
```

```
smb: \> get testfile
```

```
getting file \testfile of size 6 as testfile (0.1 KiloBytes/sec) (average 0.1 KiloBytes/sec)
```

```
smb: \> quit
```

With these methods you can read and write files shared by others, but to share yourself, you need a server – and that's where Samba comes in. There are two versions in popular use: the 3.6 series and the newer 4.x series. The major difference is that version 4 can work as an Active Directory Domain Controller, but that's overkill if all you want to do is share some files. Either version is fine for that purpose and one of them will be easily installable from your distribution's repositories:

```
$ sudo apt-get install samba
```

You configure Samba by editing its configuration file, usually `/etc/samba/smb.conf`. It is formatted similarly to the `.ini` found on Windows systems, so any text editor will do. Here is an example that provides a public share:

```
global
server string = Samba Server Version %v
# Treat unknown users as a guest (where permitted)
security = user
map to guest = Bad User

# For Windows network browsing
workgroup = LVSAMBA
netbios name = MYSERVER
name resolve order = wins bcast

tempfiles
path = /tmp
read only = No
browsable = Yes
guest ok = Yes
force user = nobody
force group = nogroup
create mask = 0755
directory mask = 0755

homes
comment = %U home directory
read only = No
browsable = No
```

Inside the Samba config file

The **global** section is for system-wide settings. Its **server string** is a description that is displayed to clients browsing the network for shares. The **“security = user”** and **“map to guest”** settings cause any unknown users to be treated as a guest. Finally, it configures the NetBIOS Workgroup. This is where the Samba server should appear in the Windows network browser (My Network Places) on Windows clients.

The **tempfiles** section describes a share called **tempfiles**, which gives access to the local `/tmp` directory. The attributes we’ve used in the example are self-explanatory; they are a few of the many available and are documented at <http://bit.ly/smbconf>. You create sections like this for each local directory that you want to share.

The **homes** section in our configuration is special because it shares users’ home directories when they authenticate using their username and password. For a user to be recognised by Samba, it needs to be created with **smbpasswd**:

```
$ smbpasswd -a myuser
```

Note that this sets up a separate password to that stored used by `passwd`. You can then use home directories

```
$ mount -t cifs -o username=myuser,password=mypass //myserver/myuser /mnt
```

It’s a good idea to test your configuration for errors

What is NetBIOS ?

When Microsoft implemented Windows, it used a networking API called NetBIOS (Network basic Input/Output System) that ran over various protocols, but TCP/IP wasn’t used until Windows 95 and, with Windows 2000, Active Directory began to lessen the requirement for NetBIOS, although the My Network Places browser still uses it and it allows older versions of Windows to co-exist on the same network.

NetBIOS includes several parallels to TCP/IP networking, such as the Windows Internetworking Name Server (WINS) that provides name resolution services to NetBIOS clients in a similar way to DNS. If you want to be able to resolve NetBIOS names when mounting shares, you’ll need to install **winbind**

```
$ sudo apt-get install winbind
```

and configure your systems `/etc/nsswitch.conf` to use it by adding **wins** to its **hosts** entry:

```
hosts: files wins dns
```

Samba implements the SMB/CIFS protocol over TCP/IP, either with (on port 139) or without NetBIOS (port 445). The **nmbd** daemon provides the NetBIOS services including WINS server.

The Samba suite gained full Active Directory compatibility in version 4, including the ability to be a domain controller, but it is unnecessary for simple file- and print-sharing.

Samba allows NetBIOS to be disabled, but doing so is only practical if Active Directory is implemented instead. Our examples keep NetBIOS, because this configuration is more likely to suit home or other small networks.

using Samba’s **testparm** command:

```
$ testparm
Load smb config files from /etc/samba/smb.conf
rlimit_max: increasing rlimit_max (1024) to minimum Windows limit (16384)
Processing section “[tempfiles]”
Loaded services file OK.
Server role: ROLE_STANDALONE
```

Samba runs two daemons, called **smbd** and **nmbd**. The former provides the sharing services and the latter provides the NetBIOS name services necessary for your Samba server to appear in My Network Places. Start the Samba daemons with:

```
$ service smbd start
$ service nmbd start
```

You should then be able to browse for the new share from a Windows machine (remember to use backslashes: `\\myserver\tmp`). Or, from a Linux (or other Unix-like) machine, you can use **findsmb** to list servers on the network and **smbclient** to view their shares.

```
$ findsmb
IP ADDR NETBIOS NAME WORKGROUP/OS/VERSION
-----
10.0.2.6 MYSERVER +[LVSAMBA] [Unix] [Samba 3.6.9]
$ smbclient -L MYSERVER
Domain=[LVSAMBA] OS=[Unix] Server=[Samba 3.6.9]
Sharename Type Comment
--- --
IPC$ IPC IPC Service (Samba Server
Version 3.6.9)
tempfiles Disk
```

We’ve covered what is necessary to access shares and provide your own. But Samba enables you to do much more, including auto-configuring home directories and sharing printers. With version 4 you can participate fully in an Active Directory network, and this is something that we will cover in a tutorial in the near future.

LV PRO TIP
You can ask Samba to reload its configuration without restarting. Use **smbcontrol all reload-config**.

THE GUI WAY TO SAMBA

Can configure and use Samba without the command prompt.

JOHN LANE

IV PRO TIP

if you log in to *Swat* as root, its status page will give you buttons to start and stop the Samba daemons.

Swat allows limited access to users without write privileges. They can view the server status and its configuration, browse documentation and change Samba passwords.

If you prefer to use a graphical configuration tool instead of manually editing files, there are various tools available that enable you to administer Samba and access remote shares without opening up a terminal window or text editor.

The first of these that we will look at is called *Swat*, or the *Samba Web Administration Tool*. It's part of the Samba suite but your distribution may package it separately from the Samba server suite. To install it on Ubuntu:

```
$ apt-get install swat
```

Before using *Swat*, bear in mind that it will rewrite Samba's configuration file `/etc/samba/smb.conf`. So, if you have carefully crafted a nicely laid out and well-commented configuration file that you don't want to be overwritten, make a backup before using *Swat*. Another thing to note is that, although it is still part of the Samba suite, *Swat* isn't actively maintained any more and there have been discussions about dropping it completely. That said, it remains a popular choice for Samba administrators because it is useful as a learning tool and as a reminder of what the available options are and their default values.

Swat runs as a web service on port 901 of the Samba server. Point your web browser at, for example,

`http://myserver:901` to see *Swat's* main page. You will need to have a login on the Samba server and use those credentials to log in to *Swat* (Samba credentials created with `smbpasswd` are not used).

The options available to you after logging in will depend on your ability to write to the Samba configuration file. The usual way to gain this right is to be a member of the `admin` group and for that group to have write access to the file. This will need to be preconfigured by a user with root privileges:

```
$ sudo usermod -a -G admin myuser
```

```
$ sudo chgrp admin /etc/samba/smb.conf
```

```
$ sudo chmod g+w /etc/samba/smb.conf
```

A user without write access can browse the Samba documentation, see server status, view the server configuration and change the Samba password for any user that they know the current password for. They can do this on remote Samba servers as well as the local one where *Swat* is running.

The Swat wizard

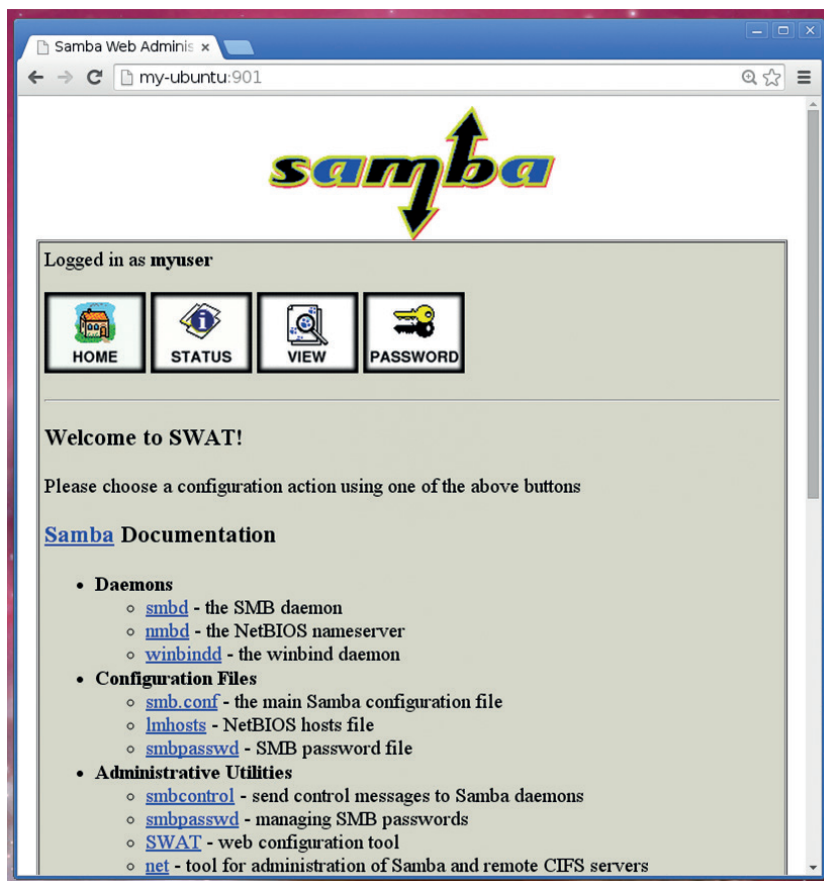
Users with write privileges also get access to the options used to configure Samba. There are screens to edit the global configuration, shares and printers. There is also a wizard to build configuration for you.

Aimed at "the Microsoft-knowledgeable network administrator", the wizard has two options: the first, 'Rewrite `smb.conf` file', rewrites `smb.conf` with the existing settings, ignoring any changes made but not written; the rewritten file will lack any comments or settings that were already Samba defaults. The documentation calls this a 'fully optimised format'. What this means is that it removes all unnecessary comments and any settings that are unnecessary because they are the defaults values anyway.

The other option that the wizard offers allows you to configure a new server, either standalone or as a domain member or controller (*Swat* doesn't offer options for the new Active Directory functionality introduced with version 4). You can select a WINS configuration and choose whether you would like to create per-user home directory shares. After selecting your desired options, click the commit button to write a new `smb.conf` file.

Using *Swat* can help you become familiar with the many available Samba configuration options because its pages display many of them along with their current or default values as well as hyperlinks that take you directly to the relevant part of the documentation. A button on each setting allows resetting to its default. Settings with their default values don't get written to the configuration file.

Because *Swat* exposes many of Samba's configuration options, it can be more overwhelming



for those with little knowledge of them. For basic configuration tasks, other tools may be more appropriate and one such tool is *system-config-samba*.

This is a Python GUI application from Red Hat that enables you to manage shares and users. It has some integrated help pages. You may find it in your distribution's repositories, or you can obtain the source from Red Hat (<http://bit.ly/sysconfsamba>).

\$ apt-get install system-config-samba

If *system-config-samba* is too basic, another option is *gadmin-samba*, part of the *GAdminTools* project. It needs to be run as root and also overwrites the **smb.conf** file, but it does warn about this when it starts. It contains lots of options and overwrites any existing configuration with a more comprehensive one that contains lots of settings, the reasons for which may not be clear if Samba configuration is new to you. You may find the resulting configuration is more complicated than you require (which may not be an issue if you only view it through a GUI application!).

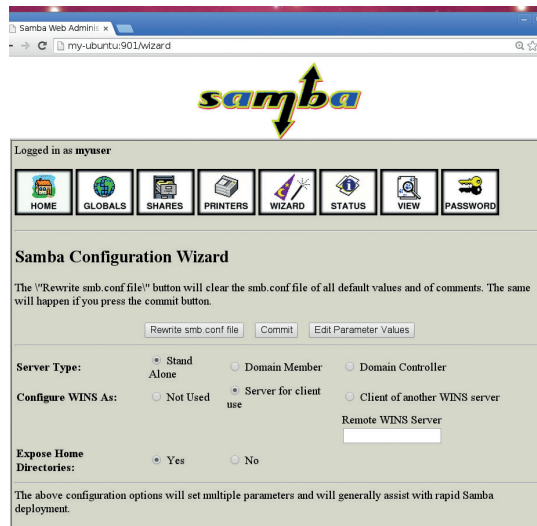
File access

Once Samba is configured, your main interaction with it will be for accessing files shared on the network by remote servers. Samba integrates well into desktop environments and allows shares to be browsed as easily as local filesystems.

Popular file managers like *Nautilus*, *Thunar* and *PCManFM* support virtual filesystems that can directly open remote Samba shares without the need for a separate step to mount them. They use a URI syntax to represent shares, and opening a share is as easy as using its URI, for example:

smb://myserver/myshare

What makes this possible are the virtual filesystem libraries like the Gnome Virtual Filesystem (*GVFS*) and the *KIO* library on KDE. Each file manager also allows you to create shortcuts to frequently used paths, for example by dragging them from the location bar into Places or adding bookmarks.



The *Swat* Wizard provides some basic options that can get you up and running quickly.

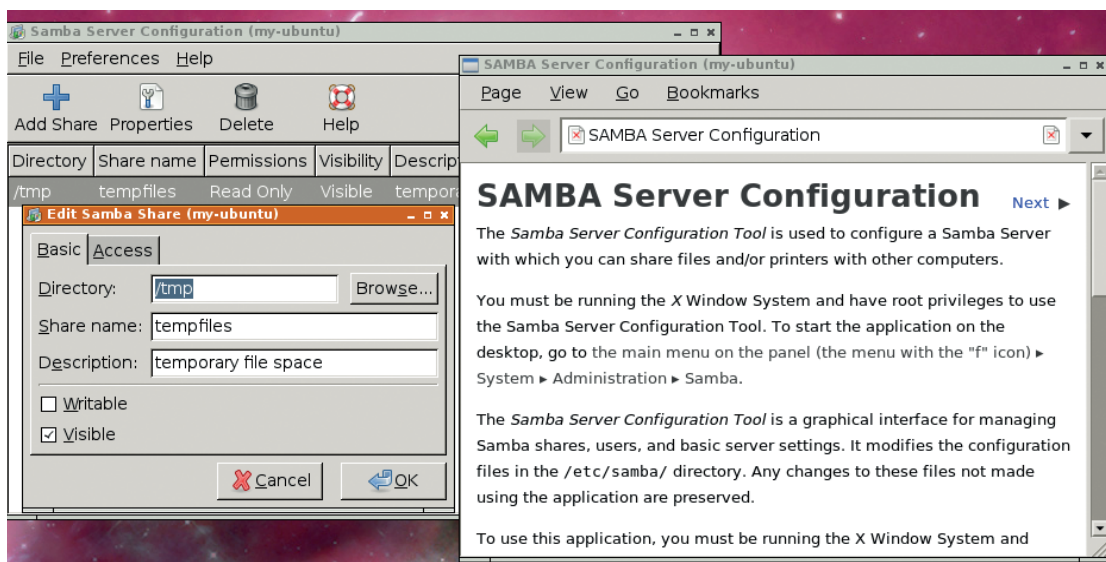
There are also GUI tools that can help if you have a large number of shares to manage. These often support multiple protocols, **smb://** being one of them. Two examples of these kind of tools are *Gigolo* and *PyNeighborhood*, available in many distros' repositories.

They offer a network browser to locate, select and mount shares. Alternatively, they allow remote server, user and share details to be specified manually. *Gigolo* supports multiple filesystem types, whereas *PyNeighborhood* specialises in SMB/CIFS browsing.

Most modern desktop environments have the integrated capability to browse Samba/Windows shares through their file managers and may include graphical administration tools as well. 📖

“Most desktop environments can browse Samba shares through their file managers.”

LV PRO TIP
After changing the configuration, Samba must reload it: the **smbcontrol all reload-config** command performs this task.



System-config-samba has the basic tools to manage shares and users.

/DEV/RANDOM/

Final thoughts, musings and reflections



Nick Veitch was the original editor of *Linux Format*, a role he played until he got bored and went to work at Canonical instead. Splitter!

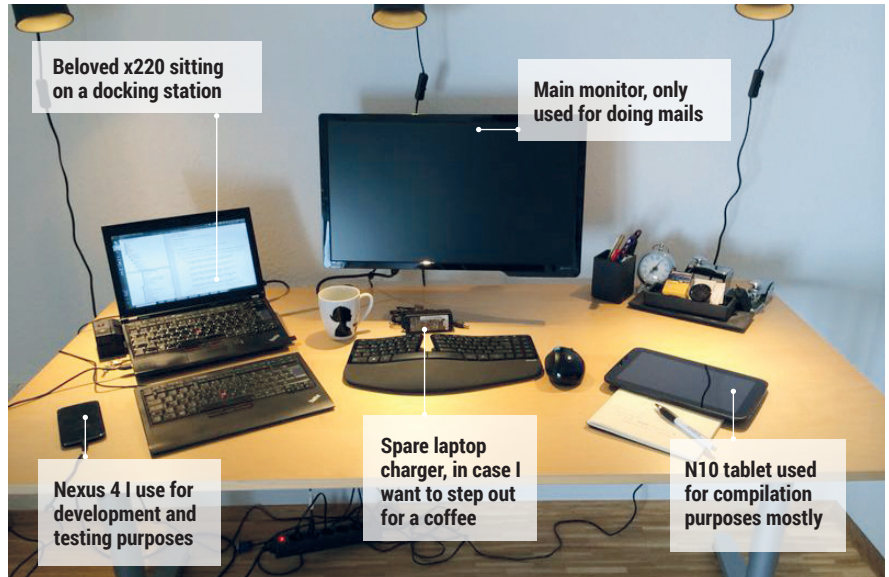
Everyone really is out to get you. Well, a statistically significant number of people are anyhow.

Many many years ago, when I set up my first Wi-Fi network and went out down the road, trying to see how far I could get and retain a connection, I also managed to spy two other networks. One which belonged to a local business and one whose owners (still to be identified) seem to be Star Wars geeks (SSID: Dantooine). Now I can see 14 networks without leaving the front door. A tempting target.

Since those early days, I have always carved off a bit of network as open access. Anyone can log in and take advantage of some meagre bandwidth (I have benefited from individuals opening their access in the past, most notably in Cornwall, where it used to be harder to get a decent signal than to find a dry patch of grass). But last week, someone tried to break in.

Fortunately, they never actually managed to actually achieve anything, so far as I can see, just used the open access to point a bot at the router for 10 minutes or so, trying a dictionary attack. My router is a bit, erm, non-standard, so I am guessing that helped; I don't know what they would have found if they even HAD got access to the main network. I guess they could have printed me out a note on the LaserJet. But my guess is that it wasn't industrial espionage trying to find out what my next feature for Linux Voice was, but some l33t idiots on their summer break looking for mischief.

Anyhow, the moral is that you don't have to be in charge of the Iranian nuclear programme to be the target for "cybercrime". With drive-bys like this, and the modern equivalent of kidnapping (<http://goo.gl/zDBNn3>) already in play, it pays to lock the doors.



My Linux setup **Thomas Voß**

The chief architect of the Mir display manager (see p40) shows us where he does his coding.

Q What version of Linux are you using at the moment?

A Ubuntu 14.10.

Q Which desktop do you prefer, and why?

A Unity – it's nice and easy, and does not get in my way.

Q What was the first Linux setup you ever used?

A Some SUSE version installed from a CD that was supplied with a computer magazine. I have no idea about the exact version, but I remember that I used *FVWM2* back in the day. The installation was far from flawless, and the package manager (I think *Yast*) was horrible to use, but still: I spent days setting up the system to my own liking, fiddling around with configuration files... and really enjoyed the experience! I stayed with SUSE for some time, but finally

switched to Debian and then Ubuntu.

Q What Free Software/open source can't you live without?

A Linux, Unity, GCC and Clang, cmake, the usual set of command line tools, and Chromium. XChat is part of my daily workflow, too. As for editors: I'm pragmatic, I use both Emacs and Vim. For some of my coding work, Qt Creator is my tool of choice.

Q What do other people love but you can't get on with?

A Some people seem to love flamewars/trolling, but I personally could very well live without them. The other thing is playing computer games! I'm totally into the technology driving those games, and I can spend hours and days reading the code of game engines. However, I hardly ever play a computer game in my free time and I stopped considering consoles after the SNES. ☑

HOW TO LEARN EMACS

A beginner's guide to Emacs 24 or later
Sacha Chua (esachac) · sachachua.com/begin-emacs

Questions? I'd love to hear from you!
2013(2)

If you're a developer or sysad...

Learn Vim

It's okay. Learn the basics so that you can easily work on other people's computers. If you know your way around Vim, people won't give you as much grief over Emacs.

Bonus mini-cheat-sheet!
Here's what you need to know in Vi:
Insert mode ↔ `ESC` Command mode
:`vimtutor`
:w write/save file
:q quit
:q! really quit
Exit mode Emacs

Emacs? You're never installing that on my computer!

You can actually edit remote files in Emacs, but that's an intermediate topic. (Curious? see TRAMP)

Be ambitious!

Why Emacs?
- Amazingly customizable
- Endless room for growth

Learn how to learn

There are some old books on Emacs, but the version differences can be rather confusing.

Start with the built-in tutorial instead. Help → Emacs Tutorial

Can't use the menu? Press `Control-h t` to start.

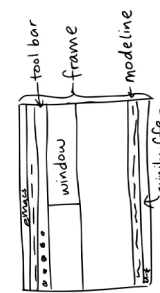
NOTE: The Emacs tutorial has lots of weird terms: Meta key, frame, "buffer". This is because Emacs started a long time ago. Don't worry, you'll get the hang of it with practice.

Some things that might help:

`C-x C-s`: This is how keyboard shortcuts are written.
- Press `Control x` at the same time
- @ then let go of `x` and press `Control s`
Tip: Since you're using `Control` for both keys, you can hold `Control` down instead of letting go between `x` and `s`.

`M-x`: Lets you call commands by name, which is useful if you can't remember the keyboard shortcut. Ex: `M-x help-with-tutorial RET` (usually `Ctrl-h`). This means press `Enter/Return`.

Other resources:
- emacswiki.org
- Lots of resources
- planet.emacsen.org
- Emacs-related blogs
- IRC: irc.freenode.net → #emacs
- Great for help and hanging out



windows show buffers, which could be:
- files
- processes
- other info

Other good help commands:
`C-h i` manual
`C-h k` keyboard shortcuts
↳ gives you help
`C-h f` describes commands for functions
`C-h a` searches for commands
`C-h ?` shows help

LINUX VOICE

Learn Emacs basics

- `C-x C-f` open (find-file)
- `C-x C-s` save
- `C-x C-c` quit

NOTE: You don't need to quit Emacs after each file. Just leave it running and use `C-x C-t` to open the next.

How to select text

Go to the start of your selection and press `C-SPC` (Control + space)

- Go to the end of your selection and run your command
- * Learn how to use keyboard macros. They're awesome.
- `C-x` (start macro)
- `C-x` end macro
- `C-x e` execute macro
- ↳ ...again

Extend & customize

- `M-x` load-theme RET
Try out color themes → use `list-packages` to see what's available
- `M-x` customize-group RET
set common options
- `M-x` customize-face RET
change background, foreground, etc.
- `M-x` list-packages RET
install lots of modules... and then...

editing your `~/emacs.d/init.el` file!
Just use `C-x C-f` to create it!

initializes your Emacs, adds new functionality, and so on.

Use `M-x eval-buffer` or restart Emacs to see the changes.

Break your Emacs Config! → Emacs - a step your customizations

Learn other handy tips

- Buffer & window management
- `C-x b` switch buffer. → given better with `M-x ido-mode`
- `C-x 2` split
- `C-x 3` split
- `C-x 0` other window
- `C-x` get rid of current window
- `C-x 1` get rid of other windows

Navigation & search

- `M-g M-g` go to line (hold `Alt/g` and then press `g` twice)
- `C-s` Interactive search
- `C-r` Interactive search backward
- `M->` End of buffer
- `M-<` Beginning of buffer
- `M-x occur RET` Find lines

Explore!

- Org-mode.org
organize your life in plain text
- TRAMP
remote access
- Calc
powerful calculator and converter
- Eshell/Term
command-line in Emacs
- Narrowing/Widening
focus.

(1) Writing & debugging Emacs Lisp (it sounds scary but it's powerful!)

There's so much more!
You'll be surprised it's not all in the ring!

Oh yeah, install!

Ask away, and discover more by exploring!

Sacha Chua

Matrix

Arm Mini PC

- TBS TUNER SUPPORT
- FREESCALE QUAD CORE
- 100% OPEN SOURCE
- XBMC, VDR, TVHEADEND



NOW ONLY
£119

FREE UK MAINLAND DELIVERY

System-on-a-chip (SoC)	Freescal MCIMX6Q5EYM10AC
CPU	Quad ARM Cortex-A9 at 1.0GHz
GPU	Vivante GC2000, Quad core GPU, Quad IPU
RAM	2GB DDR
USB 2.0 ports	3x USB 2.0
Audio & Video interface	HDMI port 3.5 mm jack
Storage	eMMC 16GB 1x SD card slot 1x TF card slot
Network	10/100/1000 wired Ethernet WIFI IEEE 802.11n/b/g
Power input	5V, 3A



BY YOUR VERY OWN: BEN EVERARD

"XBMC performance is fantastic and could make the Matrix one of the best frontends you could buy"

"The hardware looks good, feels solidly made and works well"



xbmc



ubuntu

SEE PAGES 62 & 63 FOR MORE

WWW.TBSCARDS.CO.UK