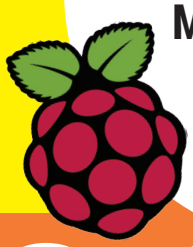
PROUDLY INDEPENDENT SINCE 2013

LINUXVOICE

June 2016 FREE SOFTWARE | FREE SPEECH www.linuxvoice.com

RASPBERRY PI!



Minecraft + Sonic Pi = hours of geeky fun

UBUNTU 16.04

The world's favourite Linux distro launched in 2004...

... and quickly became synonymous with the word Linux.

But its reign didn't last for long, and it faded into the background...

Now Ubuntu is back, and it's come to reclaim its crown...

It's #1 in the cloud, with business and on the server...

... and now Ubuntu is coming to take back your desktop!

It's better, it's bigger, it's back!

MICROSOFT Has it really made its own Linux distribution?
PYTHON Write an image gallery controlled though a web app
GIMP Why this Free Software image editor is more than just a daft name

32 PAGES OF TUTORIALS

ROBERT 'R0ML' LEFKOWITZ

METAL GURU

Literacy, history and coding as a way of thinking – prepare to expand your mind.

NANO PI

BBC MICRO:BIT

Coming soon to a school near you – the tiny machine to make Britain great again!

VAGRANT › OPENBSD › RUST › GIT & MORE!

FREE SOFTWARE | FREE SPEECH
June 2016 £5.99 Printed in the UK
ISSN 2054-3778

FOSSTALK LIVE 2016

A free evening of live Linux Podcasts

Saturday 6 August 2016

LINUX VOICE

ubuntu uk podcast

LINUX LUDDITES

Plus Stuart Langridge and Dave MegaSlippers

http://www.fosstalk.com/tickets/

The Harrison, 28 Harrison Street, Kings Cross, London, WC1H 8JF

Doors 5pm

LINUX FOR BEING HUMAN

The June issue

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.

There's an emphasis on Ubuntu this month, in part to celebrate the release of 16.04 LTS. We often forget how important Ubuntu is for users and for the broader technology industry – it's still a term synonymous with making Linux easy to use. Yet, to its credit, Ubuntu is always changing, unafraid to try new ideas. It's difficult to imagine what the Linux landscape would look like without that garish brown and orange.

On the subject of embracing change, this is going to be my last issue of Linux Voice as the editor. I'll still be contributing, writing and doing the podcast, but I'll no longer be chasing Simon Phipps for his (admittedly wonderful) words on print deadline day. From next month, Ben will have that honour. I know it's a cliché to say great things about one's successor, but Ben's subtle understanding, sceptical insights and unbridled zeal for all things Linux and open source leaves me with no doubt he'll do an amazing job.
Good luck Ben!

Graham Morrison
Editor, Linux Voice

What's hot in LV#027

ANDREW GREGORY
What is Microsoft up to? It's provided lots of open source behind the Microbit (p42). But it's also re-aligning its business, becoming more Linux-friendly. We've delved into the details.
p22

BEN EVERARD
It's easy to think that sysadmin is a specific set of difficult to learn skills, but our tutorial on automating stuff with *Vagrant* shows that's not always the case. It can actually be easy!
p80

MIKE SAUNDERS
In my opinion, Valentine's in-depth look at Linux is always incredible. There's always something new to learn, and this month he really nails the technical side of using *Bash*.
p94

Contents

Bliss was it in that dawn to be alive, but to be a geek, very heaven!

Regulars

Cover Feature

Interview

Feature

FAQ

Group Test

SECRETS OF GIMP TURN TO PAGE 26

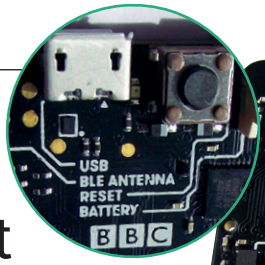
Feature

CeBIT

Mike goes to Cebit

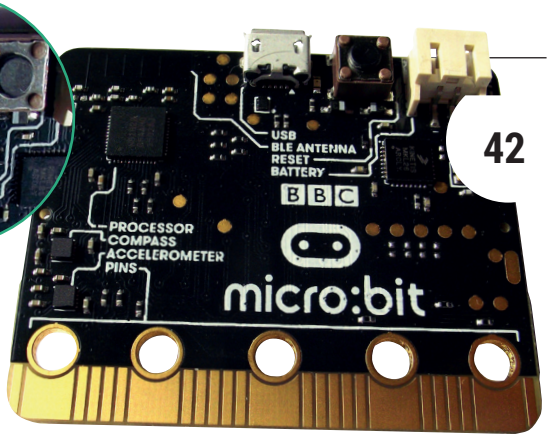
It's the biggest tech trade fair in the world – and it's full of Free Software.

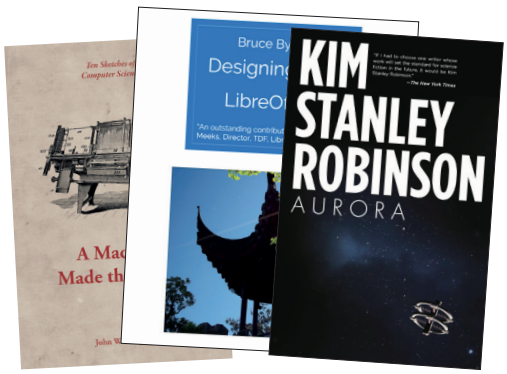
Reviews



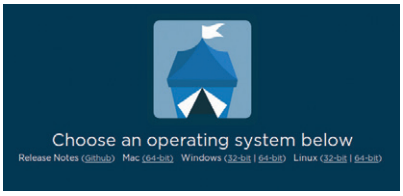
Micro:bit

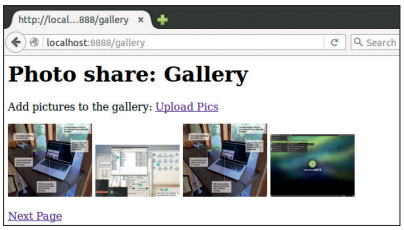
After the success of the Raspberry Pi, the BBC has finally developed its own educational computer, to be given out free to year 7 pupils in the UK. But is it any good for home hackers?



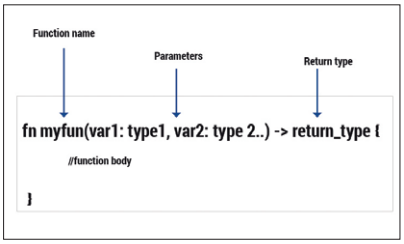


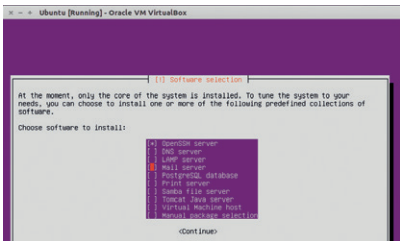
Tutorials





Coding





NEWS ANALYSIS

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

Opinion

Open Source beyond code & licensing

Licensing Free Software is only the start of helping FOSS to flourish in the real world.

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

I'm writing while attending the tenth instance of an unusual conference. The FSFE Legal and Licensing Workshop may not sound an exciting proposition (despite being held this year in Barcelona), but it's a tremendous indicator of the progress that the Free Software and Open Source movement has made.

The event gets a huge range of legal experts attending from the global community, both lawyers and specialists in the dynamics of software freedom. They gather to share their experiences supporting communities and businesses in their use of open source software, exploring new ideas and chewy problems together and sharing and improving tools and techniques that make things better for everyone. It's an expression of the principles of open source, but for legal work and other non-code aspects of software freedom.

Lawyers are not known for collaboration or for freely sharing ideas and resources, but in the slightly safe environment of the Workshop (which is conducted under the Chatham House Rule so the proponents of ideas can't be identified outside the event), they implement the open source approach very effectively. As lawyer Amanda Brock (once general counsel to Canonical) said to me, "If lawyers can make this work, anyone can".

This is all by way of suggesting that, now open source is so mainstream that it's used by almost every business, now open source as a generalised principle is being applied to so many other fields, we have to ask what it actually means beyond licensing and code. What are the attributes of open source we are actually pursuing when we practice it?

> The real win is the unexpected innovation that arises when others take your code, improve it and share their improvements

While Stallman's Four Freedoms articulate how we can retain our liberty in connection with software, there has to be more. Indeed, Stallman just published an essay about "trapped software" ("When free software depends on nonfree") which recognises that the lifecycle of usage, the practices of supporting vendors and other factors have to be considered beyond the licensing of the code itself.

It's not just the code

The need to define open source beyond licensing is also visible at the Open Source Initiative, where a new working group (called Beyond Licensing) has been formed to discuss the attributes of open source projects that we should watch to know the freedoms we get from open source licences have not been abridged by community governance. There are more places this is happening too – the Open Source Hardware movement is also interpreting open source beyond code, for example.

So what is the essence of open source that all these examples are pursuing? I think a primary goal is the unexpected innovation that arises from encounters with people, ideas and technologies outside your own bubble. Having software freedom is a key pre-condition, but the real win is the unexpected innovation that arises when others take your code, improve it and share their improvements – enabling others to then amplify them by building on them. It's also a crucial external indicator that the software freedom in a project is real.

That's also why the cult of open data that exists in the UK government's industry and innovation functions is such a problem. Innovate UK and its associated Catapult organisations are keen proponents of open data, but hardly mention open source and frequently display a shallow understanding of it. Open data is a good thing, as it creates both transparency and opportunities for reuse, but without open source it leads to isolated implementations. With no opportunity for collaboration or for innovation amplification, the scope for unexpected uses is limited to the imaginations of isolated parties. The UK government needs to discover open source not just in its internal administration but also in its innovation investment strategy. The reasons go well beyond code and licensing.

> While Stallman's Four Freedoms articulate how we can retain our liberty in connection with software, there has to be more

Ubuntu • Skype • Torvalds • SCO vs IBM • Gnome • Tablets • Firefox

CATCHUP Summarised: the biggest news stories from the last month

1 Ubuntu comes to… Microsoft Windows?!

What a strange world we live in. Microsoft spent years bashing and fighting Linux, but recently has changed its tune enormously. The company has been working with Canonical to create a system-call compatibility later (rather like *Wine*) which lets unmodified Ubuntu binaries run on Windows. That means you can use *Apt, SSH, Grep, Perl, Apache, GCC, Vim, Emacs* and many other tools without needing to download native Windows binaries. Full details here: **http://tinyurl.com/jdk9lk7**

2 XScreenSaver developer grumbles at Debian

Imagine you've written some software and it gets included in Debian GNU/Linux – pretty cool, right? Well, not if Debian's version is very much out of date, and users are complaining about bugs that you've long since fixed. This is what happened with *XScreenSaver*, and its developer, Jamie Zawinski, decided to jump into a bug report and request that his software be removed from the Debian distribution. This kicked off a bit of a debate, as you can read here: **http://tinyurl.com/j3xpk2o**

3 Skype for Linux – resting or dead forever?

Just a heads-up if you're running Skype on Linux: the software hasn't received updates in a while, and some users are reporting that it won't connect to the network. Time to find an alternative VoIP application…

4 Torvalds: "I am very happy with the Linux desktop"

Linux is everywhere: mobile gadgets, embedded devices, servers and much more. But it still hasn't made a big impact on desktop PCs. Still, that doesn't bother kernel creator Linus Torvalds – in a recent interview, he claimed to be "very happy" with the state of Linux on the desktop, and said that after working for 25 years to bring the OS this far, he'll keep going for another 25. "I'll wear them down" he added, referring to the competition. **http://tinyurl.com/hc9veqg**

5 SCO vs IBM lawsuit returns from the dead

Just when you thought the legal battles between SCO and IBM were truly, finally, completely finished – after 13 years of wrangling – SCO comes back for another round. Back in the day, SCO claimed that Linux infringed its intellectual property and sued IBM for megabucks. In the end, SCO didn't win anything, but keeps dragging IBM in front of the courts. Nobody is quite sure why, or where (what remains of) SCO is getting the funding from…

6 Gnome 3.20 released

Codenamed "Delhi", the new version of the Gnome desktop brings a bunch of new features, including: the ability to install OS updates via the *Software* app, improved support for Wayland, editing functionality in *Photos*, and new "shortcut windows" that list keyboard shortcuts and multi-touch gestures in Gnome's various apps. Expect it in the next round of distro updates, and for the full list of changes, check out the release notes: **https://help.gnome.org/misc/release-notes/3.20**

7 Canonical's convergence tablet up for pre-order

"A tablet when you want it, a PC when you need it" – that's the slogan behind the Aquaris M10 Ubuntu Edition. It's the "world's first convergent tablet", which means you can use it as a mobile device on the go, but when you connect a mouse and keyboard, its interface switches into a more familiar desktop mode. Will it take off? That remains to be seen, but we'll give credit to Canonical for having a stab at this potentially useful type of device. **http://tinyurl.com/jdryjcz**

8 Debian Iceweasel browser reverts name to Firefox

Debian GNU/Linux has included *Firefox* for many years, but due to licensing issues with the browser's name and logos, the Debian team rebranded it as *Iceweasel*. This was potentially confusing for new users, but now the Mozilla and Debian teams have got together and ironed out their differences. The *Firefox* logo has been released under a free licence that's compatible with Debian's guidelines, so in future releases of the distro, *Iceweasel* will be revert to the *Firefox* name.

DISTROHOPPER

What's hot and happening in the world of Linux distros (and BSD!).

Antergos

Arch-based elegance.

Antergos, previously called Cinnarch, is an Arch-based distribution focusing on aesthetics and a comprehensive all-round desktop experience for the average user, with the flexibility of Arch. The distribution is available in two editions: a minimal one with a 530MB ISO and "Antergos Live", which is 1.6GB in size. These are both offered in 32-and 64-bit versions, while the choice of desktop environment is given upon installation, with all the major environments covered in its very nifty *Cnchi* installer instead of the default Cinnamon and Gnome preferences previously offered.

The installer is one of the biggest selling points and offers several extras ranging from the usual proprietary packages to things like Steam and PlayOnLinux, as well as some extra customisability. One of those options is support for ZFS in its partition options during install, adding to its customisability but also raising some concerns recently voiced among the Linux community.

Cinnarch previously used Cinnamon by default, then Gnome and now leaves it to the user.

Like Arch, the distro uses the rolling release model and provides a new snapshot every month or so. Antergos is very aesthetically pleasing, in part thanks to the developers teaming up with the Numix project, which delivers some nice and consistant icons and themes. Also attractive is the notion of having a prebuilt Arch system with a simple installer and variety of desktop environments to choose from out of the box. While there are a few prebuilt Arch systems out there, none quite have the appeal and ease of use of Antergos, which seems to combine the flexibility of the base system for those who want that control, with the usability of the big Debian-based distros for those who just want a no-fuss or beginner-friendly desktop.

Cub Linux 1.0

Easier to pronounce than Chromixium OS.

The Ubuntu-based distro designed to look like ChromiumOS is back and has been renamed Cub (Chromium + Ubuntu = Cub) Linux. Version 1.0 of the operating system is based on Ubuntu 16.04 LTS, and given the stability of the long-term support releases, this seems like a good place to start anew.

The operating system itself uses none of the main components of ChromiumOS except for the browser (and obvious ones like the kernel), and focuses more on obtaining a visual affinity to the distribution rather using it as a base, with the added advantage of being able to use things like *LibreOffice* or Steam which aren't otherwise usable on Chromium. This all makes a lot of sense for Chromebook users who like the UI, but want to get a little extra mileage out of the hardware.

The distro uses *Compton* for rendering, *Openbox* as the window manager and other aspects of the desktop environment are pieced together from pieces of Xfce, LXDE and Gnome. Despite this sounding like a bit of a mess, the layout is slick and differs visually from Chromium mainly in that right-clicking the desktop gives access to another menu with traditional desktop Linux applications, but on the surface appears to look exactly like ChromiumOS minus Google's logo.

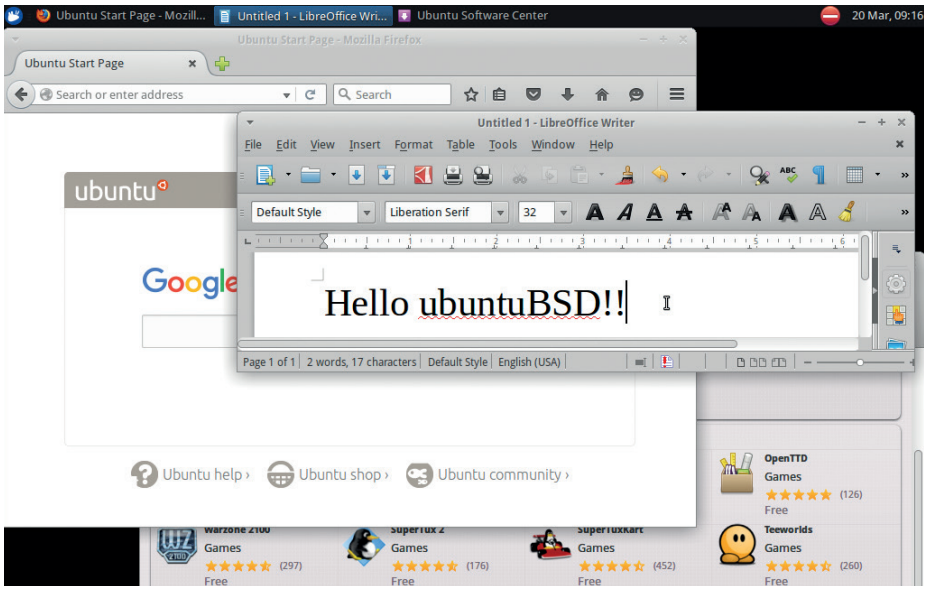
Cub Linux looks almost identical to ChromiumOS, but with the added flexibility of traditional Linux.

News from the *BSD camps

What's going on in the world of FreeBSD, NetBSD and OpenBSD.

If you think you've seen it all before, think again. UbuntuBSD is a project that aims to provide the ease of use and familiarity of Ubuntu with the stability of the FreeBSD kernel. The project – whose slogan is a rather cheeky "escape from Systemd" – is still in the beta stage, but it does indeed look promising, providing an Ubuntu back-end for BSD users. Similar things have been attempted with Gentoo, Arch and the like with little success, but Canonical has said that it intends to treat UbuntuBSD as part of the Ubuntu community, so that may help it buck the trend.

While the name suggests it uses Unity, it actually uses the Xfce desktop environment by default, along with the ZFS filesystem. The system comes with the Ubuntu repositories and official packages and uses APT, so even simple terminal commands like installing packages should be familiar to an Ubuntu user. This looks like an excellent choice for less advanced Linux users looking to take their first steps into BSD without being too overwhelmed, though those users will also want to wait until it's out of beta.



UbuntuBSD running the Xfce desktop environment and Ubuntu software centre.

Installation isn't done through the graphical installer, but one more akin to that used for Debian or Ubuntu server.

On the more mainstream side of things, OpenBSD has dropped support for the VAX architecture, resulting in owners of cupboard-sized computers from the 1970s and 80s being left out in the cold or migrating to NetBSD, which still supports it. Also gone from OpenBSD is Linux emulation through **COMPAT_LINUX**, which was very out of date and supported only 32-bit architecture, so developers have decided to stop maintaining it. FreeBSD 10.3 has also been released in the run up to 11.0, which is expected in July.

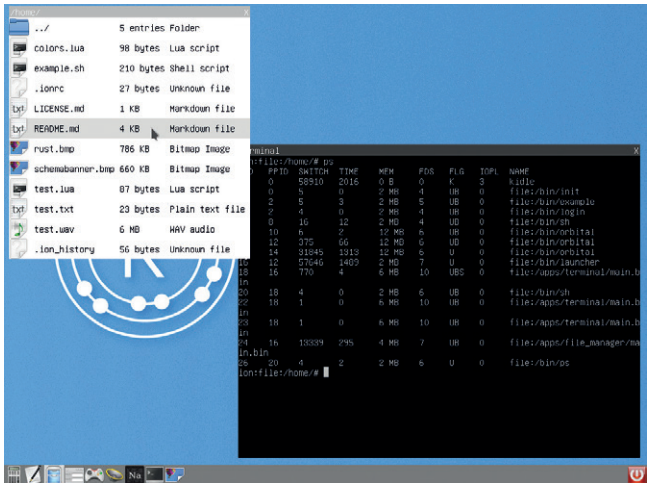
> Ubuntu BSD looks like an excellent choice for less advanced Linux users looking to take their first steps into BSD

Rethinking Linux with Redox OS

Redox is an operating system written in Mozilla's Rust language, intending to provide a Linux alternative. Immediately there are a few very noticeable differences from Linux, such as the use of a microkernel, use of the MIT licence as well as its developers very clearly stating that they will not repeat what they call "bad design choices [...] made by Linux, Unix, BSD, HURD and so on".

This very new project aims to be a "next-gen" operating system, taking the radical approach of doing away with many established Linux traditions, rather than simply re-writing those aspects. For instance, rather than treating every item as a file like in *nix systems, Redox treats every item like a URL, simplifying handlers for events. At the same time, it aims to keep as much compatibility with Linux as possible in order to offer a useful and easy-to-use alternative, attempting to provide support for a lot of software as-is.

The OS comes with an optional in-house GUI called Orbital, and though in the early stages, it can already run on a decent amount of existing hardware as well as *VirtualBox* and *Qemu*, while work is underway in supporting the ZFS filesystem. The ISO is just 26MB in size, and the developers state that Rust is used primarily for security reasons, since it enables many vulnerabilities to be mitigated by enforcing memory safety statically. Another security measure is running drivers in userspace. Redox takes some fascinating approaches in operating system design, and it's well worth checking out the "Redox book" for more details: **tinyurl.com/hau92lk**



Redox OS breaks from many long-standing Linux traditions that the developers see as drawbacks.

YOUR LETTERS

Got an idea for the magazine? Or a great discovery? Email us: letters@linuxvoice.com

STAR LETTER

A SERIOUS POINT

I think your magazine is great and also the website, I've just been listening to the podcast season 4 episode 4 in which you cover the case of the FBI vs Apple, which is in itself an interesting topic. However you introduce the item by saying some guy was arrested and they got hold of his iPhone, when in fact the individual concerned was shot dead by police in a shootout after a terrorist attack where 14 innocent people lost their lives and 22 others were injured. You then carry on the piece in a jovial manner making jokes etc. If you're going to cover something like this on your podcast I feel you should a) Use the relevant facts of the issue and b) show a bit more respect to the people who lost their lives through the behaviour of the individual concerned. I hope you all the best for the future. Sorry if this seems to be nit picking.
Thomas Allen

It turns out that the FBI paid hackers to crack the San Bernardino perpetrator's iPhone – Apple's hands are unsullied.

Andrew says: Nit picking? Not at all. The perpetrators of the San Bernardino attack were killed during rather than arrested afterwards, which is a huge difference.

I believe that our point stands that giving the security services the power to investigate all of our data by default makes us less safe, not more, as it creates a single point of failure that could be exploited in future – whether by other terrorists or by future governments, which will wish to restrict our freedoms even if we can assume that their motives are benign. We should, however, have discussed the events leading up to the technical question with more respect than we did, and for that, I apologise. Thank you for pointing this out.

I STILL HAVEN'T FOUND WHAT I'M LOOKING FOR

I've recently subscribed to the magazine and have all the earlier PDF editions which I have now read and enjoyed. However I occasionally want to reread an article because I am doing something new and I know that the one I seek is buried in that mass of PDFs.

Can we please have an index of all the articles? Preferably updated at least every six months. It should be possible to generate one from your database(s) and divided into topics if possible
Ken Riley

Andrew says: You know, this is an excellent idea. Now that we have over two years' worth of back issues all free to download for subscribers, we should hack something together. It won't be pretty, but it'll be better than nothing.

Damn, look at all this reading matter. We'd better get an index or search system shaped up so we can all find what we're looking for.

CAVEAT EMPTOR

A word of warning – I ordered some stuff from the Free Software Foundation's website on 20 January. After a few months it still hadn't arrived.

So I contacted their sales office, with a view to sorting out lost package compensation. The FSF sales office said I had to apply for compensation. The Royal Mail said that the sender (FSF) had to apply for compensation. Not very good. This means that when you buy stuff from the FSF and it gets lost in the post, you are out of luck.

Fortunately my package arrived today. It looks like they mis-typed my country as "Gabon" instead of "GB" – and yes, I have checked my receipt and I definitely gave the destination country as "GB".

Ian

Andrew says: Is this an illustration of the dangers of autocorrect, not sanitising data input, or something more sinister? Answers on a postcard.

FREE SOFTWARE FOUNDATION

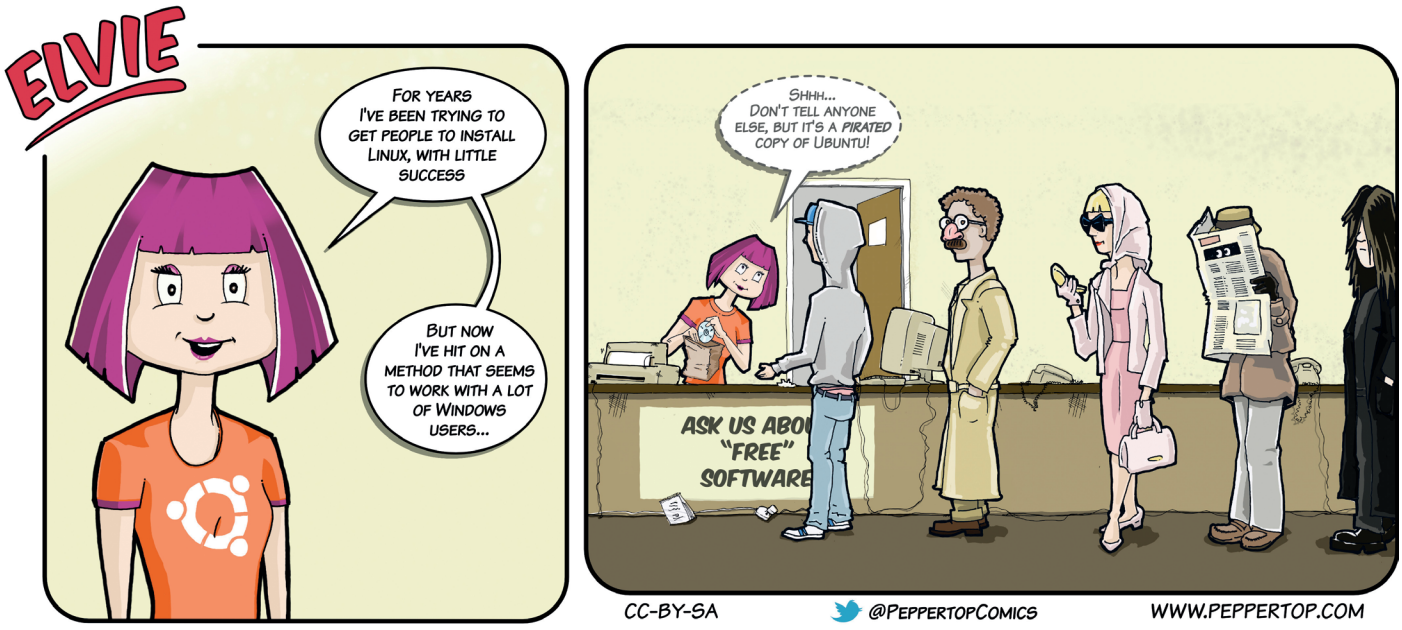
Support the Free Software Foundation – it does a lot of work for the essential liberties that our digital lives depend on. Just remember that GB ≠ GAB.

PLAY ON

Dear Linux Voice, I wanted to thank you for the latest issue, specifically the coverage on FOSS, open source and indie games! That's just what I want to find out more about (especially with the state of consoles like the Xbox One... oh dear).

Stephen Bell

Andrew says: Games are a big deal, a massive industry and a huge drag on Linux adoption that is rapidly shrinking thanks to the increasing quality of gaming on Linux. That's why gaming gets a regular space in the magazine – should this be larger, or are we getting the balance right?



Subscribe

shop.linuxvoice.com

Get your regular dose of Linux Voice, the magazine that:

- Gives 50% of its profits back to Free Software
- Licenses its content CC-BY-SA within 9 months

US/Canada subs prices

1-year print & digital: £95
12-month digital only: £38

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

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

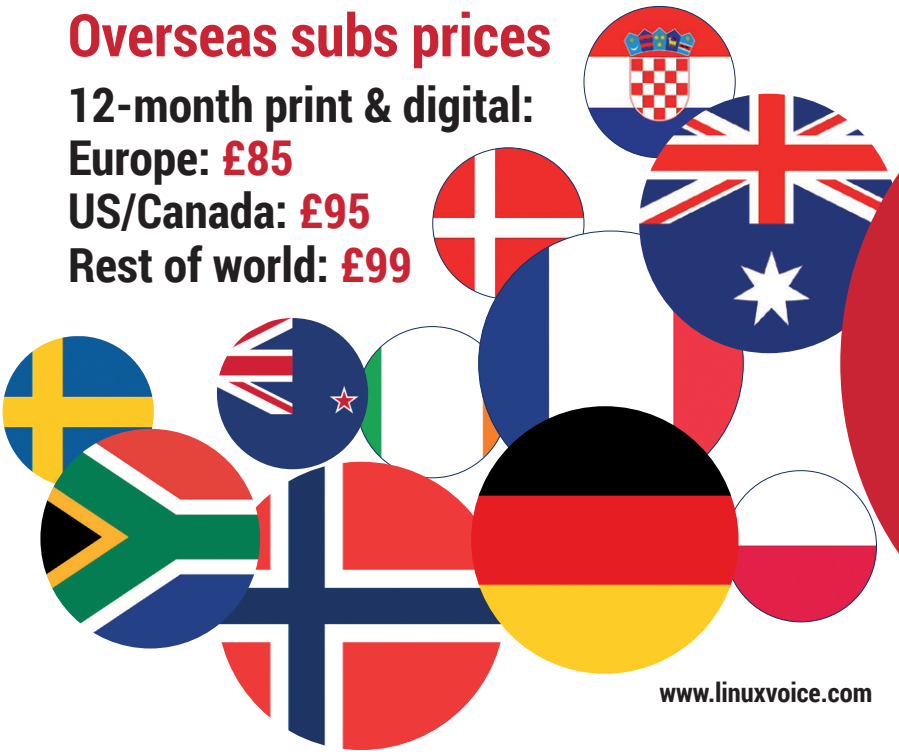
Save money on the shop price and get each issue delivered to your door

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.

All subscribers get access to every single digital back issue – that's about 1,000,000 words of tutorials, reviews and free software hackery at your fingertips

Overseas subs prices

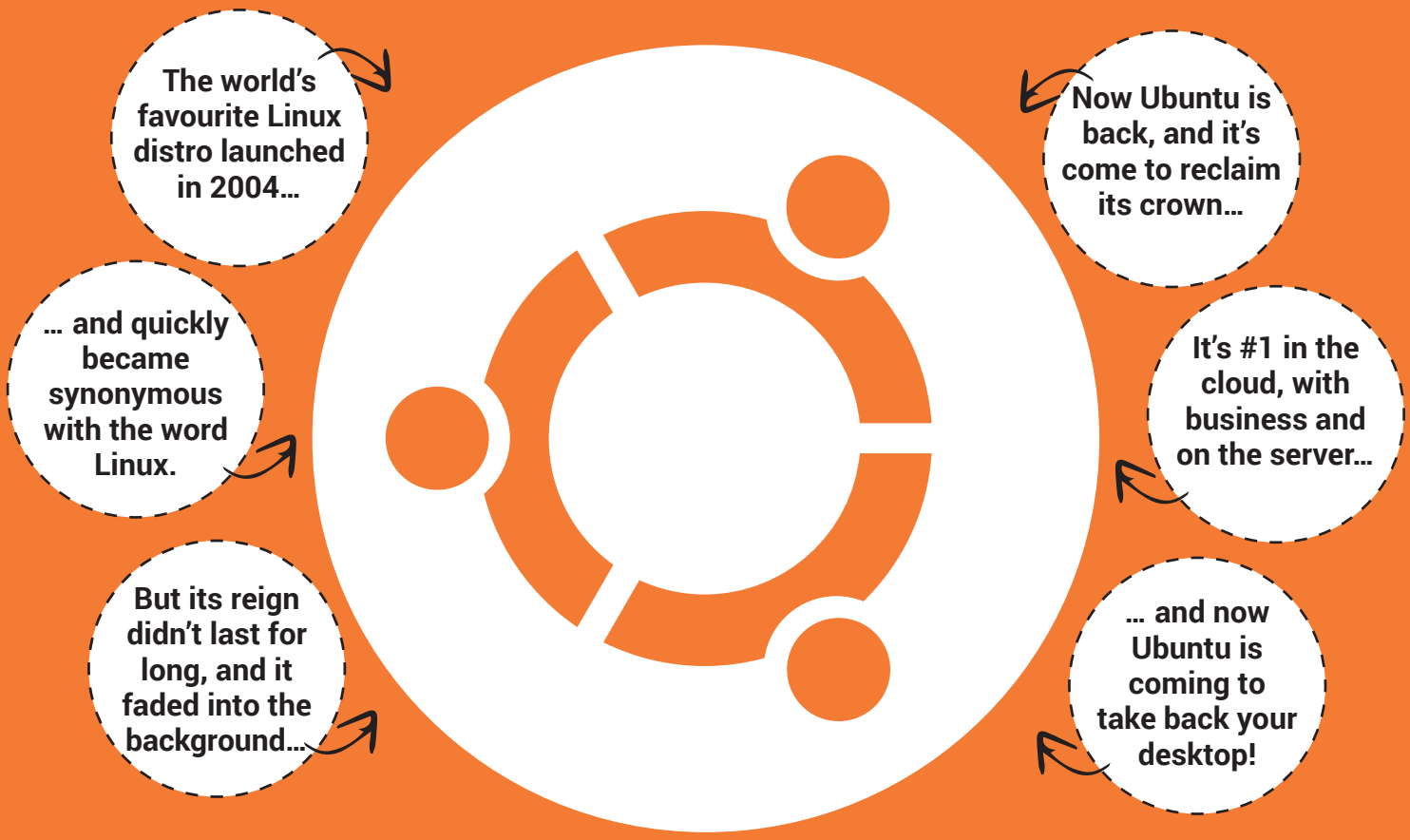
12-month print & digital:
Europe: £85
US/Canada: £95
Rest of world: £99



DIGITAL SUBSCRIPTION*

ONLY £38

*WHEREVER IN THE WORLD YOU ARE – IT'S DIGITAL, SO THERE ARE NO POSTAGE COSTS



UBUNTU 16.04

It's back – and it's brilliant.

Ben Everard fell out of love with Ubuntu – but the latest release has him well and truly convinced again.

Once, the word Ubuntu was synonymous with Linux. For many people, it represented the pinnacle of what Free Software could be; technical excellence married with ease of use. From the first release (Warty Warthog in 2004) until the end of the decade, Ubuntu continued to grow in terms of popularity and mindshare. Then, in 2011, something changed. For the first time in its history, Ubuntu slid down the Google Search trends, and it lost the top spot on the Distrowatch chart to Linux Mint.

It would be easy to look to Ubuntu to see what changed to lose users, but the truth behind its slide in popularity is a little more complicated. 2009–2011 marked a turning point in the history of computing, and a whole array of things changed quickly at almost every level of computing. Smartphones changed from curiosities to almost ubiquitous powerful computers; PC sales started to drop for the first time; and cloud computing became a popular way of managing servers. At the same time, the user experience on most Linux distributions had improved to the point where it was difficult for any distro to stand out from the crowd.

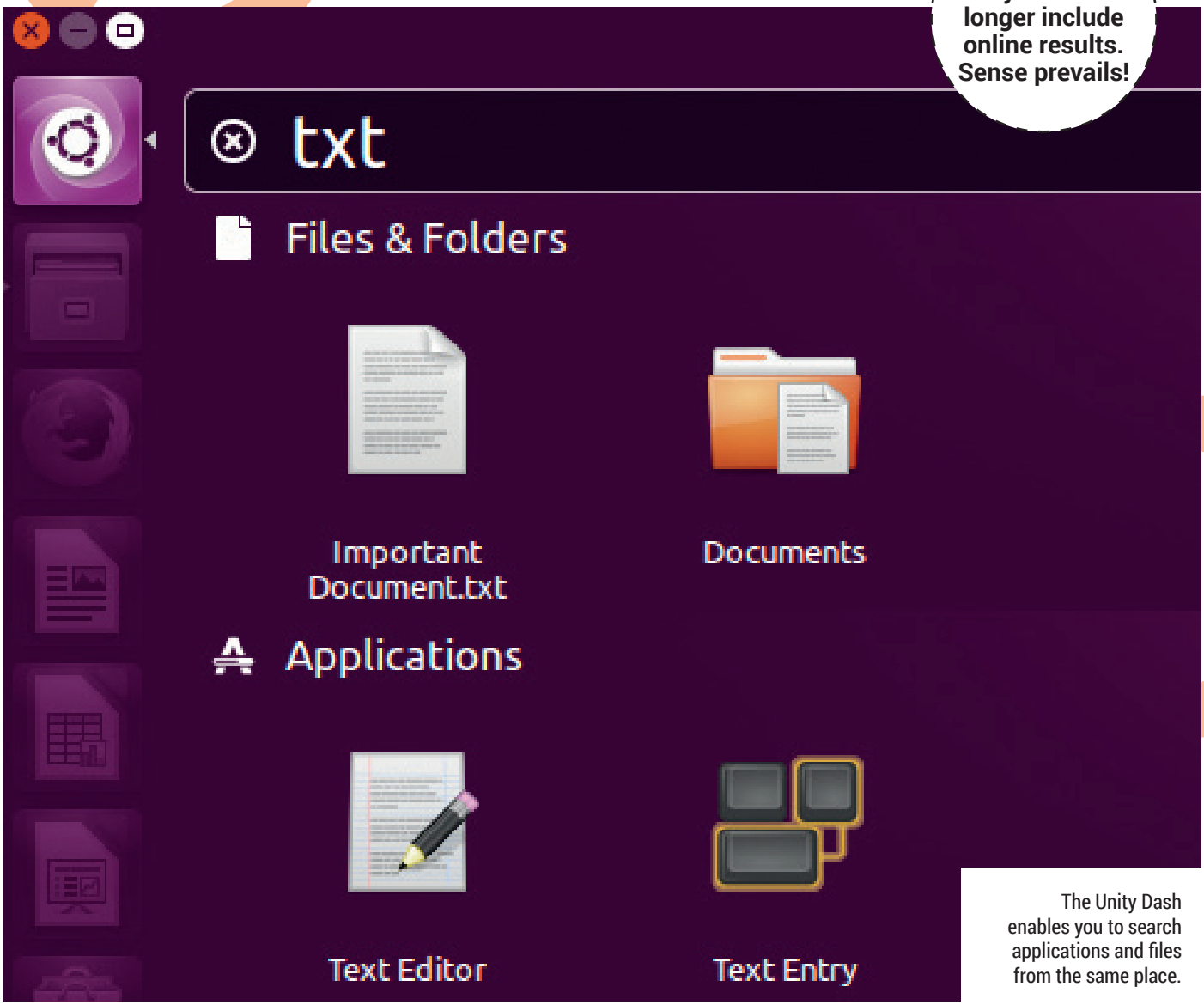
A change is gonna come

In Ubuntu, mature, well-understood, but aging technologies have made way for newer, less established tech. Releases 11.10 to 15.10 fell into the middle ground where the advancements weren't yet mature enough to pay off, but the older tech had already gone.

This can't go on forever, and in order for Ubuntu to stay relevant, it needs something to show for all this effort. With long-term support releases coming out every two years, if 16.04 doesn't deliver, will anyone remember the name Ubuntu by the time 18.04 comes around? We have our own ideas...

16.04 ON THE DESKTOP

A visual return to form.



The Unity Dash enables you to search applications and files from the same place.

There are two things about Ubuntu 16.04 that may make Ubuntu-haters reconsider their position: online searches are no longer included in the Dash; and Gnome 2 (in the form of Mate) is included for the first time for in an LTS launch since 10.04).

While Mate is an official spin, we'll look at Unity first, as this is still the default desktop environment. The biggest new feature of Unity in 16.04 is some added customisation. You can now change the position of the launcher from the left side for the screen to another edge, set it to auto-hide and change the size. Let's be honest, Unity is never going to appeal to tinkerers. If you like to tweak things to make them unique to you, then there's nothing for you in Unity – and that's OK.

> If you like to tweak things to make them unique to you, there's nothing for you in Unity – and that's OK

Mir

Many words have been written, said and shouted about the future of display servers on Linux over the past five years. The short version is that the old X Windows system is being replaced, and most Linux distros are moving to Wayland while Ubuntu is creating its own graphics server called Mir. However, it's very early days for both Wayland and Mir, and most stable distros – including Ubuntu 16.04 – are still using X.

If you want to try out Mir (and Unity 8), there's a package in the repository called **unity8-lxc** that installs a containerised version for you to try. This is available in all Ubuntu versions since 15.04, so you don't have to upgrade to the latest version to try it out.

At the time of writing, Mir is expected to be the default display server in Ubuntu 16.10, but Mir's release has been pushed back in every release since 13.10, so we're not overly confident that it will come out in 2016. Users should hardly notice, but the change in display servers is such a big change behind the scenes that we'd rather the devs take their time.

AMD graphics

The driver stack for AMD graphics cards is shifting from closed source drivers to an open source driver stack. The launch of 16.04 falls in the middle of this process with the older closed source drivers not supporting newer software but the open source stack not yet fully capable. The result of this is that AMD graphics cards won't be quite as quick or capable (particularly with regards to newer OpenGL features and OpenCL) in 16.04 as they were in older releases of Ubuntu.

This should only be true until the work has finished on the AMD open driver stack which should be towards the end of 2016. At this point, the newer drivers will be brought into Ubuntu 16.04, and AMD cards should work as well as they did on older Ubuntu releases. If you rely on an AMD GPU, it's worth waiting for this work to finish before switching to 16.04.

The official Kubuntu release of 16.04 is the first LTS release to use Plasma 5.0

If you don't like the launcher on the left of the screen, simply move it to the bottom.

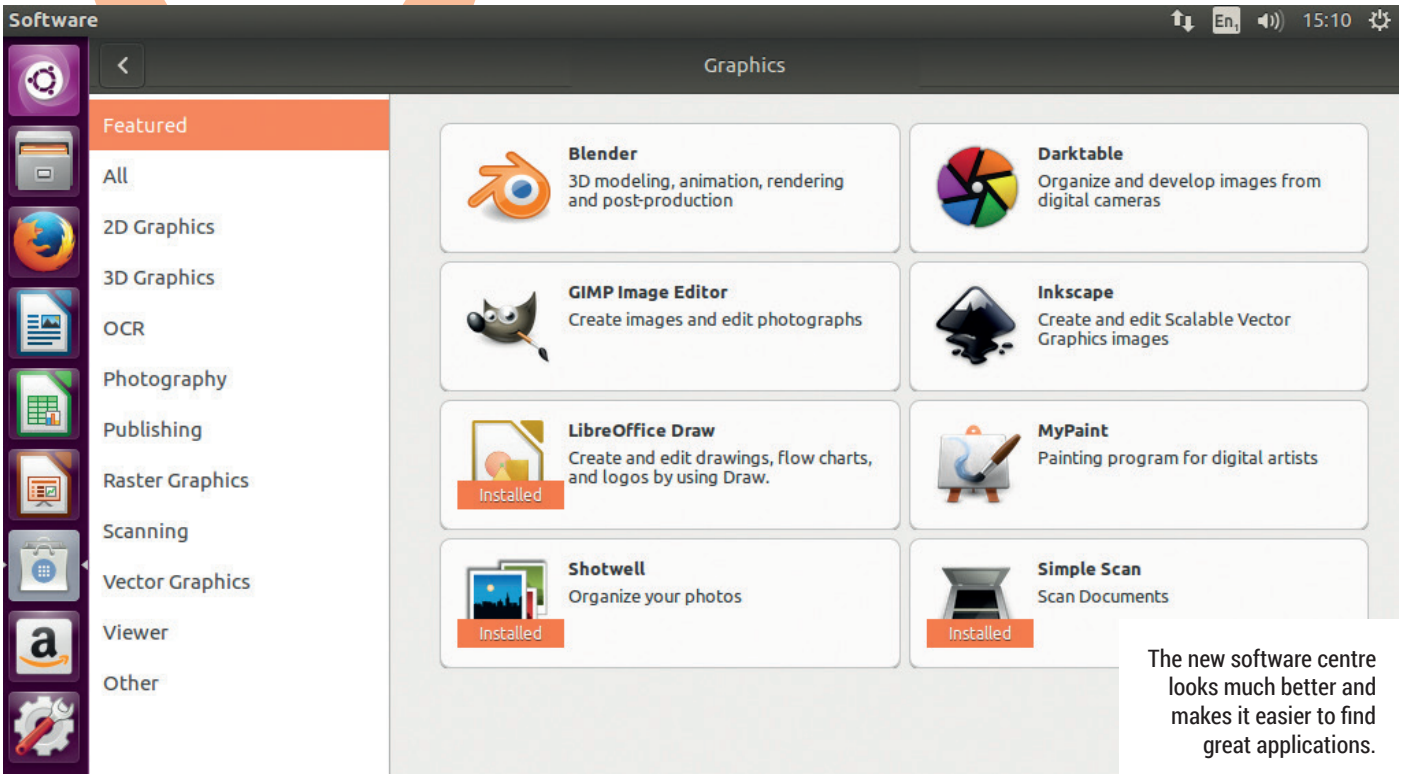
Unity's real strength is creating a desktop that's both simple for beginners and complex enough for power users. Beginners are accommodated through the simple Launcher and Dash combination, while power users are accommodated through the key combinations that enable you to do almost anything without moving your arms off your wrist rest. This keyboard navigation goes beyond simple window management and permeates the applications themselves through the Head Up Display (HUD). Tap the Alt key and you'll get a text-search box for the current application's menus.

This means no more searching through hierarchical menus trying to find the option that you're sure you remember seeing somewhere, but doesn't seem to be where you expect (*LibreOffice Calc* graphs, we're looking at you – why aren't you in the Data menu like the pivot table?).

The launcher bar combines application shortcuts with the task manager in a way that makes it easy to launch common applications and switch windows with the keyboard. Alt + <num> opens the corresponding window counting down from the top. Since the applications with shortcuts are always locked to their position on the launcher regardless of what other applications are open, this makes it really easy to switch. On this writer's setup, for example, *Firefox* is the second icon on the launcher, so Alt+2 will always switch to *Firefox*. These aren't new features in 16.04, but they've been added gradually and, when combined with the new flexibility, make an impressive desktop.

A bigger software ecosystem

The only major change in the applications that come with Unity is the switch from the Ubuntu App Store to the Gnome Software Centre. The new



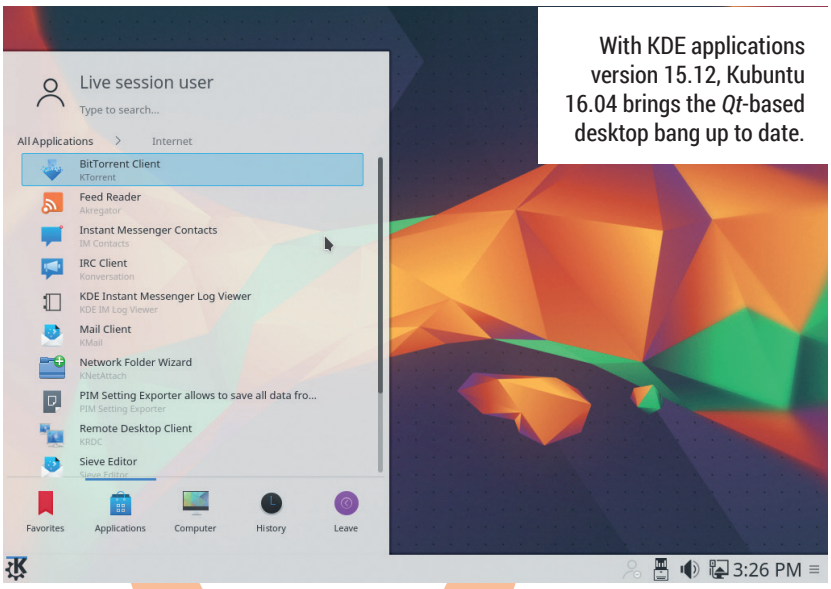
The new software centre looks much better and makes it easier to find great applications.

Mate isn't a fork of Gnome 2, it's a continuation. It's still getting updates and new features, but it's keeping the tried and tested look.

Software Centre works in much the same way as the old store – it enables users to browse and review software – but it looks nicer and seems to be attracting more developer attention.

This release is the first LTS Ubuntu to launch with a Mate spin (this desktop has since been added to 14.04). This is symbolically important because Mate is a continuation of Gnome 2, and the removal of Gnome 2 from Ubuntu (replaced by Unity in 11.04) brought much criticism from the community. Although Unity is still the default interface, having Mate as a first-class desktop in the Ubuntu family will go a long way to appeasing desktop traditionalists.

> Having Mate as a first-class desktop will go a long way to appeasing desktop traditionalists



With KDE applications version 15.12, Kubuntu 16.04 brings the *Qt*-based desktop bang up to date.

Snappy

In almost all cases, Linux distros' security works on a per-user basis. There is one or more user, and each user has particular permissions – they can execute certain pieces of software and alter particular files and directories. Any software runs with the permissions of the person running it. In this way, any software you run can alter any files in your home directory, or access the network.

For along time this has been the way most operating systems – not just Linux or other Unix-like systems – have managed permissions. However, many mobile OSes – most famously Android and iOS – work in a slightly different way. They require applications to explicitly state the type of activity they want permission to use and everything else is blocked. For example, a word processor may need access to the Documents folder, but not network access.

The Snappy packaging system works in a very similar way to these phone OS software managers; in fact, it's derived from the Click packaging system built for Ubuntu Touch. With Snappy, software that you install doesn't integrate with the whole system, but remains standalone and can only perform the actions explicitly allowed by the package.

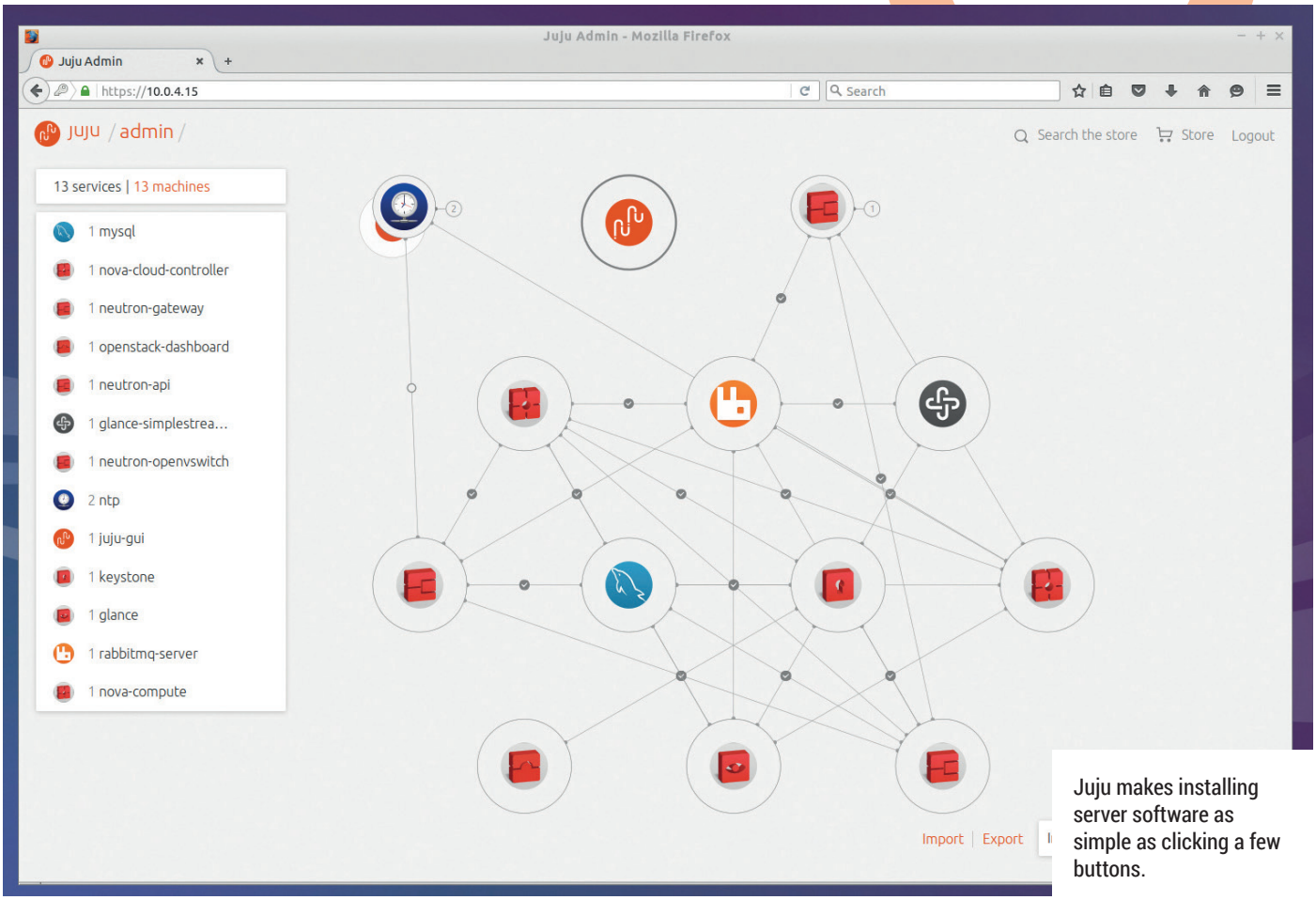
With Snappy, packages include all the libraries needed along with the executables into a single package that can be installed. Upgrades to this package are atomic, which means that if anything goes wrong, the upgrade can be easily rolled back to a working version.

Snappy has both good and bad points when compared with the traditional packaging method. The downside is that if there's a vulnerability discovered in a library, every package that uses the library needs to be updated. The upsides are that a package will never get broken by an update to some other part of the system, and the increased security focus will mean that a compromised package will not mean a compromised system.

Snappy is available on all versions of Ubuntu, although it's most widely used in Ubuntu Core, which is a minimalist version of Ubuntu designed for Internet of Things (IoT) devices, which are less likely to receive software updates than desktop computers. That said, there are quite a few pieces of software available for Snappy (**https://uappexplorer.com/apps?type=snappy**) that are for server and desktop use.

ON THE SERVER

Software running in the cloud.



Juju makes installing server software as simple as clicking a few buttons.

When it comes to Linux servers, there are two types of people. Firstly, there are those who like servers to continue to work as they have for the last couple of decades or so. These people run web servers or database servers and have an unholy knowledge of the syntax of the configuration of every piece of server software ever released.

The second type is chasing the latest server tech. They like OpenStack, Docker and containers. To this sort of person, a computer is never just a computer, it's a hive of many virtual machines, and a part of the cloud. Here at Linux Voice, we're not going to judge, because we understand that both of these types of people have and important role to play. We run our own servers in a traditional style, but we can certainly see the benefits of the new style, at least in certain types of servers.

The biggest bit of news will be a boon to both types of sysadmin: Ubuntu 16.04 will be the first major Linux distribution to come with the ZFS filesystem installed and ready to go. There isn't a default filesystem for Linux, but there are two next-generation filesystems that are starting to be more widely used: ZFS and BTRFS. Sun Microsystems originally developed ZFS for OpenSolaris, while BTRFS is written specifically for Linux. Both are copy-

> Ubuntu 16.04 will be the first major Linux distro to come with the ZFS filesystem installed and ready to go

ZFS and the law

The Linux kernel is released under the GPL v2 and ZFS is released under the CDDLv1 licence. Both of these are open source licences, because with both of them you can modify the source code and redistribute your changes. However, with the CDDL you are permitted to include non-open source code in a derivative product, provided that the parts originally covered by the CDDL are still open source. With the GPL, this isn't allowed. The GPL also requires the entire work to be covered by the GPL exclusively, so licences aren't compatible with the GPL unless they allow the work to be relicensed under the GPL, which the CDDL doesn't. It's therefore a breach of the licence terms to combine CDDL and GPL software in the same software product.

Canonical claims that ZFS is an entirely separate piece of software to the Linux kernel, and the fact that it's loaded by the kernel is irrelevant to this. It doesn't really matter what the lawyers at Canonical think or what the authors of the GPL thought when they wrote the GPL – it only matters what a judge or jury think should the case come before a court. The arguments both ways depend on subtle interpretations of the technical situation as well as the wording of a licence that's never been tested in court before.

From a user's point of view, it's important to realise that the terms of both licences only kick in when the software is distributed. This means that whatever the outcome of this, no one using ZFS in Ubuntu will find themselves in legal trouble.

on-write filesystems that enable you to create snapshots of the filesystem (or a portion of it) at a point in time in a way that's very efficient with both time and space. They both also manage filesystems across multiple physical storage devices far better than older filesystems.

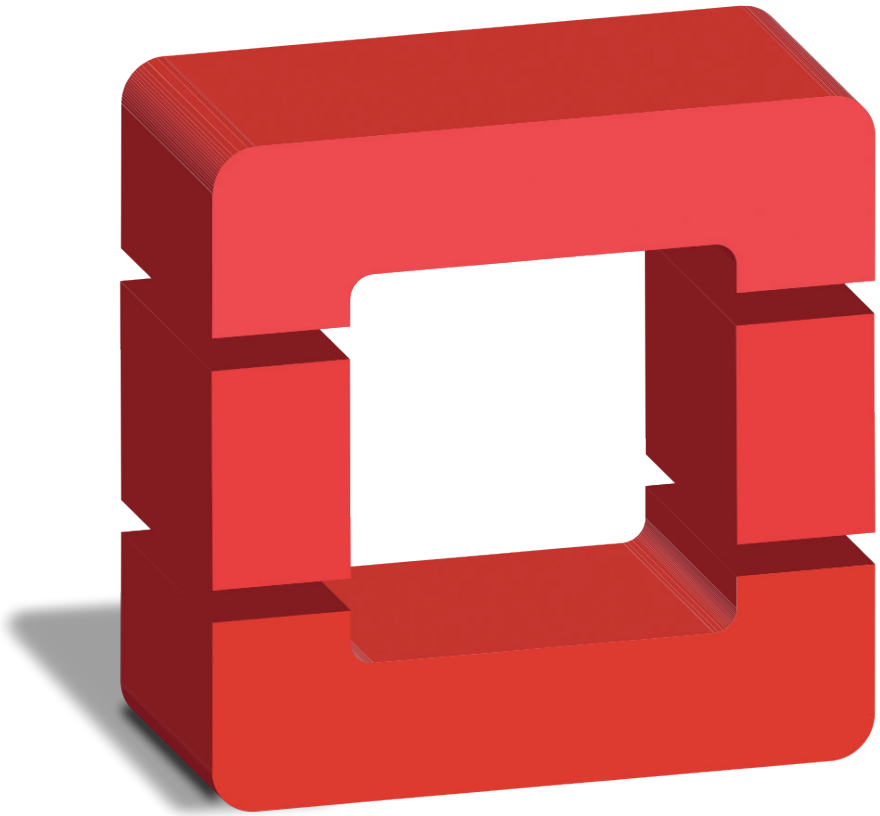
Both BTRFS and ZFS are very capable filesystems, and both are significantly better than the other options for Linux. The biggest difference between them is that ZFS has been widely used in production for several years while BTRFS only just on the cusp of becoming stable in 2016. In 2016, we consider ZFS to be the better choice because it's far more production-tested than BTRFS. Having this filesystem included by default in Ubuntu is a major advantage to this distro, and one that will put Ubuntu at a significant advantage over other distros until either they ship ZFS or BTRFS becomes more stable.

While the inclusion of ZFS is an update that differentiates Ubuntu from the majority of Linux distros, another change brings them into line with the general consensus among enterprise distros. *Systemd* has been around for a couple of releases, but, let's be honest, no one runs non-LTS releases of Ubuntu server, do they? Ubuntu 16.04, then, will be the first release of Ubuntu Server that's widely used to ship with *Systemd*.

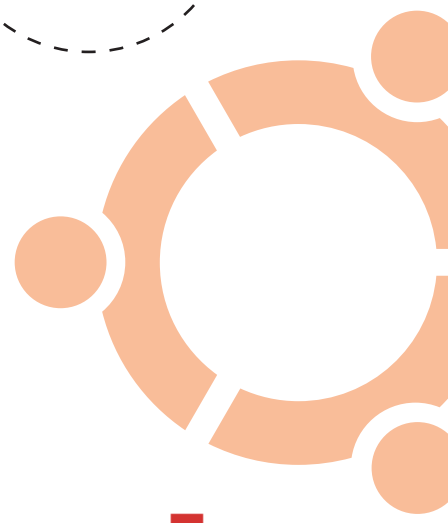
Ubuntu on Windows

Canonical (the company that develops Ubuntu) has joined forces with Microsoft to create a version of Ubuntu that runs on top of Windows. This uses an emulation layer to enable the binaries for Ubuntu command line tools (including the *Bash* shell) to run on Windows 10. If you open *Bash* in Windows, you'll see the familiar commandline environment, and a Linux-like directory structure. Windows filesystems can be mounted in the **/mnt** directory as though they were on a separate partition.

This whole setup allows Windows users access to the whole Gnu toolset running natively on Windows. At the moment, this supports primarily command line tools (including **apt-get**). We wouldn't like to speculate whether support will ever come to graphical tools, but at the very least, this isn't likely to happen soon. This is great news for Windows users, who get access to a whole new set of powerful tools, and it's good to see Free Software spreading, even if it is only as applications on top of a closed-source kernel.



The first version of Ubuntu, codename Warty Warthog, launched in October 2004



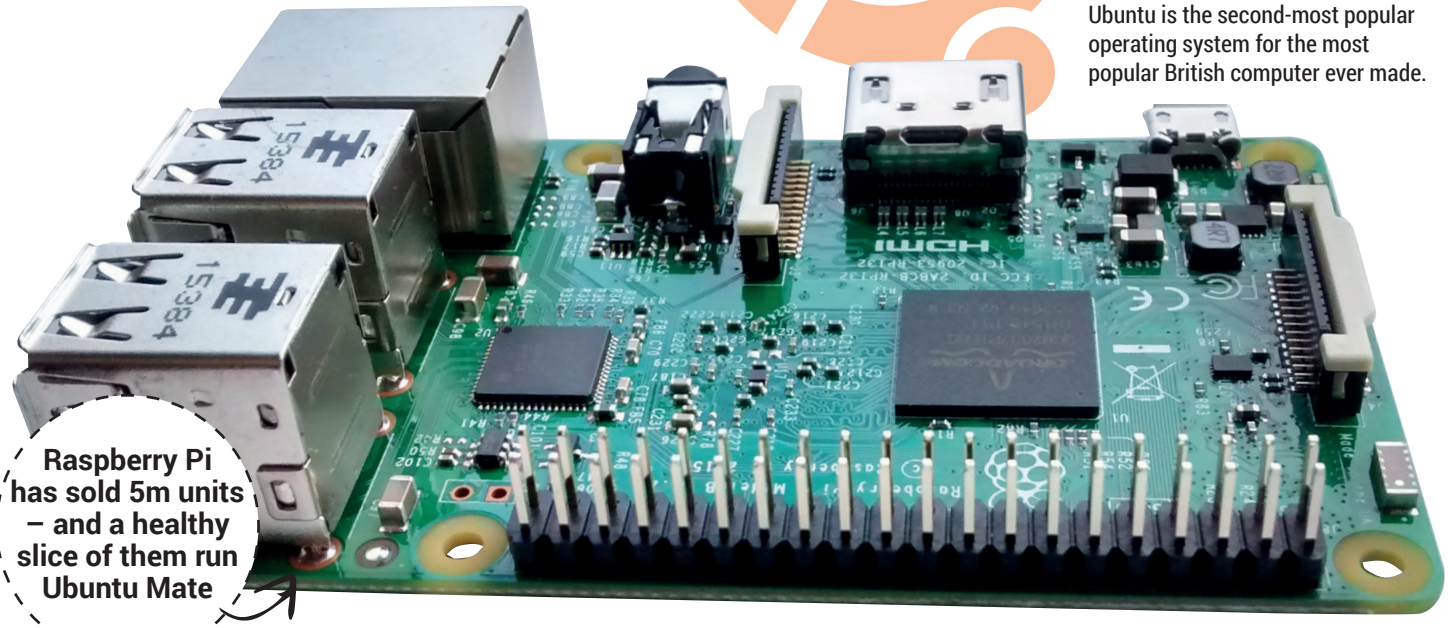
Ubuntu is the world's most popular platform for OpenStack.

openstack®

CLOUD SOFTWARE

DEVICES

Beyond the PC.



Ubuntu is the second-most popular operating system for the most popular British computer ever made.

Raspberry Pi has sold 5m units – and a healthy slice of them run Ubuntu Mate

Angry Tux

No distro can please everyone, so in the interests of fairness, we searched the Linux Voice server room looking for our grumpiest team member. We stalked him through the pre-dawn haze before he'd had coffee in order to catch him at his weakest and found out his thoughts on the new Ubuntu release. Here's what he had to say:

"I've just checked my calendar and it really is 2016. I had to make sure because we're praising a desktop environment for allowing users to move the launcher to different parts of the window. Yes, it's a good feature – I really liked it when it came to KDE some time in the late 90s. And while we're here, let's just take a minute to consider that we're talking about the removal of spyware as a good thing and not running screaming because it was put there in the first place.

"Now, a lot of people are getting excited about ZFS, but really, why is Ubuntu sticking in a potentially illegal filesystem rather than contributing to the effort to complete the next-generation Linux filesystem BTRFS? Yes, ZFS is good, but BTRFS will be better, and it's GPL, it will work across all Linux distros and it's almost production ready. If Canonical put engineers to work on BTRFS in the same way they've put lawyers to work on ZFS, we'd all be better off.

"Juju and Snappy? Have we as sysadmins really deteriorated to the point where we need a fancy GUI to install applications? Since when was **./configure && make && sudo make install** too hard to type? Need to replicate a procedure across a wide range of machines? Then write a script you lazy sod! All these things – Docker, Juju, Snappy and the rest – are just more layers to obfuscate the code that's running on your server, and if you don't know what's running, how are you supposed to fix things?"

He's a bit gruff our angry reviewer, but he does make some good points. While all these are legitimate criticisms, none of them are really about Ubuntu. If you don't like Unity, that's fine, don't use it. There are plenty of other options on Ubuntu. The same goes for Juju and Snappy – they're options for us to use not things being thrust upon users (we would like to see more love for BTRFS though).

When it comes to modern computing devices, Ubuntu officially supports more platforms than any other Linux distro. You can run it on your server, desktop, small-board computer or phone. The only other distro that comes close to this is Arch, which also runs on just about everything, but almost all builds other than x86 and AMD64 are unsupported community versions.

At the time of writing, Ubuntu ships on three different phones with a fourth shipping soon. There's also a tablet device currently on pre-order. Of the three recent Linux-based phone OSes, Ubuntu is gaining the most traction. Firefox OS is no longer being developed for phones (though we may see it again in other devices), Sailfish OS is still under development and a phone is available for purchase, but Jolla (the company behind it) has failed to deliver crowdfunded tablets – a refund is underway – and has had to lay off a large portion of its staff. Ubuntu is still progressing and the long-promised convergence is due to debut in the tablet version. In this, you can connect your device to a monitor, mouse and keyboard and the interface will transform from a touch-based system into the full Unity desktop. Microsoft currently offers a similar experience with Continuum, and Google is developing similar capabilities for Android, but both of these are restricted to running mobile apps and only the display changes to take account of the larger screen, while Ubuntu convergence will enable you to run all the usual Ubuntu applications.

Ubuntu pie

The Raspberry Pi Foundation recommends only two desktop Linuxes for its tiny machines: Raspbian and Ubuntu Mate. And for good reason: the Ubuntu Mate team have done a great job of not just getting their distro to run on the Raspberry Pi, but doing the job well. Unlike Raspbian, the installer prompts you to choose a username and password, so you

> The long-promised convergence will transform a touch-based tablet system into the full Unity desktop

won't get into trouble if you forget to change the default and make the Pi publically addressable (as you can with Raspbian), and the default selection of software is more suited to a home user. Mate also seems to find the right balance of eye candy and computing power for the Raspberry Pi 2's quad-core ARMv7 processor.

Almost all single-board computers come with Ubuntu, including the ODroid, Banana Pi, Orange Pi and Udoo (the only major exception to this is the CHIP, which currently has its own customised version of Debian as the only recommended OS).

There are actually two forms of Ubuntu that run on the Pi. We've talked about Snappy's potential on the server and the desktop earlier, but the area it's moving into first is the Internet of Things (IoT). There's now an Ubuntu spin called Snappy Ubuntu Core, which just contains a very basic OS with Snappy installed on top of it. It's designed for small devices such as the Raspberry Pi or Beaglebone.

From desktops to servers to devices, 16.04 is the most important release of Ubuntu for many years, and it will be the release that, we think, will define Ubuntu long after the next LTS release comes out. In order to stay relevant, this release needs to regain some slipping market share, or at least stop the decline, and it needs to achieve this in an increasingly competitive Linux marketplace. On the desktop, Elementary, Solus, Mint, Arch and others are better than they've ever been. Meanwhile on the desktop, CentOS is now aligned with Red Hat to offer an alternative enterprise-class distro backed by a major company, and CoreOS is pushing cloud computing in new directions.

With 16.04, Ubuntu is rising to these challenges, and at the same time, pushing into new areas. Whether you love or hate Ubuntu, it's great news, because a strong Ubuntu makes for a better world for Ubuntu as a whole.

In the competitive, fast-changing and fickle world of Linux distributions, nothing is certain, but if it can continue this level of progress, this release will mark the beginning of a glorious new period for Ubuntu.

One Linux to rule them all

Traditional desktop PCs and laptops aren't about to disappear, but other computer devices are growing rapidly and account for an ever increasing amount of computer usage. If Linux is going to stay relevant in this new world of phones, tablets and the Internet of Things, then it needs a distribution to champion it. Yes, there's Android, but this is so different from other Linux distros that it's barely relevant.

Ubuntu – more than any other distro – is not just running on other devices but adapting to the way they're used and becoming a great distro far beyond its desktop and server roots. This is good for all Linux users, not just those that use Ubuntu, because it brings more people into the Linux fold, and the more people we have within the community, the easier it is to get hardware manufacturers to support Linux, and the more users there are to test and develop software.

16.04's code name is the Xenial Xerus (apparently a xerus is a bit like a meerkat).

Mobile phones outsell PCs 4 to 1, so even with a small market share, the Ubuntu phone could be huge.


MICROSOFT AND LINUX: WHAT'S GOING ON?

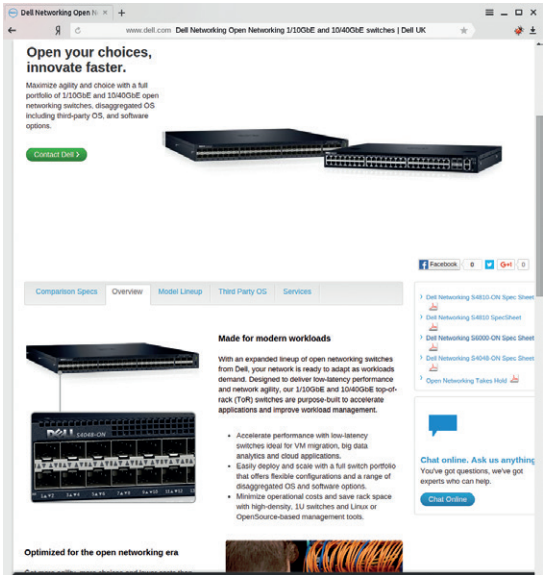

Has Microsoft really made a Linux distribution, and should you **apt-get install sql-server**? **Valentine Sinitsyn** finds out.

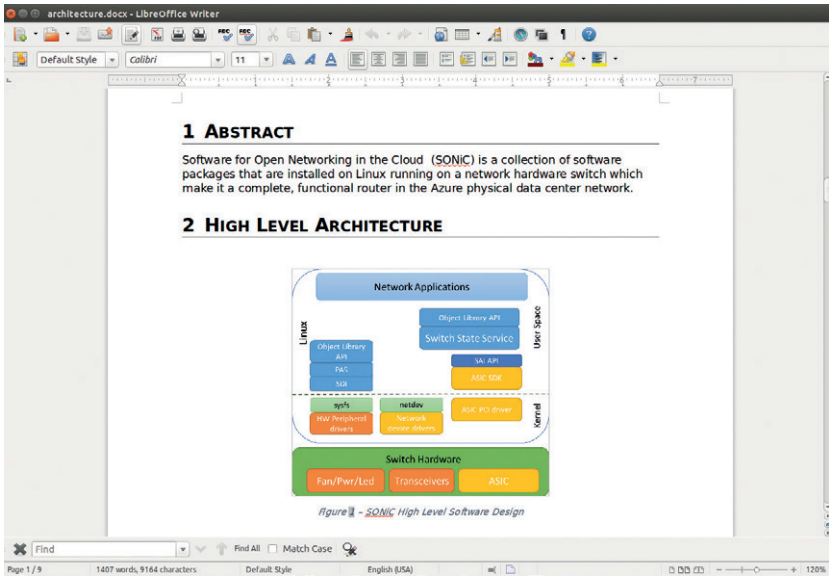
Many human languages provide an idiom saying "never". In English, it's "when pigs fly", my mother tongue has something that literally means "when a crayfish whistles on the hill". Inside the Linux community, we often used "when Microsoft come to Linux", or words to that effect. That happened early spring 2016, and hell is probably frozen now. It generated a lot of buzz, naturally, which made it difficult to sort the wheat from the chaff. We did it and are happy to share our findings with you. First, let us begin with some background.

Many of us have a home wireless router. Often, it runs a specialised embedded Linux distribution, so many people across the globe use Linux without even noticing it. Typically, such routers combine a relatively slow ARM or MIPS CPU with an Ethernet switch and a Wi-Fi access point. Packet switching and routing code (usually the Linux kernel) runs on the CPU alongside other components like a PPPoE client or a DHCP server. You configure these routers manually, and their settings change rarely, if ever.

Inside the data centre

Most home networks are simple, and all-in-one devices work well. Things change drastically at data centre level. Data centre switches are dedicated boxes with dozens of ports operating at impressive 10 or 40 Gbps. Even a powerful x86 CPU won't cope with such





high rates, and they are only made possible with application-specific integrated circuits (ASICs). A typical setup may involve hundreds of switches whose settings are highly dynamic to deliver the performance and reliability that internet users demand.

Now, think of the switch that only runs pre-installed software. To make it work the way you want, you contact the vendor, put in a feature request, and wait. There are no guarantees, though. The OpenFlow protocol, once synonymous with Software Defined Networks (SDN), adds more flexibility. It decouples routing logic (the control plane) from vendor-specific packet switching code (the data plane). This facilitates custom routing algorithms, managed from one central location. But in some cases, you need even more control. Perhaps you want to run existing server-based tools, like Chef or Puppet. Or you don't want features (and associated bugs) that you won't use. Running your own software on the switch is the ultimate answer. If you ever used custom firmware on your home router, you've got the idea.

Today, data centre operators can buy a bare-metal switch and install whatever Network Operating System (NOS) they need. If this NOS is open source, this adds the usual benefits. But it's somewhat impossible to build an open-source NOS if the interfaces to program the ASIC are proprietary. Historically, this was the case, but a few years ago things changed, thanks to Broadcom's OpenNSL and Switch Abstraction Interface (SAI), led by Microsoft. Around the same time, Facebook launched Facebook Open Switching System (FBOSS), which is "a set of applications, not an operating system". HP started an ill-named OpenSwitch project (**www.openswitch.net**) which aims to build a full-fledged community-based NOS (and not to be confused with Open vSwitch). The Open Compute Project (OCP) made the Debian-based Open Network Linux (ONL) available at **http://opennetlinux.org**. For Microsoft, which operates large data centres for the Azure cloud, it was probably only a matter of time to join them.

Above left: Many vendors today produce bare-metal switches that are specifically designed to run an OS of your choice. **Above:** In case you managed to miss the fact that SONiC comes from Microsoft, here is a fat hint: the docs are in OOXML format.

What Microsoft did

Back in September 2015, Microsoft announced and showcased Azure Cloud Switch (ACS). Yet it wasn't available to the public until March 2016, when Microsoft kicked off the SONiC project (**http://azure.github.io/SONiC**). SONiC stands for "Software for Open Networking in the Cloud". According to Microsoft, SONiC is ACS but without the company's internal cloud management applications.

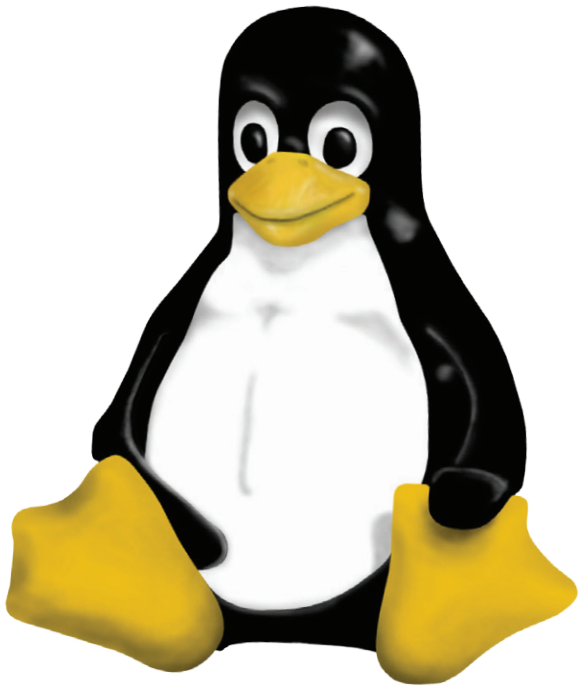
You may now think that Microsoft has released a Linux distribution, perhaps a specialised one. Many news sites suggest the same. That's not true. Like FBOSS, SONiC is a collection of services that run on top of existing Linux system. Currently, it's Debian Jessie. Future versions may use ONL (also Debian-

SAI and OpenNSL in a nutshell

Separating switching logic from the hardware is important, but eventually, both ends need to meet each other. This implies an API to handle port configuration and link state monitoring, fill switching tables and forward selected packets to CPU for analysis.

OpenNSL (**https://github.com/Broadcom-Switch/OpenNSL**) and SAI are this API. OpenNSL provides support for Broadcom switching ASICs. SAI is generic, and most current implementations are really wrappers around vendor-specific APIs. For example, the Broadcom SAI (**https://github.com/Broadcom-Switch/SAI**) builds on OpenNSL, and the Mellanox SAI (**https://github.com/Mellanox/SAI-Implementation**) relies on custom SwitchX APIs.

While the interfaces themselves are open (often under Apache 2.0), their implementations have no such obligation. The situation largely resembles graphics drivers in Linux: OpenGL is an open standard, but it is implemented in a proprietary Nvidia driver. OpenNSL, for instance, provides GPLv2 Linux drivers (Broadcom ASICs are PCI devices) and free (as in speech) header files, but the library itself is generally available in binary form only. Open interfaces are a big step in right direction, but there is still a long way ahead.



based) or even Ubuntu. It doesn't make SONiC a distribution: if you Dockerise some application on top of an Ubuntu base image, this doesn't count as an Ubuntu derivative. So, Microsoft has not released its own Linux flavour (yet). And unlike Facebook, which designed the Wedge switch to run FBOSS, Microsoft's SONiC is purely a software offering. It runs on commodity hardware such as the Dell S6000 or Arista 7050 switches. Yet it's really free, as in speech: the sources (mostly C++ and Python) are released under Apache 2.0 licence.

This being said, some essential SONiC bits were missing at the time of writing, though this doesn't mean they won't be released eventually. Most likely, Microsoft was in a hurry to have something ready for the OCP Summit 2016. The architecture is well documented (not surprisingly, in the DOCX file) and the code is unencumbered and easy to understand.

SONiC is all about building high-level abstractions, so the user can concentrate on network applications logic, not hardware and low-level stuff. To that end, it has a bunch of daemons that listen to various events and maintain switch state in a key-value database (currently, Redis). Those events may come from the Linux kernel (if cable was plugged or IP address added), or other network applications. Say, Microsoft employs the Quagga suite (**www.nongnu.org/quagga**) and BGP protocol for dynamic routing in Azure data centres. Routing table updates are application events which end up in the key-value store. SONiC doesn't currently include an OpenFlow agent, as it isn't deemed necessary. If they have it, the agent would also be the source of events.

Free as in... err, kind of

There is another SONiC component, which subscribes to database updates and applies changes to the switch hardware. This is where SAI comes into play. ASIC is the most important to manage, but not the only one. Switches contain LEDs, fans, power supplies, transceivers and other "platform devices". You want to configure and monitor this equipment, so SONiC should provide tools for that.

As it stands, SONiC is not rocket science, and it won't probably make big news if it weren't from Microsoft. It's relatively small and simple, and tailored for specific requirements that Microsoft has for Azure data centres. But as it is free, things may change quickly if Microsoft builds a vibrant community around the project.

SQL Server on Linux

Another Microsoft crown jewel coming to Linux is *SQL Server 2016*. While this isn't going to happen until mid-2017, you can already apply for the private preview. It should start around the time you read this, but we already know enough to have a good idea of what's going to happen.

In the beginning, *SQL Server* will be available for Ubuntu (with Red Hat likely on the horizon) or as a Docker image. Microsoft embraces Docker at large: it's a first-class citizen in Azure and found in many company's projects, including SONiC and ASP.NET. *SQL Server* on Linux leverages existing open-source offerings in the field, like the FreeTDS protocol driver (**www.freetds.org**). Client libraries are available for many languages, including Python, JavaScript/Node.js, and Ruby, and are free (as in speech). This doesn't mean that *SQL Server* itself would become free, of course.

Why is Microsoft doing this? The reasons are likely pragmatical. Linux is strong in clouds, including Microsoft's own Azure, which we have to keep reminding ourselves is intimately bound with Linux. And it's obviously an opportunity to grab a slice of Oracle's pie, whose proprietary database has run on Linux for many years. With Docker gaining popularity as a deployment tool, people want *SQL Server* Dockerised as well. Finally, this move completes the ASP.NET stack on Linux, expanding its reach and bringing Microsoft new developers.

MEDECINS SANS FRONTIERES
DOCTORS WITHOUT BORDERS

The world's A&E Department

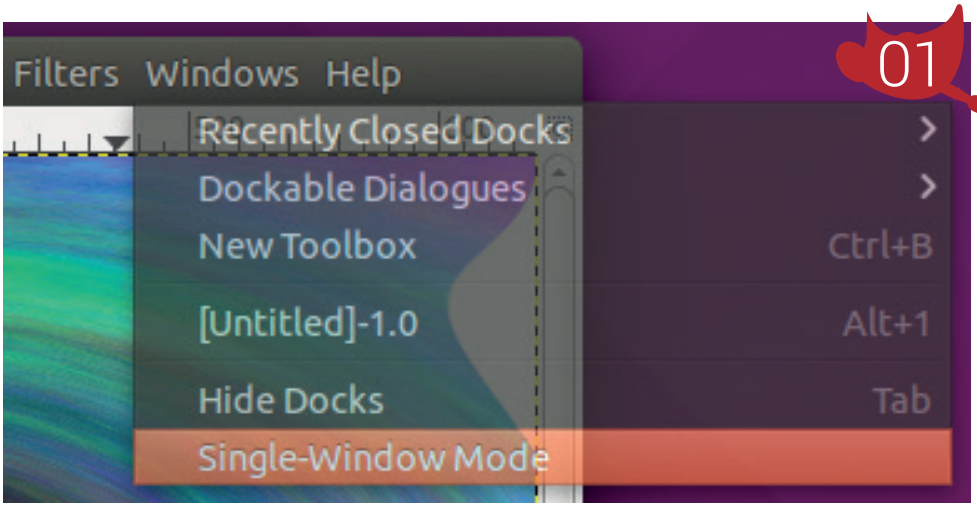
Find out more at msf.org.uk

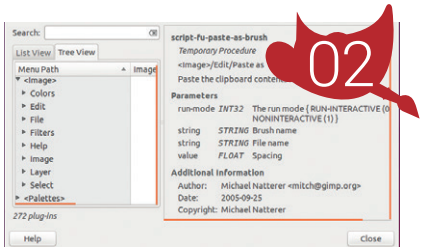
SECRETS OF GIMP

Your pictures will never be the same again after you've gimped them up!

When it comes to free software image editors, there's one name that always comes to mind, partly because it's a great bit of software and partly because it's a wildly inappropriate name: The Gnu Image Manipulation Program (*Gimp*). Since 1995, this has been the go-to application for Free Software lovers who need to make changes to images, however large or small.

Gimp is a complex program. Here at Linux Voice, we think that it's well worth spending a little time becoming familiar with it. The skills we've learned have paid off time and again in tweaking and modifying images, and not just because we run a magazine – personal photos and graphics of all descriptions can be improved with a little knowledge.





01 Configurable user interface

Gimp is suited to a wide range of image manipulation tasks – it's equally adept removing red-eye from photographs as it is creating sci-fi fantasy images from scratch (in the right hands that is), and each use case requires a different set of tools.

Perhaps you want one big window focused on the image you're working on, for example, or perhaps you want to split the display between two monitors with one showing the image and one showing the tools. You can hide, display, rearrange, dock and align almost everything to build just the interface you need without any unnecessary clutter.

02 Plugins

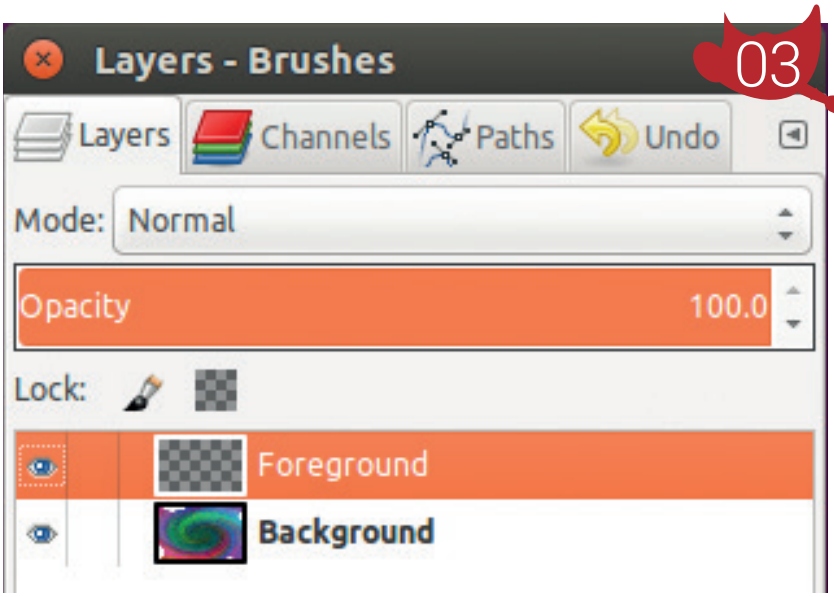
There are a huge number of features in *Gimp*, but no software can provide everything that everyone will need. If you need something that isn't in the default version, you can usually find a plugin. Go to Help > Plugin Browser to see what's available. If you switch to Tree View, you'll see them laid out in the menus that the new features will appear in, so filters are under Filters, edit options are under Edit, and, well... you get the idea.

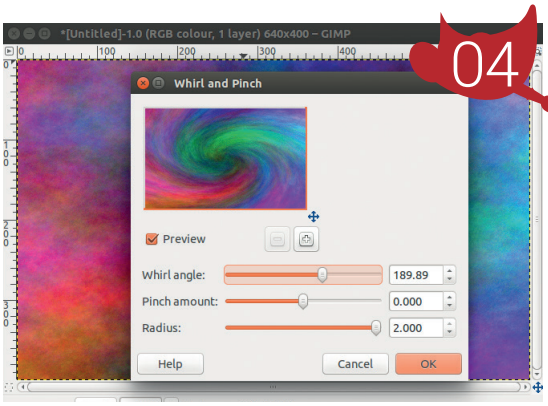
03 Layers

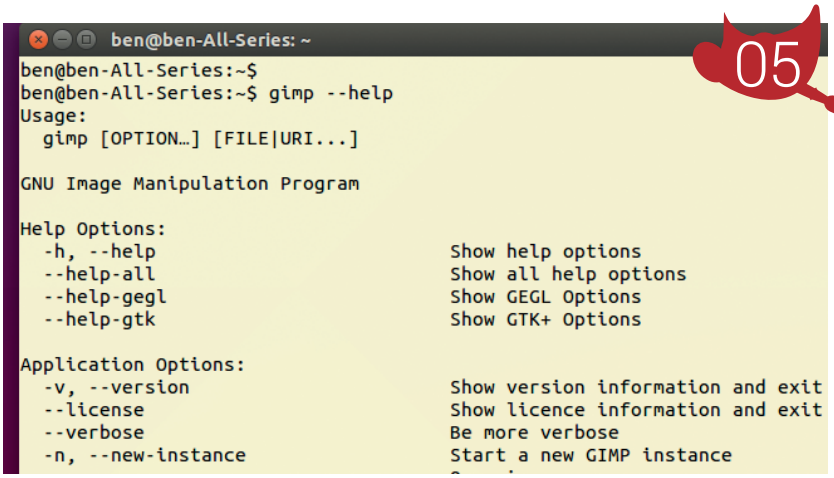
Images can be built up of different components. There might be a background, a foreground, and a series of changes to the foreground. All these components can go in different *Gimp* layers so that you can treat them as individual images when editing, but render them together for the final picture. Layers enable you to make changes to one aspect of an image without altering the whole thing.

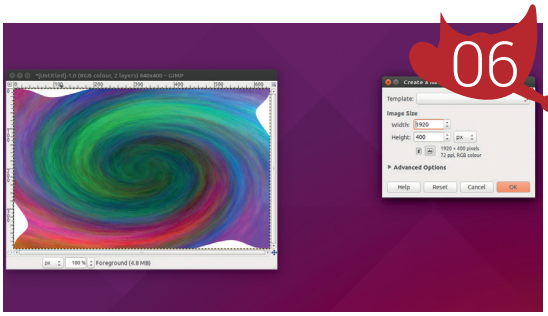
04 Filters

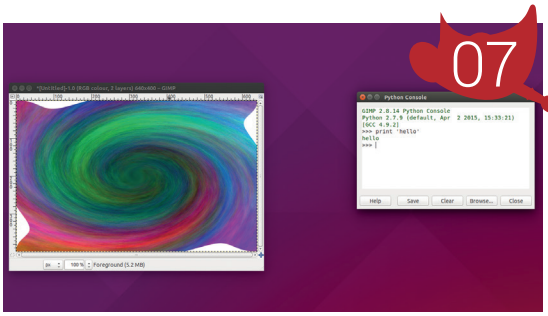
Filters are used to make automatic transformations to an image. For example, the blur and sharpen filters can be used to make an image look less or more in focus. They can be used to generate images as well as modify them, and the Filter > Render

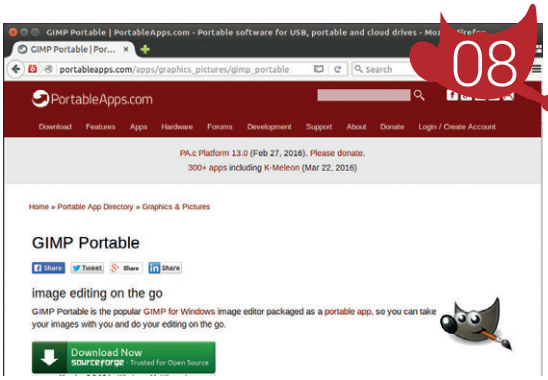












submenu includes options for creating things including clouds and lava.

05 Batch mode

What's the point of a fancy user interface if you're an elite Linux user? Well, actually, there are plenty, but there are also times when it's better to run on the command line. If you launch *Gimp* from the terminal with the **-b** flag, it'll start in batch mode, which runs operations from the command line. Usually, this is in the form of scripts. You can even use blobs to run the same action on a range of files at once. With great power comes great responsibility, so be careful with this and don't convert all your holiday snaps into sepia without a backup.

06 Maths in size boxes

Images are sized in pixels, but you might not always know the exact size you want. For example, you might know that it's going to be displayed at 278x278 pixels, but want the image to be twice this size for display on retina screens. You could work this out and type in the answer to the *Gimp* size box, but there's no need: just enter the size as 267*2 and it'll calculate the correct size for you.

07 Python

If you want to make a very specific set of adjustments to an image, it can be a waste of time to click around with the mouse, and it can be better to write a simple script. *Gimp* covers this with a choice of its own scripting language and Python. There are consoles for both, so you can code interactively from within the main application. Go to Filters > Python-fu > Console to get started.

08 Portability

Gimp is easy to install on Linux, but we don't always have the luxury of having our own Linux machine available. The Windows version can come as a portable app which means that all the necessary files can be stored in a single place, such as on a USB drive. You can download the application from **http://portableapps.com/apps/graphics_pictures/gimp_portable**. Open this in Windows and *Gimp* runs with no installation necessary. If you need image-editing capabilities, you can carry this with you in the knowledge that any PC you use can be turned into an image-editing workstation.

CeBIT 2016

The world's largest computing expo took place in Hanover in March, and Linux Voice was there to check out the Linux and FOSS-related stands.

CeBIT is all about superlatives. It's the world's biggest computer show, held at the world's largest fairground. At the height of the dot-com booth it received a staggering 850,000 visitors – but in recent years the attendance has been more subdued at "just" 330,000.

While CeBIT is very much about besuited business types impressing one another with shiny booths and even shinier business cards, this year's event also played host to various open source projects. Now, given that most of the companies exhibiting at CeBIT are based on proprietary technology and software, does it make sense to have open source there? Or is it a clash of cultures? Well, in our experience it's a positive thing. Attendees can see that open source isn't just the domain of bedroom-dwelling geeks, but actually produces professional and highly regarded software that's used around the world. Many businesses are using and contributing to open source – even if the bulk of their work is proprietary – so it makes sense to have FOSS represented at CeBIT.

> There's a growing trend for companies to take Free Software and add value on top (such as long-term support options) while the core product remains free

In this age of almost constant internet connections, the whole idea of a computer expo where people meet up in the flesh may appear arcane and unnecessary. But sometimes communication is so much more effective when done face-to-face rather than online. If your company is looking to buy some new kit or software, what's better: sitting on the end of a phone waiting to speak to some faceless sales rep, or actually being able to see the product in question being demonstrated and ask questions directly in front of someone?

Money talks

So we went to Hanover to see how the GNU/Linux, Free Software and open source ecosystems are being represented at these big events. And CeBIT certainly didn't fail to disappoint on the wow factor – clearly a lot of money goes in to making shiny booths and even shinier presentations. Sometimes it boggled the mind that companies would invest so much for something that only lasts for one week, but if it results in some major contracts being won, it's worth it...

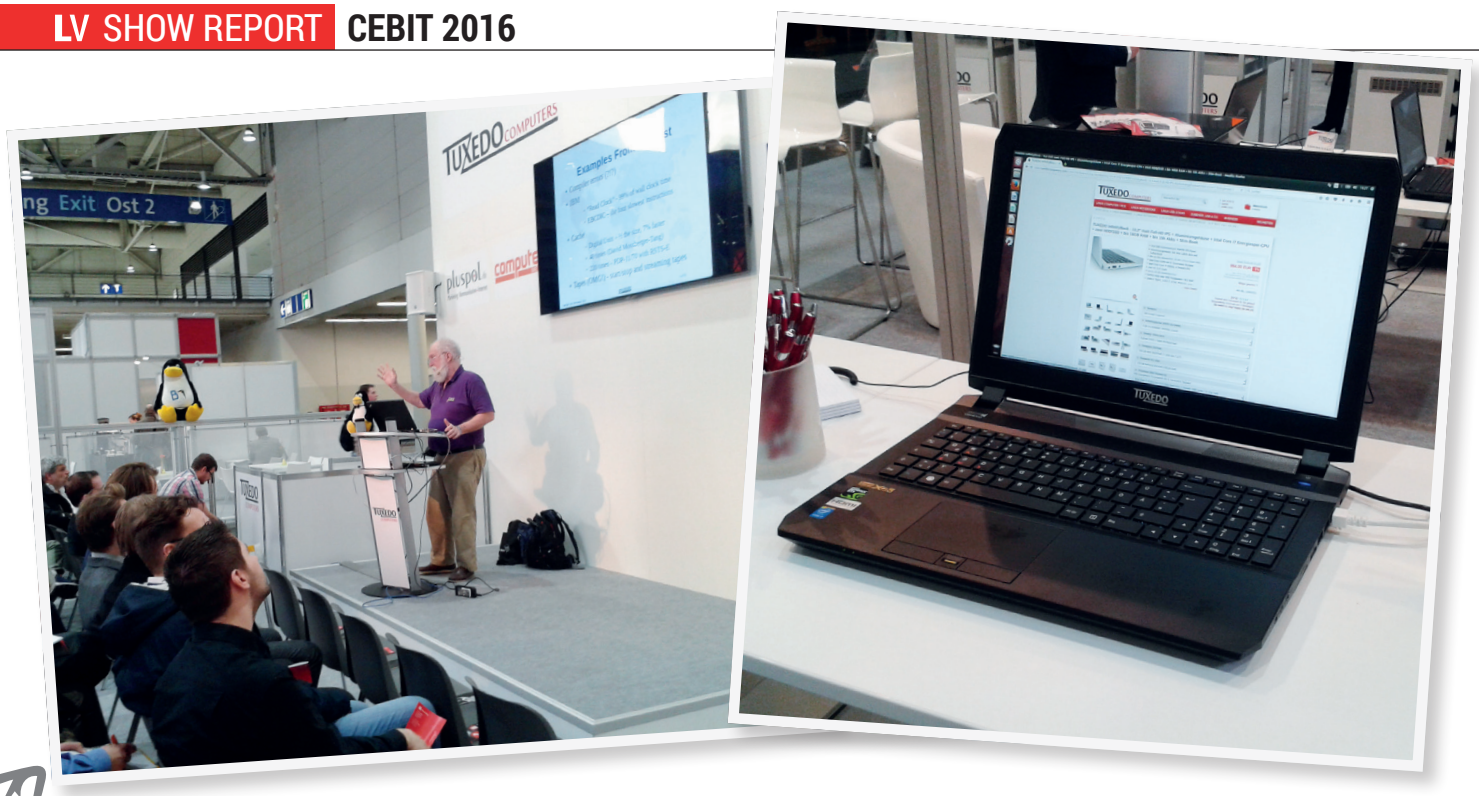
While most space in the gargantuan expo halls was devoted to proprietary products and software, a sizeable area in one hall dedicated itself to FOSS and was known as the "open source park". Many of the companies that had booths there were active primarily in the German market, such as B1 Systems, which offers training, consulting and development services for Linux and other open source projects. Similarly, Arogorum was present with open source document management software.

Some big-name open source projects showed their faces as well. *LibreOffice* had a small team from The Document Foundation along with supporters from the wider community, answering questions, handing out flyers and trying to spread awareness of the project. We were told that many people who visited the stand simply wanted to say thanks for working on the software, while others were still using the (largely dormant) *Apache OpenOffice* version and weren't aware of the mighty strides *LibreOffice* is making.

LibreOffice's stand was part of a larger booth from CIB, a German software and consulting company that has been very active in *LibreOffice* development recently. This reflects a growing trend in the open source world: commercial companies taking FOSS, adding value on top (such as long-term support options or paid-for features) while the core product remains free. Of course, in an ideal world everything would be completely free (as in both beer and speech), but this seems like a healthy balance to us –

The Document Foundation had a small stand answering questions about *LibreOffice* and handing out flyers.

The Open Source Park had plenty of booths devoted to companies that develop and support FOSS projects.



Jon "Maddog" Hall gave an entertaining speech explaining the history of Free Software and why it's important.

FOSS gets better and developers can still put bread on their tables at the end of the day.

Shiny hardware

Other open source projects that had booths at CeBIT included *MariaDB* (the community-developed fork of *MySQL*), while on the hardware front Tuxedo Computers was present with a bunch of laptops to try out. Linux on laptops has always been a bit of a thorny issue – although it has gotten a lot better in recent years as hardware has standardised. But it has always been frustrating to find a decent, well-built laptop that runs Linux flawlessly. There are usually one or two niggling things that don't quite work as expected (eg Wi-Fi or suspend/resume), or you have to download a poorly maintained proprietary blob, which defeats the point of using open source in the first place.

The Bavaria-based Tuxedo Computers had laptops of varying sizes on show, each of which runs Linux out of the box (customers have a choice of Ubuntu, Kubuntu, Xubuntu or OpenSUSE). Each model is supplied with a two-year guarantee, and although the company is based in Germany and its website (**www.tuxedocomputers.com**) is currently only available in the German language, it's possible to order laptops with international keyboard layouts. We will try to get some laptops in to review in Linux Voice – so watch this space.

Along with the stands and booths there were a number of open source-related presentations given in front of seated audiences. Jon "Maddog" Hall, the Executive Director of Linux International, gave a speech explaining why Free Software is important, how it's developed and where it's going – nothing new to long-term GNU/Linux fans, but useful for CeBIT

Sadly, B1 Systems wasn't selling these large plush penguin toys. They could've made a fortune, we reckon…

visitors who were still wary or even sceptical about FOSS. Maddog did a good job of explaining both the philosophical and practical benefits of Free Software.

Then there were some more technical talks, such as the one from Thorsten Behrens, a prolific *LibreOffice* developer. Behrens outlined some of the design changes being made in the suite, while Italo Vignoli from The Document Foundation gave a talk about migrations to *LibreOffice* in Italy – focusing especially on the bumper 150,000 PC migration of the Italian Ministry of Defence.

Ein Weißbier, bitte

There was plenty of fun to be had at the end of each day as well. An Oktoberfest-esque beer hall was set up for plenty of boozing fun (certainly appreciated by many visitors from the Far East who didn't have time to travel down to Munich), and the centre of Hanover was fairly easy to reach with a 20-minute tram ride.

So on the whole, Linux and Free Software was represented pretty well at CeBIT, even though it wasn't the main attraction. It would have been nice to see more Linux distributions present, handing out DVDs so that visitors could try the operating system on their own machines, but we were told that the booth prices are extremely expensive. But it was good to meet some familiar faces from FOSS projects and hopefully the talks from Maddog and co managed to convince some attendees that open source works, it's beneficial to all, and it's here to stay.

This has nothing to do with Linux, but we thought this assistant robot was rather cool.

Would we recommend visiting CeBIT next year to Linux Voice readers? If you're Stateside or in Australia then no – it's a long way to travel for an event where FOSS only plays a small part. But if you can reach Hanover within a few hours, keep an eye on **www.cebit.de** later in the year for news about the 2017 event. If it looks like a lot of FOSS projects and related companies will be present, it's worth going along – even just to be wowed by the sheer size of all the halls. And who knows, maybe there will be even more robots next year... LV

FAQ

Servo

The new browser engine from Mozilla built for security and speed.

BEN EVERARD

Q Actually, I know this. A servo is kind of like a motor except it enables you to rotate to a particular point. I remember them from your walking robot tutorial in issue 18

A Well, yes, those are servos, but the Servo (note the capital S) we're talking about today is Mozilla's new browser framework that's going to bring parallelisation to web browsing.

Q Wait, bring parallelisation? This is 2016 and I've had a multicore processor for the last decade. Are you telling me that I've been only using one core for all that bleedin' time?

A If you've been using *Firefox* then yes, you've only been using a single core for your web browsing, wasting all that processing power. *Chrome* (and *Chromium*) are multi-threaded and balance multiple tabs across many cores of a CPU. *Firefox*, on the other hand, runs everything on a single CPU core.

Q Everything?

A Well, almost everything. Some plugins – such as Flash – run in separate processes, but everything else runs on just a single core. This can lead to poor performance when you've got a lot of tabs open.

How bad the performance is depends a lot on your CPU: if you've got a powerful desktop CPU, then it can probably open quite a few tabs without a significant slowdown; however, if you've got a slower CPU then the slowdown can happen quite quickly. Servo is being developed by Mozilla with support from Samsung with a view to it working on Android and Arm processors. These less powerful processors stand to gain more by being able to split the load across cores.

Q *Chrome* splits up the processing load by running each tab in a different thread. Does Servo break down the task in the same way?

A No. Servo breaks down the processing load in a more fine-grained way. Individual tasks within each page are handled separately. For example, HTML parsing, image decoding and layout can all be handled by different threads.

> Individual tasks within each page are handled separately... HTML parsing, image decoding and layout can all be handled by different threads

Q But isn't it also a good idea to break down the different tabs into different processes?

A Yes it is, and Mozilla has a separate project for this called Electrolysis. You can try out Electrolysis if you're using one of the developer versions of *Firefox* (either Nightly or Aurora). Open Preferences and check the "Enable Multi-Process" checkbox, then restart your browser.

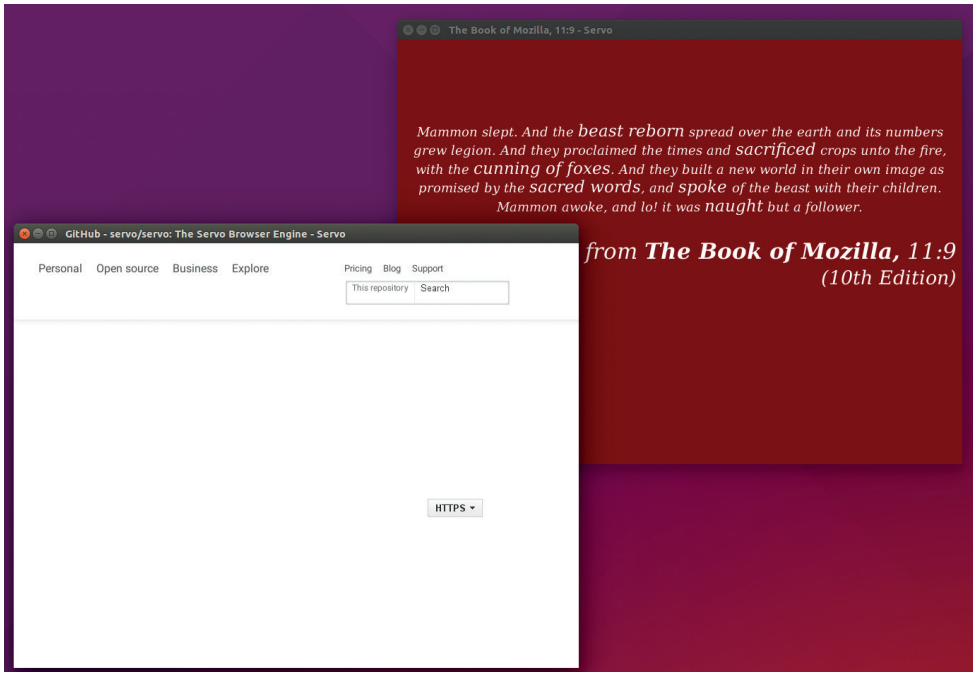
Servo is being built with support for Electrolysis, so as well as the performance improvements in Servo, you'll get these benefits as well.

Q You've talked quite a lot about potential speedups, but not mentioned just how much quicker it is. What sort of performance can you get out of Servo?

A This is quite a hard question to answer. Most browser benchmarking focuses on JavaScript

performance, but this isn't the main area that Servo will speed up. It should speed up the entire process of rendering a website. A more significant problem is that we found that the current version of Servo struggled to render many complex websites correctly, which means that the sort of situation that we would expect to see Servo perform particularly well in, we can't yet try.

When splitting load across more than one core, performance rarely improves linearly. In other words, if you split a task across two cores, it doesn't usually run twice as quickly, because there are overheads in coordinating tasks between the different threads. The actual speedup depends a lot on the particular task and the design of the application. If we were a gambling magazine, we'd bet on a 2–3 times speedup across a four-core CPU, but this really is just speculation.



Servo can display its own test page, but can't manage the project's GitHub site.

Q Other than improved performance, are there any other reasons to use Servo?

A Speed is important for web browsers, and in order to be quick, browsers tend to be written in quite low-level languages that give you a lot of control over what's running on your machine, and especially, how memory is handled. *Firefox*, for example, is mostly written in C++. The problem with low-level languages is that they tend to be prone to security problems, particularly around memory management. Problems like buffer overflows can enable attackers to run malicious code and missing bounds checks that can lead to information leaks.

Mozilla developed the Rust language specifically to solve the problem that browsers have in that they need to be both fast and secure. Without wanting to be too technical, Rust enforces rules around memory ownership that prevent many security problems while at the same time don't compromise significantly on the amount of control a programmer has. These same rules also help multithreaded performance because they ease the way data is shared between different threads.

Servo is written in Rust, so in principal, Servo should be more secure. However, *Firefox* is mature and is has been heavily tested in the crucible that is the open web. Many security bugs have been found and fixed and now *Firefox* is very secure. Even with the additional security that comes through using Rust, it's unlikely that Servo will be as secure as *Firefox* initially, because security bugs can be subtle and take time to find. In theory, however, Servo should eventually mature into a browser that's more secure than *Firefox*.

Q So far it's been all good news about Servo. I get the feeling that there's some bad news you've been avoiding telling me?

A With all the potential changes coming with Electrolysis and Servo, Mozilla has taken the drastic step of changing the way add-ons work, and is in the process of bringing them into line with *Chrome* and *Opera* by using the WebExtensions API. This interface gives the developer less power than the XUL approach used previously, and the decision to change has met with ire from some within the add-ons community.

An easy way to tell if the extensions on which you rely will be affected is to see if there's currently an add-on for *Chrome* that does the same thing. If there is, then there'll almost certainly continue to be a *Firefox* add-on to do the job. If not, you could be out of luck once the change happens.

This doesn't just affect Servo, but it's coming to all versions of *Firefox*

Q Hmm, I can cope with that. I think it's time for me to switch over and get started.

A Hold your horses – Servo isn't even in Alpha testing yet. Or, at least it's not as we're discussing this. By the time you read this it might be, and the first testing versions are expected in June 2016.

We're a bit skeptical of the release date, but you can compile the development from source even before it reaches Alpha. There are detailed instructions for all major platforms on the project's GitHub page: **https://github.com/servo/servo**. You'll need a couple of gigabytes of disk space (the exact amount changes from day to day as it is being constantly updated).

Rust, the language of Servo, only reached the first stable version in May 2015, so the whole technology stack down to the code is very young, and there's plenty of possibilities for road-bumps along the way. Work on Servo is being undertaken as an experiment to see if it works, and only if it does prove to be superior will it be considered as a replacement for the *Firefox* rendering engine. Therefore, there's no planned release date for a stable browser based on Servo.

R0ML

AKA ROBERT M LEFKOWITZ

There are programmers of the mundane and then there are astral philosophers. Graham Morrison finds an old-school hacker who inhabits both spheres.

From linear algebra and nuclear physics to Wall Street and Haskell, all via the Massachusetts Institute of Technology. Robert M Lefkowitz is a programmer who has been working with computers since the 1970s. He's a proponent of both open source and new development methodologies, mostly from the unique perspective of working within proprietary companies.

He's also one of our favourite people. He's got some brilliant ideas about how programming fits into a grander literary landscape, and how we seem to be entering an age of immutability. The only problem he had was getting him to admit whether this was a good or a bad thing...

> The important bit isn't the explanation but the equation – once you've read the explanation you can throw it away

LV You've spoken before about how you think technology should make this the fourth age of publishing, where readers are also the publishers. Do you think this is still happening?

r0ml: My thinking changed when I read Deborah Brandt [professor emerita of English at the University of Wisconsin-Madison]. Her research area is around the distinctions between reading and writing, the history of literacy, but differentiating between reading for specific things and writing for specific things. We often conflate the two but in fact, reading and writing often are different and many of the changes associated with the literacy landscape are related to the change in the balance between reading and writing that has happened recently and that is related to what is called "The rise of writing."

Strangely enough, that intersects with my interest in Haskell. But the original conceit that I was working along was not only the idea of reading and then reading versus writing, which threw a monkey wrench into it, but also the idea of 'literacy.' There is a strong current going back to [Gerald Jay] Sussman and [Donald] Knuth that writing software is a literary activity, and certainly in the open source community we encourage that specifically because we use copyright on the Free Software side to be the instrument of defence, if you will. It's the weapon of choice in staking out the landscape, and we see things through this copyright filter. We can boil it down to 'Patents Bad', 'Copyright Good.' And we worry about copyrighting software licences, and there's a lot of interest and activity around copyright as it pertains to software, which only makes sense if we view software as a literary, or expressive, activity.

LV Yet copyright is relatively unfit for purpose, and the people who need to change it are those who benefit most from it's current state.

r0ml: But what if – and this is one of my interests in Haskell – Haskell is not a 'literary programming language.' It is a mathematical programming language, and so the sensibilities are completely different. In every programming text, in chapter one there is a section that goes into the importance of naming things properly, that you need to name your functions and your variables for legibility, so that it's understandable.

In real-world Haskell, there's a line where they say, "Always choose extremely short names as it enhances readability." And I thought, "Wow! That's an interesting take on that." And the reason for that is that if you say it's not a literary activity, it's a mathematical activity, when you do equations, you don't want descriptive words: it's not the words that are important, it's the structure that's important. It's thing 1 and then thing 2. You want to be able to manipulate the structure to create equivalences. So you can say, "If we take this thing, and then we change it in this way, we have this thing which we have proven is exactly the same as that thing." If you approach programming in that way (and Haskellers tend to approach it that way) it's manipulating symbols where the symbols do not necessarily have to have any relationship to reality, so you can be manipulating abstract symbols. There was a British study where they were looking at this notion of whether everybody can be taught how to program. What they did was, for people how had taken a programming class, they gave them a test before they did anything, and then gave them a test afterwards, and then based on the

> Programming is not a literary activity, it's a mathematical activity

The essence of open source: "I'm working on stuff, what you've done doesn't do exactly what I want but rather than patching and changing yours, I'll just copy the bits I need and bring them over to mine and keep going."


Should everyone be a programmer? No!

people who had learned stuff – was there anything about the test they had given first to predict who was going to learn and who wasn't before they took the class. The test was of the form, "A is assigned 3. B is assigned 2. B is assigned to A. What's the value of B."

Half the people don't get that right. It may be obvious, but there's a huge body of people who don't find it obvious. The conclusion was that there are people who want it to make it sense. They want something to work a certain way and they build a mental model that's associated with the things they're familiar with, and in the absence of that mental model, they're guessing.

That's what people do!

r0ml: Right! So Haskell is for people who don't care – B doesn't have to stand for anything. A doesn't have to stand for anything. But you re-arrange the symbols and you move the thing inside the A to the thing inside the B and what's in B and A, I don't know, but it doesn't matter because it's a game where you're rearranging symbols.

That's what programming is, and it's not a literary activity, it's a mathematical activity. And then – copyright doesn't apply! You can not copyright a different expression of an equation, since you can show mathematically that they're all equivalent. I

But that's what we've all thought for a long time.

r0ml: I would say that there are two different sensibilities. You read the introduction to *Structure and Interpretation of Computer Programs* [Gerald Jay Sussman, 1st ed. 1985], or Knuth's work, and they start out by saying that programming is not about talking to the machine: programming is about explaining your algorithm to other humans so that they can understand it. So there's this literacy ethos, and in fact, Knuth's book was literate programming.

How do you write things so that they're publishable and understandable as texts? The mathematical [way] says, "It can be extremely dense and bizarre Greek symbols laid out on the page in some way that then needs some explanation so you can figure it out." But nevertheless, the important bit isn't the explanation but the equation, so that once you've read the explanation you can throw it away, because you have it now, and you can see it in the symbols. If the code is the symbols and the mathematical explanations, copyright doesn't apply. What does that mean for our concept of open source?

In the absence of needing to license copyrights, what's the difference between open source and not open source, and how do you differentiate between the two? There's a cultural thing about how we want it to be literary, but what if it's mathematical?

We wouldn't think there was a single answer to this. Not everyone who understands Python needs to understand Calculus. But there is a profound difference in the procedural approach of Python and the functional approach of Haskell.

r0ml: You're right. It's not necessarily that there's a right and a wrong answer – that it's all one or all the other. But then if you look at it in the literacy vein, we do say "Everybody should be literate and everyone should learn arithmetic." There's some level of mathematics that everybody needs to learn and there's some level of mastery of their native language that everybody needs to learn and both of these are good.

R0ml switched to *Vim* after 32 years of using *Emacs*. His fingers must hate him.

Maybe that's sufficiently similar that we can take it from there, but what does that imply for a long-range view? Does it mean there are different kinds of programming languages or programming languages about maths?

There's a programme that the United Nations is trying to achieve called Universal Literacy – do we have a programme to achieve universal numeracy and how do we define numeracy? Literacy is more binary – you can or you can't. Numeracy is a little fuzzier because people can count on their fingers, but is that enough? Or should people be able to add and subtract three-digit numbers. Where do you draw the line?

LV This is a similar question to the one we asked Tim O'Reilly – should everyone be able to code? He said no. But he also said everyone should be able to change the way their computers worked.

r0ml: In order to find the middle road, I think we need to understand whether we're trying to find a road that's more like literature or a road that's more like maths, and if it's more like maths then possibly we're going about it the wrong way, because a lot of the easy-to-learn languages are less mathematically rigorous and more, I would say, literary oriented. You have to learn what the words mean and what the libraries means and it's about understanding a large vocabulary of built-in libraries as opposed to understanding how to manipulate symbols.

LV Why do you think functional languages have become so popular recently?

r0ml: In my nth age of computing idea, I think the underlying theme that is taking hold in computing across the spectrum is immutability. That once it is writ, the hand moves on! I would struggle with the idea of 'Big Data', and my new definition of big data is immutable data – it's data that you never update values, you only append as new information comes in. So all of the three or four Vs of big data (Volume, Velocity, Variety and Variability) had nothing to do with bigness. In fact, you'd see 1.5MB data sets that they would argue were big data, so I think the attribute that is the essence of big data is 'append only. "Once it is writ, the hand moves on."

Functional languages are all about no variables. Once you have bound a name the hand has writ and you move on. You can create a new one but you can never change the old one. Docker, the containerisation, it's all about creating this environment that you can deploy as an immutable thing without making changes to it. And git, in some sense, is that as well. You never change stuff; you always apply the patch and branch again so you start from nothing, and it's all a sequence of patches that are appended to the nothing until you arrive at the current state, managing the path that you have to take to arrive at your destination, although there might be a little conflict resolution. All of these are all about immutability.

LV You said you were joking earlier, but could this a threat to the future of open source?

r0ml: I believe the way I phrased it was, "It's a threat to our current conception of open source." My interest in open source, since the beginning, was the educational value, and I subscribe to those literary interpretations of software. If I were attempting to solve a problem and other people had written software to do that, I could read that software and I could learn from that. I believe the way we teach it is – you have a blank piece of paper, write me a program that does X. Starting from there you never have to read old code, you just have to understand how the

> My interest in open source, since the beginning, was the educational value

symbols manipulate, you make your program, you run it against the test stuff, it works – you're a programmer. Nobody does that anymore. What you really have is vast quantities of software, and you have to navigate it and figure out where [to change it].

We have this big mass of software and we have to go there and change it because it was immature and everything was growing quickly. So another thing that functional

programming tries to encourage is 'one-liners,' functional expressions that can be composed. That way I don't have to understand all of those bits, I just have to say, "I want this bit, and this bit, and this bit, and I'll make my thing which is a different composition of those things that solves my problem."

As we move to the idea of whether it's data or code, the way we make new things is by composing these bits and we in general have a distaste for modifying things, it's all about making another thing atop that previous thing. Our concept of open source – which is people contributing patches, and working to modify a codebase over time – it instead becomes this conception of things that have been written and once writ the hand moves on. If you never have to look inside, the source has no value. So you wouldn't need to do that in order to build software, although you might need to do that in order to understand software.

"The sense of the age is immutability, and functional programming fits into that whole paradigm."

LV Does that mean a severance with the old political ideology behind Free Software?

r0ml: The political ideology might shift. If it's software that is deciding when to brake or swerve in your self-driving car, does it need to be publicly posted so that people can evaluate it and so the code is available for court cases? Is it a legal requirement because of the way it intersects with reality? You could argue for public health reasons, or for voting machines – to have it be closed is like having a closed society – going into a room and picking the next leader, kind of thing. If it's going to be open it needs to be open and possibly the mechanisms need to be auditable.

LV Do you think there are any new freedoms that come with modern development?

r0ml: Security and privacy are rising concerns, and how they intersect with this immutability notion, but this immutability intersects with the right to be forgotten. At a technical level, we're exploiting immutability more and more, and that becomes the threat. In privacy and security terms, things become immutable and can't be forgotten and can't be changed – there's a good side to that and a dark side to that. We're giddy with the prospects of the good side, and the dark side is slowly revealing itself.

R0ml is a guru. Just out of shot were gathered multitudes come to listen and learn.

LV But there's nothing unnatural for the current generation when they think about Facebook's immutability and their online lives living forever on the internet.

r0ml: But if there are dark consequences that are not clear yet, in the long term. As a young child I remember my mother taught the history class... One of the stories she told (which Wikipedia disagrees with, but I grew up with) was this notion that what caused the downfall of the Roman Empire was plumbing. They developed plumbing and they brought water to the houses of the rich and powerful, and it's called plumbing because it's made from plumbum – lead – so this poisoned the upper classes and led to the fall of the Roman Empire. So is plumbing a good thing or a bad thing? They had some activity that they felt was benign, or even good, which turned out to be the thing that caused the downfall of civilisation.

REVIEWS

The latest software and hardware, rigorously bashed against a wall by our crack team.

Andrew Gregory
Is afraid of frosts in April and so is late getting his potatoes in the ground.

The Linux desktop has changed beyond recognition over the last five years, when Gnome 3 was released in April 2011. There are swishy panels, full-screen by default, and new ways of finding applications that involve something more innovative than using a boring old menu. But have humans changed that much over the last five years? Of course not. We all put so much effort into doing whatever else we need to do that learning a new interface is always at the bottom of our priority list.

Back to the future

Hats off then to the Mate team for delivering such a superb desktop. Ignoring trends and looking extremely like a bunch of luddites at first, they stuck to a system that was tried and trusted and have made it better slowly, gradually, at a pace that our tiny human brains can handle. That's Linux for human beings, and it's why Ubuntu Mate makes so much sense. Not all change is good – remember that the next time you see someone praising modernisation for its own sake.

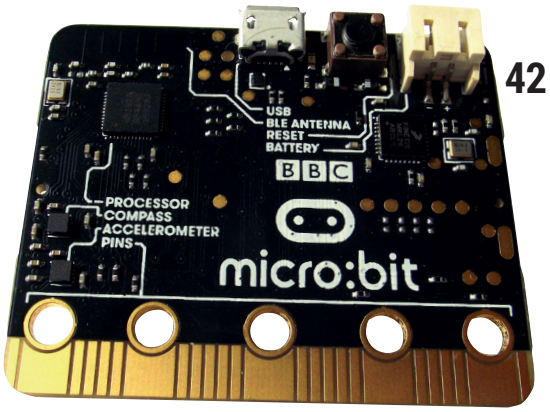
andrew@linuxvoice.com

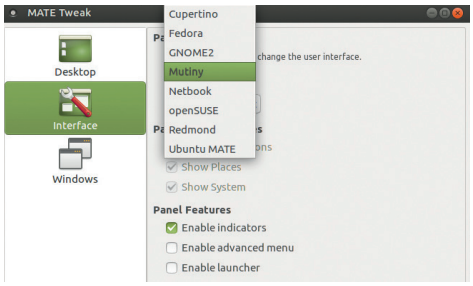
On test this issue . . .

BBC Micro:bit

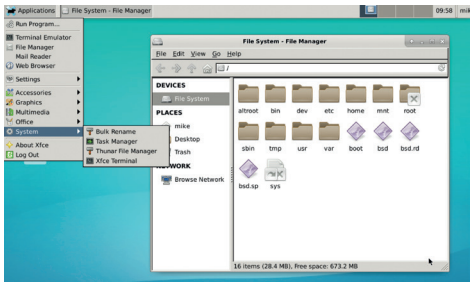
42

At last, the BBC is using some of its vast wealth to foster computing in schools. And by the looks of it, this little device may well end up in a few sheds, garages and workshops too.



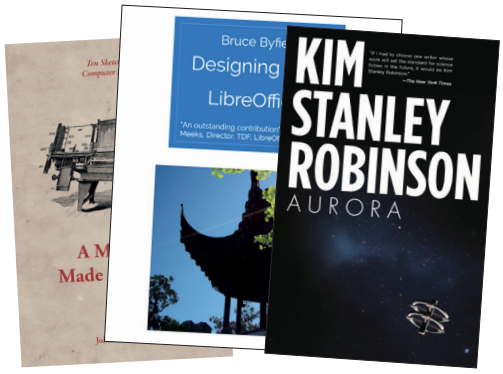


Ubuntu Mate 16.04 **44**
Take Ubuntu, add the user-friendly Mate desktop. Marriage made in heaven or unholy alliance?



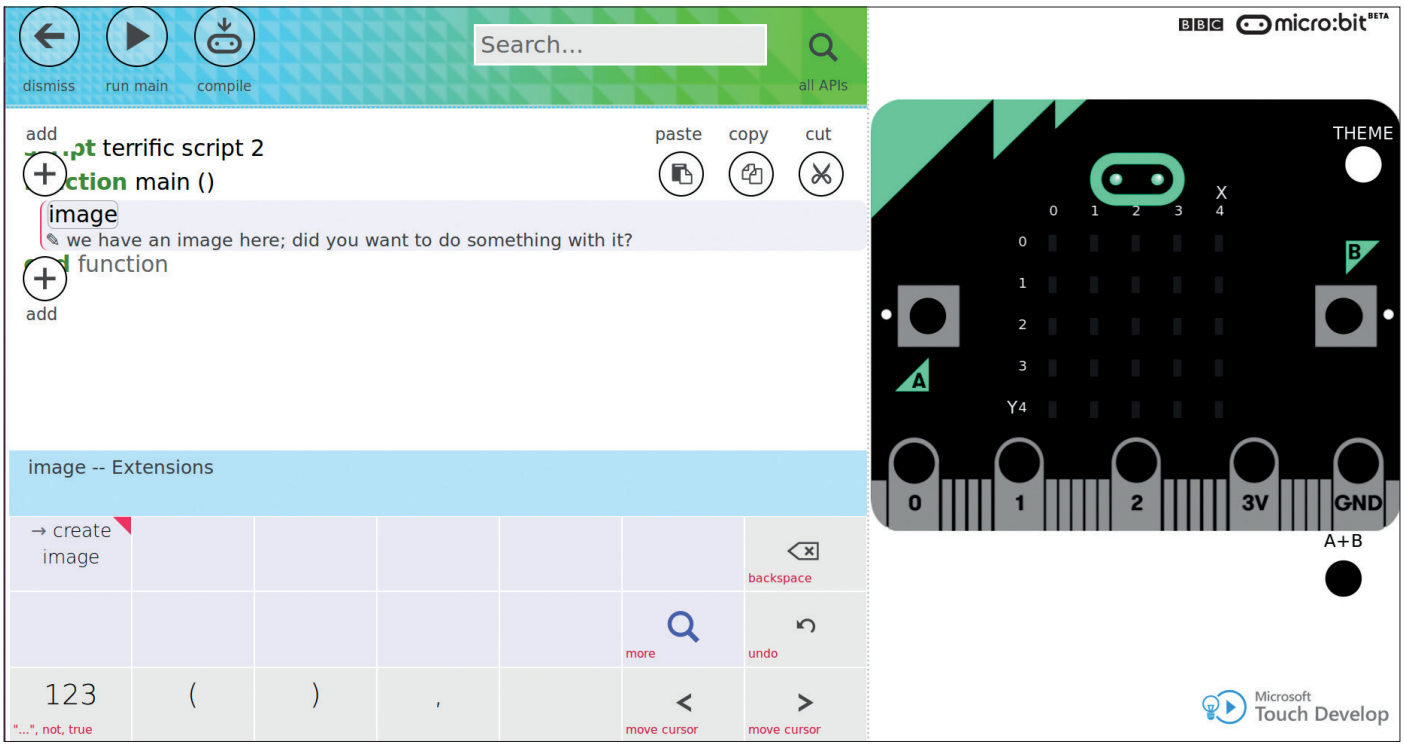
OpenBSD 5.9 **45**
The latest desktop offering from the non-handholdy, ultra-secure Unix flavour, OpenBSD.

Group test and books



Booooooooooooooks!!!! **48**
Juxtapose the vast bleakness of space with the claustrophobia of living on board ship, LibreOffice and computer science in our papery trio.

Group test – Instant messaging clients **50**
Low bandwidth, no-fuss communication ideal for working from home/the pub. No wonder we love instant messaging (and you should too).



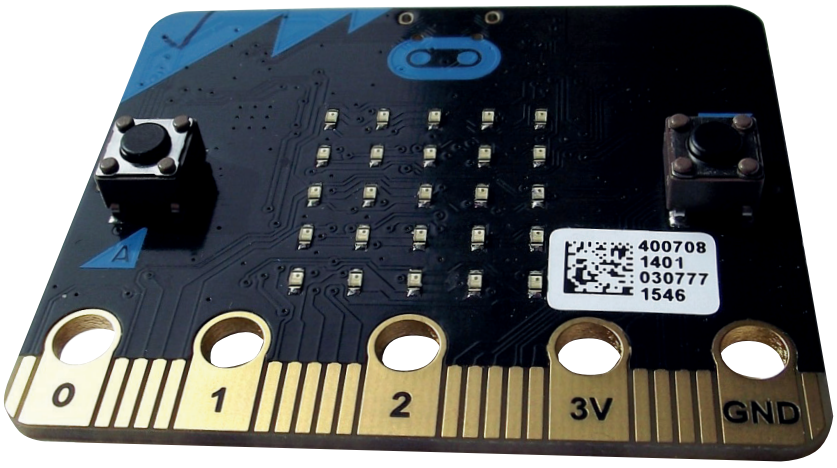
BBC Microbit

Ben Everard investigates the first BBC computing device in 22 years.

Website www.microbit.co.uk
Developer BBC
Price Free to year 7 school children

The Microbit is a microcontroller developed by the BBC to help children learn to program. It's a bare circuit board about half the size of a credit card, with an array of buttons, LEDs and output connectors (three programmable, a 3-volt and a ground). Flip it over and there's a micro USB connector, reset button and a battery connector.

The board is powered by a small ARM Cortex M0 processor, which is designed for embedded applications and doesn't have the features needed to run a full operating system such as Linux or Windows. There are no ports to plug in your mouse, keyboard or monitor, so you need to write your code on a different computer and upload the compiler output to the Microbit.

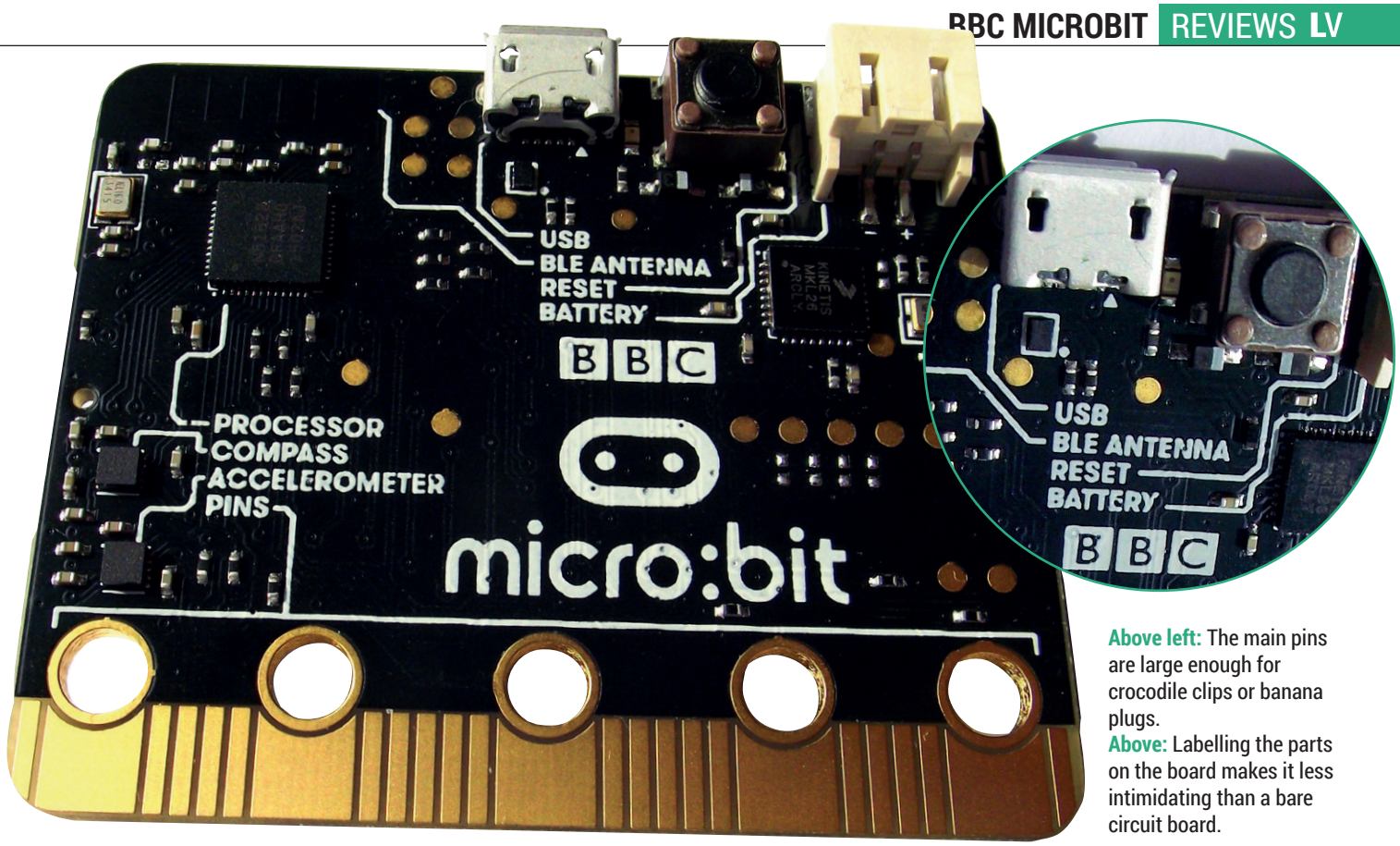


The layout of the Microbit is well suited to making simple games.

There are four officially supported languages: Code Kingdom's JavaScript; Microsoft Block; Microsoft Touch Develop; and Python. They all run in the browser through **microbit.co.uk**, so there's nothing to install or set up. On that website, you can write code, compile it, and download the result as a HEX file. Of the four languages, only Python is traditional in the sense that you write code in text. In the Block language, you drag and drop bits of code together and you can create programs without ever needing to use the keyboard. Code Kingdom's JavaScript is about halfway between this graphical approach and writing code, because although you drag and drop blocks, these blocks contain real JavaScript. Touch Develop has an on-screen keyboard that changes depending on what it expects you to type next. If the cursor is next to a variable name, for example, the keyboard will display all the options that could go next such as +, =, > , etc.

But there's no Basic!

Our inner geek is a little disappointed that BBC Basic (the language of the much-loved BBC Micro used by most British school children in the 80s and early 90s) isn't available for the Microbit, but it's much better for today's programmers to learn with modern languages rather than relics from the past. The four languages do complement each other well, and between them, they give a good range of options for all levels of programmer. We're a little concerned that half of the

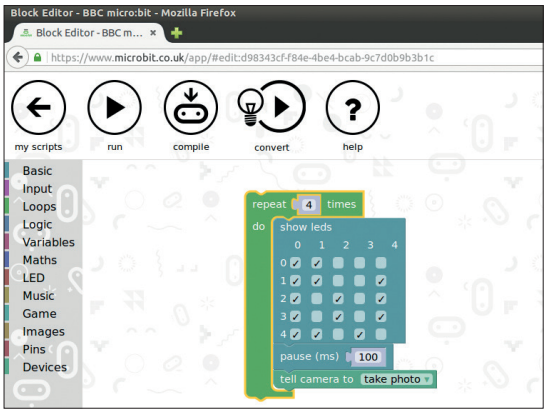


Above left: The main pins are large enough for crocodile clips or banana plugs.
Above: Labelling the parts on the board makes it less intimidating than a bare circuit board.

recommended programming tools come from Microsoft, but the languages are open source and available through Microsoft's GitHub account.

The initial setup of the hardware is just as easy as the software: you just plug the board into your machine via USB and it'll appear as a USB storage device. Just copy any compiled code (a HEX file) into the mounted storage, and the Microbit will run your program. There's nothing to install, and it works on any modern OS. As well as USB, you can upload programs via Bluetooth from phones and tablets using the Microbit app from Samsung (available for Android version 4.4 and up and iOS version 8 and up).

The HTML interface that enables the Microbit it to work on most computers also runs on phones and tablets, but performance is poor. The development environments struggle to run on modest hardware, and don't work particularly well with touch input.



The Block language enables children to code without having to remember syntax.

We'd recommend sticking with the desktop for development if at all possible.

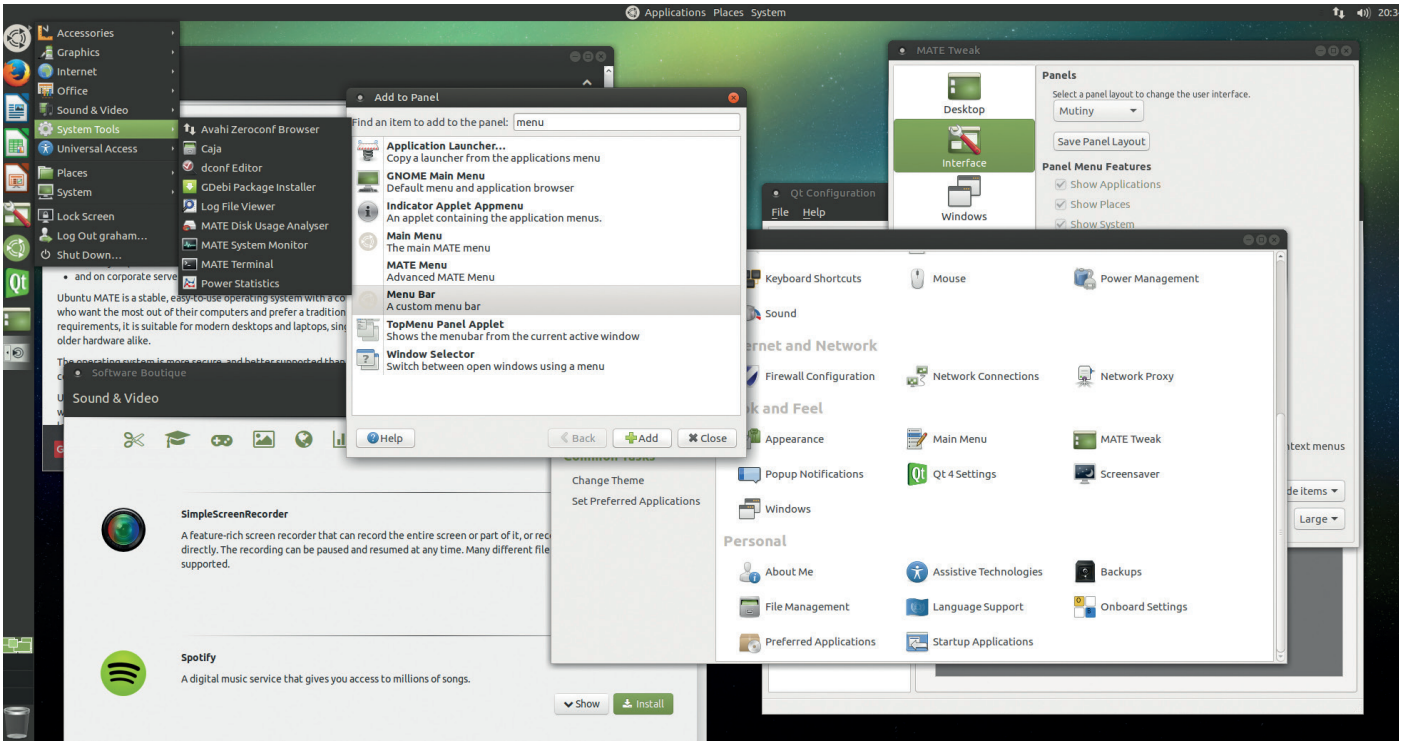
While the software setup is excellent and the hardware works well, the board is limited. With just two buttons and a five by five grid of LEDs, an interested child could quickly run out of things to play with. There are also options using the compass and accelerometer, though these are still held back by the lack of output hardware. Along the bottom of the Microbit, there are an additional 16 GPIO pins that could be used in more complex projects, but in order to access them, you need additional hardware. There will, no doubt, soon be a healthy market for add-on boards to bring more features.

> It's much better for today's programmers to learn with modern languages rather than relics from the past

The Microbit doesn't feel like a board to teach a few students a lot about computing – it feels like a board to teach a lot of students a bit about computing. It's an introductory device that many students will outgrow, and that's fine. There are loads of platforms that students can use to grow the skills they first learn on the Microbit. Getting children interested is the hardest part of teaching anything, and the Microbit is an exciting device for kids to play with.

Really easy to use and great way to get children interested in computing.

★★★★★



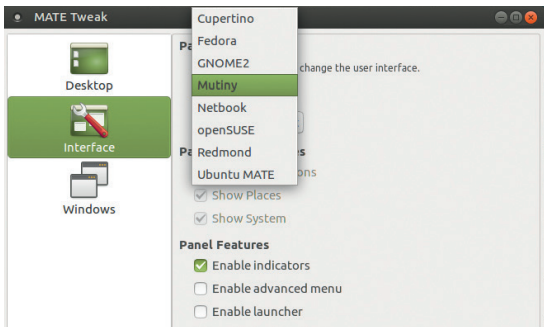
Ubuntu Mate 16.04

With more features than a Yamaha DX7, **Graham Morrison** has got this one covered.

Web https://ubuntu-mate.org
Developers Martin Wimpress and an awesome team.
Licence Various open source

We know this is a rather Ubuntu-centric issue, but this release of the Mate edition is definitely worth the extra attention. In the 12 months since it became an official member of the Ubuntu family, the distribution has been crammed full of new features and usability improvement, making it one of our absolute favourite Linux distributions.

At its core is Mate's continuation of the Gnome 2.x desktop, and this version, 1.12.1, is another reminder that Gnome really was on to something with its original twin-panelled desktop. Everything looks and feels so intuitive. The unification here of *GTK*, *GTK 3* and *Qt* applications, means they all look almost identical in their theming. And we're big fans of the default – it's consistent, polished and professional, and it's the theme we've found anyone dumping Windows 10 feels most comfortable with. But the desktop is also configurable, and this release bundles the (ironically named?) 'Mutiny' layout that turns Mate into an Ubuntu Unity look-alike, complete with a launch panel on the left. More than a joke, this layout will help people who have by now become used to Ubuntu's default configuration and want a lighter desktop. Lighter is emphasised in the distribution's breadth too, with added support for both the Raspberry Pi 2 and Raspberry Pi 3, complete with video acceleration for *VLC*, *FFmpeg* and *Kodi*.



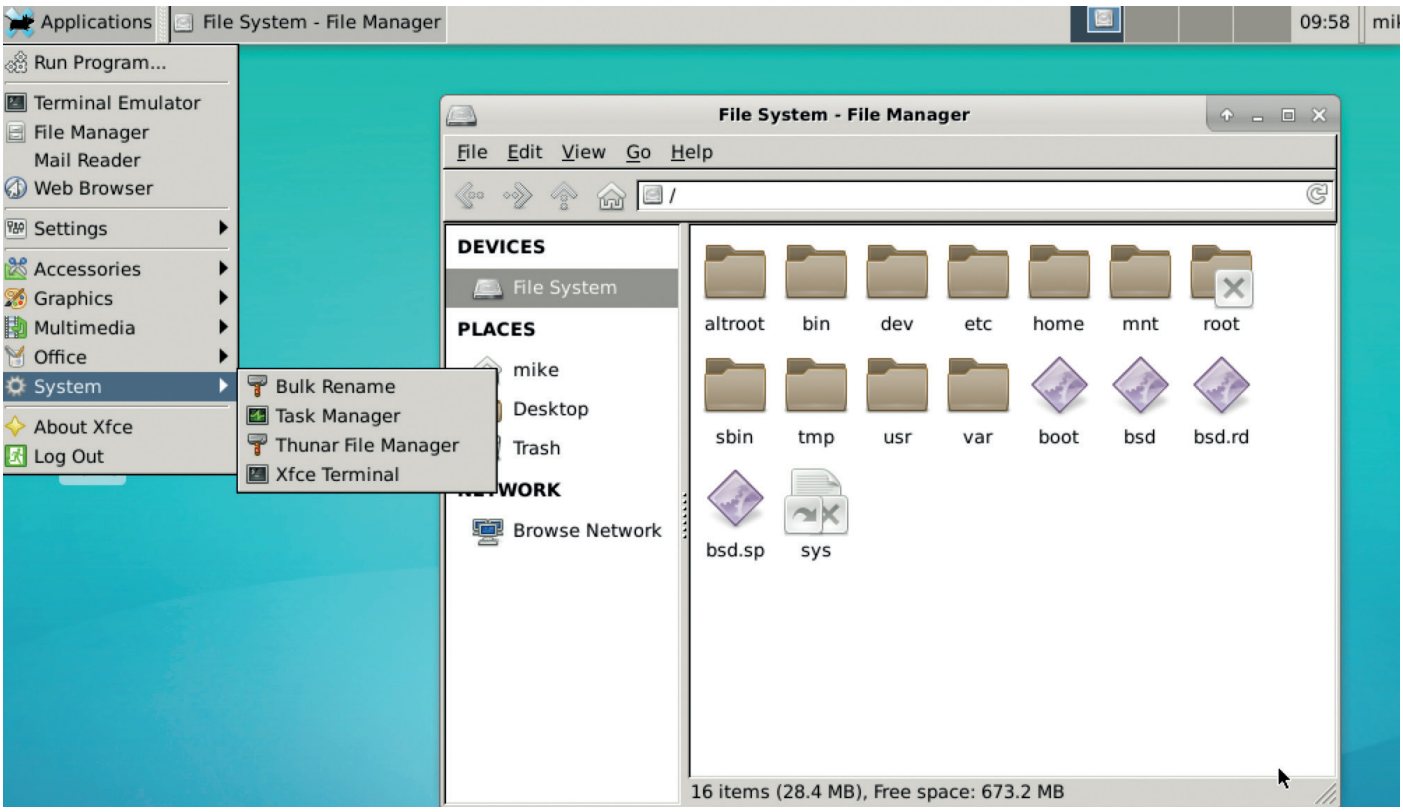
Bundling a tweak tool that gives you much finer control over the display is a brilliant idea.

We didn't have a multi-touch trackpad handy, but we have tried an earlier beta running on Entroware's excellent Apollo laptop, where the new touchpad tweaks make a huge difference. The ability to perform one-click PPA package installs via the human-curated 'Software Boutique' is also unique, where adding even proprietary software, like *Spotify*, is simple (although is there really no 'Search' function?).

We love the desktop's integrated tweaks tool, the monster list of settings and all the new panels. It's also worth mentioning that the team's primary objective is to make Ubuntu Mate "Accessible to all, regardless of language and physical ability," and there's improved support for braille displays with the new language packages adding 400MB to the size of the ISO image. The overwhelming feeling is that this release comes from a team that genuinely cares.

Not too taxing on your system or your brain, Ubuntu Mate is becoming the perfect distro for people who like to get things done.

★★★★★



OpenBSD 5.9

It's free, Unixy and ultra secure – but how does OpenBSD 5.9 match up to Linux?

Whenever a new OpenBSD release comes out, we rush over to the FTP mirrors, grab the ISO and install it in *VirtualBox*. And one thing always impresses us: the OS remains consistent and stable despite all of the changes that go on under the hood. OpenBSD is an open source Unix flavour that runs pretty much everything you get on Linux (at least FOSS), but it's incredibly hardened out of the box with security features that are optional bolt-ons in most Linux distros (and the other BSDs).

OpenBSD doesn't hold your hand: its simple text-mode command-driven installer assumes you know exactly what you want to do, and gets out of your way. After installation you have a very minimal bare-bones Unix flavour – it's your job to set it up as you like it.

So what's new in this release? It arrived slightly earlier than expected, possibly to provide a longer development cycle for the next version (which may have some big changes to multi-processor support). The biggest change is the integration of the **pledge** system call, which restricts the system calls that a program can make for improved security.

On multi-processor (SMP) machines (ie pretty much anything from the last few years) the network stack's performance has been improved, and then there are the usual small updates and bugfixes (see **www.openbsd.org/59.html** for the full list). But SMP is still an issue on the desktop, though, making the likes of *Firefox* (especially when watching video) choppier than when using Linux. The OpenBSD team recognises this as an issue. Similarly, releases are only supported for 12 months, and there's no system of binary updates built-in as standard.

Still, OpenBSD is a very well curated Unix flavour that mixes simplicity with useful security-oriented features. Tune in next month for a full tutorial...

Every OpenBSD release has a variant on the Puffy mascot – and songs as well!

Web www.openbsd.org
Platforms x86, amd64, SPARC, ARM, PowerPC
License BSD (some parts GPL)

An elegant, well-engineered OS at the forefront of security technology. Some issues with performance and long-term support though.

★★★★☆

GAMING ON LINUX

The tastiest brain candy to relax those tired neurons

ROB HALFORD RULES!

Michel Loubet-Jambert is our Games Editor. He hasn't had a decent night's sleep since Steam came out on Linux.

Things have been moving along quickly with the Vulkan API, and AMD has now released a Vulkan driver. This comes shortly after the fglrx driver was dropped from Ubuntu, a decision which now makes more sense. This should hopefully also mean that AMD graphics card owners should start to see performance shift towards what is seen on Windows, curbing the number of developers who simply don't support the cards on Linux.

To make it three out of three vendors, Intel has also gotten in on the act, showcasing its open source graphics drivers with Vulkan on *Dota 2* and some benchmarks comparing Vulkan with OpenGL and OpenGL ES, with some very hefty performance increases in the region of 30% and close to a threefold performance increase on mobile. *Wine* gamers will be pleased to know that it has also added Vulkan support.

Meanwhile, many developers have come out and expressed support for the Vulkan API along with numerous engines. With DirectX 12 only supporting Windows 10 and the recent controversy over the Universal Windows Platform initiative, which garnered a very negative response from developers and the press, the conditions do seem favourable for Vulkan to become the API of choice.

There are 1.2 million Linux users on Steam, up from around 630,000 in October 2013, and this isn't including SteamOS. The percentage has dropped slightly, but this is due to the increasing number of Steam users, which has doubled in recent years.

Payday 2

BREAKING THE LAW! BREAKING THE LAW!

Website http://store.steampowered.com/app/218620
Price £14.99

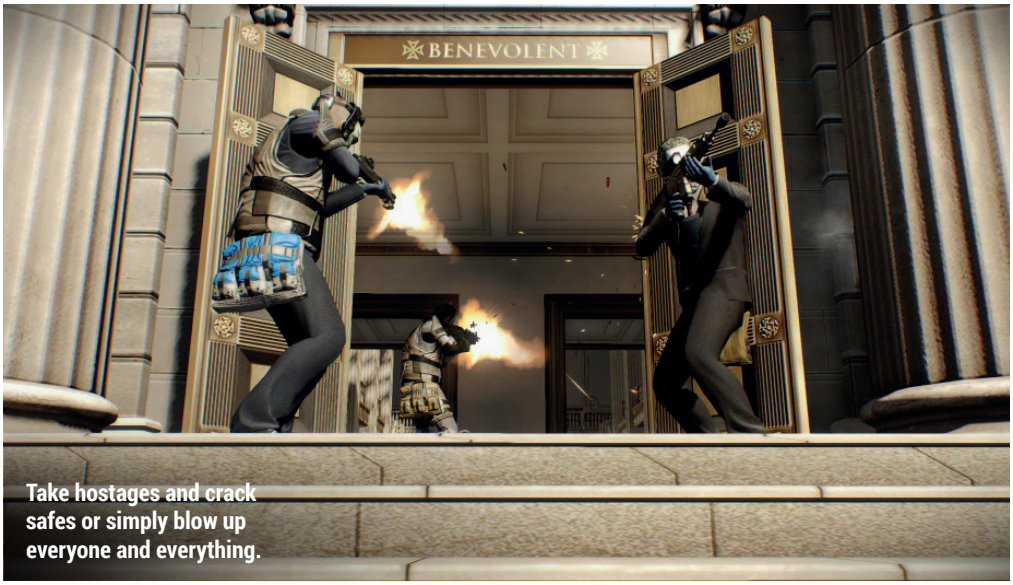
Payday 2 is one of the most popular multiplayer games around, allowing up to four players to carry out intricate heists or simply go guns blazing when it all inevitably goes wrong. The customisability is deep in this game, with unlockables and perks allowing the player to specialise or tweak equipment depending on the heist. Seeing the player's safehouse expand and upgrade over time also provides a nice sense of progression on top of the levelling system, keeping the game interesting and allowing for many hours of gameplay.

On the surface, *Payday* is simply a shooter, but more experienced players will uncover a multitude of strategic options, such as scouting out locations and disabling security mechanisms before donning the mask and going in.

The game is essentially an exclusive multiplayer experience, and while there is an offline mode, the AI does little more than serve as some extra firepower, and there's no story mode to keep things interesting. In fact, more complicated missions can be nearly impossible to complete in single player. However, at its best, when played with others online, *Payday 2* is deeply exhilarating, aided by its soundtrack, which changes in intensity to match the situation.

Graphically, the game does its job and should run well on older machines.

If you're looking for a strictly single-player experience then it's best to give this a miss since it has little to offer in that regard. However, *Payday 2* is a great deal of fun played with friends, through communication and tactics can be somewhat more limited.



Take hostages and crack safes or simply blow up everyone and everything.

> At its best, when played with others online, Payday 2 is deeply exhilarating

Villagers

A delightfully old-school city builder.

Website http://store.steampowered.com/app/412460
Price £14.99

City building god games seemed to have died out some 15 years ago, after the likes of *Populous*, *The Settlers* and the like were extremely popular. *Villagers* recreates that feel, even down to its simple graphical style.

The game has received a fair bit of flak for being similar to *Banished*, a game whose Linux port is in the works but as yet uncompleted. This seems something of a moot point since neither game is original, and as such, there is nothing particularly groundbreaking here as the player builds up a settlement while contending with the elements, famine, wolves and disease.

Nevertheless, it's great to see a return of this genre and *Villagers* does well in breathing new life into it, keeping it simple and not overloading it with forced additions. There's also a story mode, which is worth playing through to get to grips with the mechanics before embarking on a full sandbox mode.

***Villagers* is both graphically and mechanically reminiscent of a bygone age in gaming.**

Deponia Doomsday

An unexpected encore.

Website http://store.steampowered.com/app/421050/
Price £23.99

Those who played the hugely successful *Deponia* adventure game trilogy will know that the ending was divisive among fans and a sequel has been demanded for a long time in order to amend this. Out of nowhere, *Deponia Doomsday* was released, with the unfortunate precedent of creative vision succumbing to fan pressure.

Although the game comes close to needless fanservice, it ties everything together nicely while surpassing its predecessors in many areas. The game's protagonist had been criticised for being amoral and unintelligent, and this has been toned down, while puzzles are more streamlined, addressing concerns with the previous titles.

The plot centres heavily on time travel, but manages not to be too chaotic.

The game won't make a huge deal of sense if you haven't played *Deponia: The Complete Journey*, and is intended as a tongue-in-cheek love letter, delivering more laughs, more characters and a bit of closure and doing most of these better than before. *Deponia Doomsday* has turned out to be a very decent addendum to a fantastic series, bringing its lovable characters back to life.

ALSO RELEASED...

Out of the Park Baseball 17
This is worth picking up for fans of the sport management genre. However, if you already have one of the older games you should note that the new installment consists mostly of a roster update, though there is a new 3D ballpark view. If you haven't tried the series, and want a change from the likes of *Football Manager*, this is for you.
http://store.steampowered.com/app/402430

Dog Mendonça & Pizzaboy
This beautiful point-and-click adventure game – based on the comic book of the same name – is the first major game to use the FOSS Godot engine. If that alone isn't reason enough to get it, it's also worth noting that this has some of the best visuals seen in an adventure game, with its amazing pre-rendered backgrounds. There's also a nice film noir vibe and a variety of intriguing characters.
http://store.steampowered.com/app/330420

Sheltered
The nuclear apocalypse has wiped everything out and now it's time to build and manage a bunker, ensuring the survival of family members and potential drifters. Like other such games, *Sheltered* starts off simple enough, building and upgrading items in the shelter, but quickly turns into a hugely addictive exercise in multitasking as it grows. It isn't as deep as *This War of Mine* story-wise, but it's far less depressing.
http://store.steampowered.com/app/356040

A Machine Made This Book

Ben Everard is a machine that converts noodle soup into words.

Author John Whitington
Publisher Coherent Press
Price $19.99
ISBN 978-0957671126

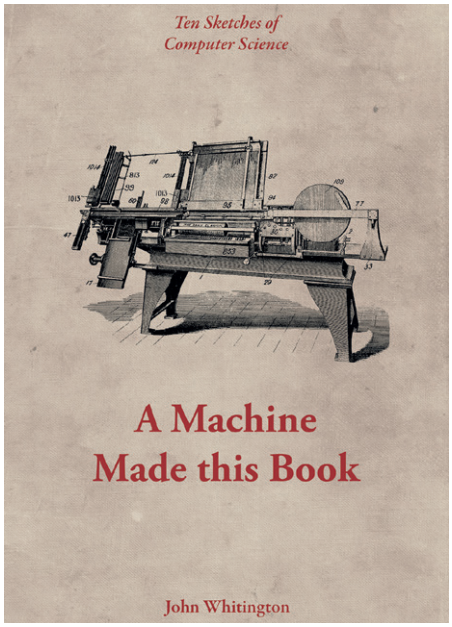
Computer science is an odd subject. It's not really about how computers work, or how to use them. The introduction to this book quotes Edsger Dijkstra as saying "Computer science is no more about computers than astronomy is about telescopes." Instead computer science is a highly theoretical subject dealing with the very nature of computation. That, of course, is a wholly unsatisfying explanation of the subject, but we simply don't have enough space to explain it fully. *A Machine Made This Book* tries to explain the fundamental principals of computer science to a non-technical audience.

John Whitington tackles this by looking at the various bits of computation involved in putting a book together. This covers everything from rendering fonts to text layout, and each task to be completed needs a separate computational solution. There's no programming involved (computer science isn't about programming), but instead, Whitington looks at the underlying ideas behind the tasks. He also covers some of the history of the subject.

A Machine Made This Book is entertaining to read and gives a good basic introduction to the subject for anyone who hasn't studied computer science. It won't make you a better computer user, but it will open your eyes to a new way of looking at computing.

A readable foray into the often opaque world of computer science.

★★★★☆



Technically, several machines made that book, but we aren't going to be pedantic.

Designing With LibreOffice

Can LibreOffice styles save **Ben Everard** from CSS madness?

Author Bruce Byfield
Publisher Friends of OpenDocument, Inc.
Price Free or £15.08
ISBN 9781921320446

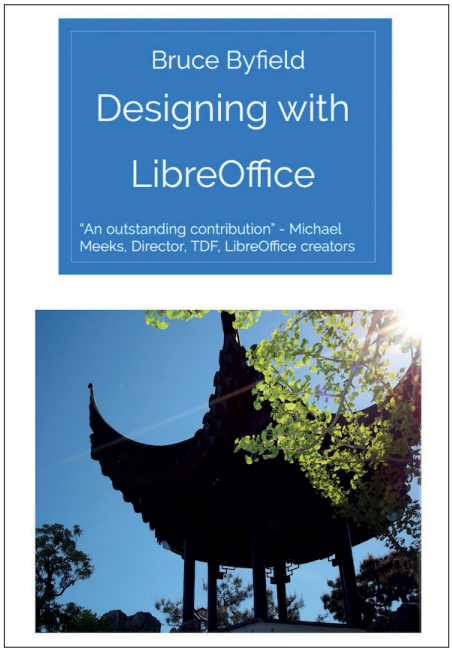
LibreOffice can be thought of as a communications tool. It can perform other tasks, but mostly it conveys information from one person to another, whether in the form of a letter, a slide show or a spreadsheet. Good design makes this communication clearer and more attractive to your audience. *LibreOffice* provides many tools to help you with this, such as the style manager and templates. Use these well, and creating good-looking documents is easy.

Designing With *LibreOffice* shows you how to make the most of all the software in *LibreOffice* (except *Base*), though over two thirds of the book focuses on *Writer*. At almost 500 pages long, it's a bit daunting, but the text is well structured, so you don't have to go through the whole thing in one go. By going through a chapter at a time, you can gradually build up your *LibreOffice* skills without having to commit weeks to the effort.

You can download the source files for the book (in the Free and non-DRM-encumbered ODT format, naturally) to take a look at how the author, Bruce Byfield designs his own documents. These source files are licensed CC-BY-SA so you can also make changes and release them if you wish. The French *LibreOffice* documentation team are currently working on a translation, so if you have time on your hands and would like to join them, download the files and start spreading the word.

If you use *LibreOffice*, this book will save you time and make your work look better.

★★★★☆



You no longer have any excuses for sloppy-looking documents.

Aurora

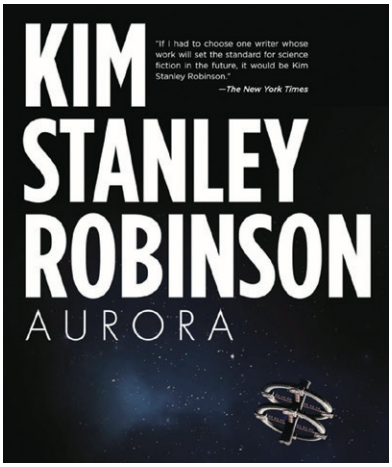
Graham Morrison cancels his ticket for Elon Musk's starship.

Author Kim Stanley Robinson
Publisher Orbit
Price £8.99
ISBN 978-0356500485

As regular readers will know, we're partial to a little science fiction, and we're not alone. Tim O'Reilly has a vast personal collection, and he even became friends with Dune author, Frank Herbert, while writing a book about him. The last sci-fi book we reviewed in Linux Voice was *The Martian*, a book recommend to us at OggCamp, and we loved it. *Aurora* comes with similar praise. It's an award winning novel from an award winning novelist with glowing reviews. It even has a somewhat similar premise to *The Martian*. It's set inside a 'generational starship,' designed to accommodate several generations as they travel for 200 years to colonise Tau Ceti. But despite this interstellar setting, the story quickly becomes one of resources and resourcefulness, and depressingly, the fragile art of community. Perhaps the reason why *The Martian*'s biologist, Mark Watney, was able to accomplish so much alone on Mars was precisely because he was alone. Put 2,000 people on a ship, it seems, and no one ever makes a reasonable decision again.

Sadly, we found this all-too-earthly premise all too familiar. *Aurora* could have been set anywhere – on a small island, on a transatlantic voyage, in Cambridge. The science fiction feels like a beautifully hung backdrop, like a Hubble image glimpsed through the starship's dimly lit interior at night. There is plenty of science – we're constantly reminded about the incredible balancing act required to maintain life, where biodiversity rules and where any change in one of a million unknowable feedback cycles can and will have catastrophic consequences. But most of the book surrounds the humans and how they arrange themselves for such an arduous journey.

Predictably, there are parts similar to William Golding's *Lord of the Flies*



In space, no one can hear you scream or read your bad book reviews.

balanced against parts similar to Prime Minister's Questions in the UKs Parliament. There's lots of diplomacy, lots of drama and lots of strategising. For us, there's too little classic sci-fi escapism. In particular, we were expecting a story more like Arthur C Clarke's, *Fall of Moondust*, in which a Lunar tourist vessel becomes stuck and submerged in a moon crater, or even Clarke's *Rendevouz with Rama*, where a feeling of alien isolation pervades every sentence as the silly humans break into an unknown craft passing through the solar system.

Bustle in your hedgerow

Our main problem with *Aurora* is simple: we found it mundane. In all the various trials faced by the on-board population ,travelling light years from one star system to another, through the vast inky blackness of space, for the first time in human history, there was too little awe at the cosmos. However, there was plenty of apathy, an all-too earthly trait. The in-flight human machinations are far more probable, and may be interesting to some, but we wanted more simple adventure and less political intrigue. And sadly, we didn't get it.

A play in two acts that could have been set anywhere.

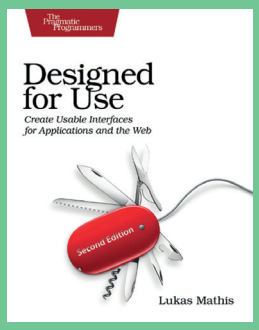
★★☆☆☆

Also released...

May 2016

Designed for Use

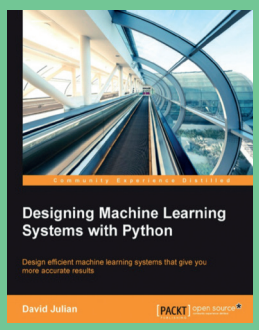
The world wide web has been around for a while now, and yet it's littered with poor design. Making sure products work the way your users expect them to seems like a simple task, and yet it's almost impossible. This is the premise of this book – now in its second edition – and we can only hope more designers and web developers take its message to heart.



Bring back GeoCities! (only joking).

Designing Machine Learning Systems

The full title of this book is *Designing Machine Learning Systems with Python*, and it's incredible to see how popular Python has become. This has happened quietly, as a new generation of developers discard compilers and complicated build systems. And machine learning is another trend that's just discovered the same thing.



Teach the machines to think for themselves!

Electronic Components vol 3

Soldering components and building electronic things is a physical process. It's difficult to use a screen at the same time, so having a physical encyclopedia to the components you're using is very helpful, especially when it comes from a publisher with the clout of Make. Books like this also make the perfect birthday gift, so we'd suggest leaving this page open...

Build something now!

GROUP TEST

Mayank Sharma is as stingy as a Yorkshireman the day before pay day, and would rather test different instant messaging clients than pay for a real phone.

On test

Gajim

URL www.gajim.org
Licence GNU GPL v3
Latest release 0.16.5
Can the lightweight app hold a candle to the biggies?

Jitsi

URL www.jitsi.org
Licence Apache
Latest release 2.8
A Java-based IM client that began life to facilitate VoIP.

Kopete

URL http://kopete.kde.org
Licence GNU GPL
Latest release 1.7.2
It might be on the way out, but it still packs quite a punch.

Pidgin

URL www.pidgin.im
Licence GNU GPL
Latest release 2.10.12
Can one of the oldest IM clients stave off competition from the young 'uns?

qTox

URL https://github.com/tux3/qTox
Licence GNU GPL v3
Latest release 1.3.0
The only client in this group test that doesn't route IMs via a central server.

Wickr

URL www.wickr.com
Licence Proprietary
Latest release 2.6.0
A proprietary software for secure communications sounds oxymoronic…

Instant messaging clients

Instant messaging is probably more prolific than email. It started as a means for sending simple text-only messages. But over the years IM has evolved into a full-fledged feature-rich medium of communication that involves images, audio and even video.

There are two very distinct settings for the use of IM, which have a direct bearing on its features. The primary users of IM are individual home users who use it for touching base with their friends and family. Increasingly, IM is also being adopted by institutional users inside the corporate environment. These of users have slightly different requirements. One key differentiator between the two types of user bases is the choice of IM protocol. For home users, the choice of protocol isn't usually a conscious one. Rather it's determined by the messaging service they choose to use for the correspondence. For this reason, a majority of home users would prefer to use an IM client that enables them to communicate on multiple different services.

Some corporate environments prefer a protocol that excels at transmitting audio and video, while others would be willing to accept a lag as long as the protocols guarantees security and privacy.

In this Group Test we'll look at some of the best IM clients that'd best serve both of these kinds of users. Besides their core functionality and support for multiple protocols, we'll also be on the lookout for the IM client that you can use in all sorts of environments and for all kinds of uses, be it personal or corporate.

> The primary users of IM are home users who use it to touch base with their friends and family

Protocol stew

Before IM clients can stream your text, audio or video over the internet, they need to first process and transform them into a form that is suitable for passing over the network. This conversion is handled by bits of code called codecs. There have been several attempts to create a unified standard for instant messaging, including IETF's Session Initiation protocol (SIP), SIP for Instant Messaging and Presence Leveraging Extensions (SIMPLE) and the XML-based Extensible Messaging and Presence Protocol (XMPP). Despite their popularity in certain use cases, the majority of free IM services by popular networks continue to use their own proprietary protocol. That said, many services have taken steps to enable users on their network to communicate with users on another network. None of the protocols, however, has received the same level of acceptance as XMPP, popularly known as Jabber. Designed to be extensible, XMPP has taken on new features and is today one of the best all-round IM protocols.

Go grab a Jabber

It's free and interoperable.

Jabber came to life in 1998 and was formalised as the XMPP protocol by the Internet Engineering Task Force (IETF). It is by far the most popular IM protocol and is implemented by several instant messaging servers. Google Talk also uses XMPP. However, the service dropped XMPP federation in 2004, though users can still connect to their GTalk account using third-party XMPP clients.

Federation is one of the best things about XMPP/Jabber. Users registered with one Jabber service can interact with users on another Jabber service without any issues. There are several XMPP servers on the internet that enable you to register a free account. Jabber.org is the original XMPP service. Although it currently isn't registering any accounts, the website hosts a list of other XMPP services that allow public registrations (**https://xmpp.net/directory.php**). Besides the link to the service, the directory also grades the services based on a number of factors including their country of origin, the Certification Authority and the XMPP server that powers the service. While you can register with any of the listed services directly by visiting their webpage, several IM clients in this group test also allow you to register with a public XMPP server.

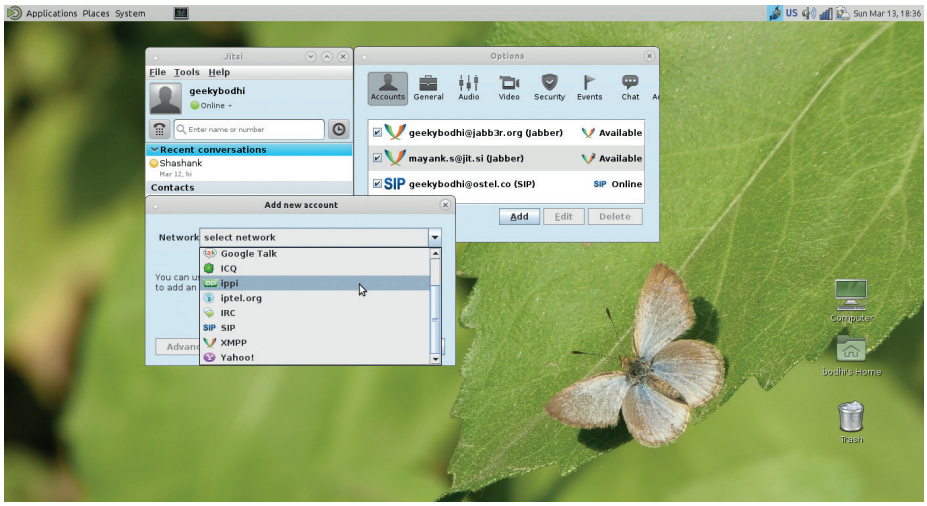
Jitsi

One with everything.

The Java-based *Jitsi* IM client runs on multiple desktop platforms and is easy to install. When you launch it for the first time, *Jitsi* opens up a window where you can enter authentication information for several different services and protocols including Google Talk, Facebook and XMPP. Furthermore, *Jitsi* is a full-fledged VoIP client as well and also lets you make calls using the Session Initiation Protocol (SIP). Unlike the other clients in this group test, *Jitsi* started out as a SIP client (and was in fact initially named the SIP Communicator) and was later renamed because of its enhanced protocol support.

The client supports one-to-one and multi-user conference chats on the supported services. You can use *Jitsi* to make audio and video calls to one user or to several users on both SIP and XMPP networks. During such an audio/video call, *Jitsi* can mute calls, put them on hold, transfer them, and can also record them. Again this feature is available for SIP and XMPP networks depending on whether the service being used supports the feature. One of the best features of the client is its ability to make registrar-less SIP calls to other *Jitsi* users on the local network.

Jitsi also has some of the best security features. Not only does it store your login details in an encrypted warehouse, it can also authenticate the identity of a contact via their unique fingerprint. If your friends are using another client, *Jitsi* can also use the standardised Off-The-Record (OTR) extension to encrypt instant messages over all the supported networks. *Jitsi* is also one of the few clients that can also encrypt the audio and video calls as well using the SRTP and ZRTP protocols over both SIP and XMPP connections.



Jitsi can help you create temporary and permanent private chat rooms.

Sharing is caring

One of *Jitsi*'s unique features is its ability to stream and share your desktop without using any of the traditional desktop streaming mechanisms such as VNC. *Jitsi* also lets you stream either the entire desktop or a portion of the screen. You can even enable your contact to remotely control your desktop. Moreover, users at both ends of a call can share their desktops with the other person at the same time.

Jitsi enables you to select contacts that you want in a conference call. The good thing about *Jitsi*'s conference call is that it lets you add contacts from different services and protocols and join them in a single conference call. One useful button is Record, which lets you save the audio from the call as an MP3. You also get buttons to create a conference call or transfer a call, both of which open a dialog box to select contacts.

The *Jitsi* developers have also created an XMPP component called Jitsi Videobridge that enables multi-user video calls. Jitsi Videobridge receives video from every participant and relays it to the others. The app also has a range of narrowband and wideband codecs, including the G.722, Speex and *Skype*'s SILK codec. *Jitsi* integrates nicely with the desktop. You get pop-up messages to indicate when a contact is writing a message, as well as the message itself if the message window isn't open. The app also has enterprise-friendly features, such as support for LDAP directories.

VERDICT

The only thing you can hold against this client is that it's resource hungry.

★★★★½

Pidgin

The ageing padawan.

The well-known *Pidgin* IM client quite possibly supports the largest number of networks. It features a tab-based interface for hosting multiple conversations simultaneously. You can use *Pidgin* to sign into multiple accounts and services at the same time. On these services, the app supports all the typical IM features such as file transfers, away messages, buddy icons, custom smilies, and typing notifications.

The default user interface of *Pidgin* is easy to navigate. Settings that influence the app can be configured from the app's main window, while settings pertaining to a particular chat can be tinkered with from the chat window.

One of the client's best features is buddy pounce. Using the feature you can configure *Pidgin* to perform an action when a buddy does something like sign-in or send a message and so on. One of Pidgin's strengths is its plugin infrastructure. Most *Pidgin* installations come with a handful of useful plugins, including an auto-accept plugin for file transfers, and a psychic-mode plugin, which pops up a window as soon as someone is typing out a message to you. There's even a plugin that hooks up *Pidgin* with *Rhythmbox* and updates your status automatically with the title of the track you're currently listening to. One useful plugin is the History plugin, which displays your last conversation with a contact whenever you open a new IM window.

Third-party add-ons

Besides the ones supported officially, there are tons of third-party plugins linked to on the project's website. However, be aware that not all of them work on Linux. One useful plugin that is often available as a separate package on most distros is the Pidgin-OTR plugin for encrypting the chat sessions.

Pidgin developers are quick to fix vulnerabilities in the app as well as in its *libpurple* library.

Some of *Pidgin*'s features work best if the contact at the other end is also using the same client. This includes everything from minor features such as the ability to buzz your contact to grab their attention to audio and video calls.

VERDICT
A long-time Linux mainstay that's ideal for texting across networks.
★★★☆☆

Gajim

The bantamweight pugilist.

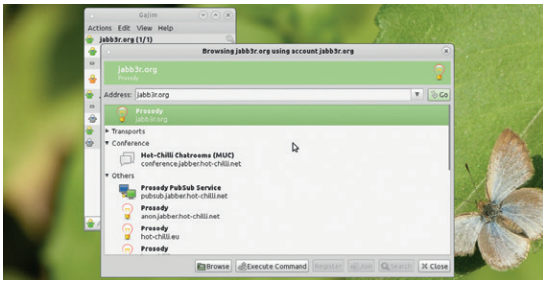
This is another small but powerful XMPP/Jabber client. If you aren't already registered, *Gajim* is aware of a large number of Jabber services and can help you set one up with ease. However, the client doesn't let you add accounts on popular services such as MSN, ICQ, AIM and others. To use it to connect with buddies on these popular networks, you'll have to rely on IM transports and gateways – but not all services support transports to other networks. *Gajim* includes the Discover Services option to list all supported transports on the connected service.

Like *Pidgin*, *Gajim* has a simple interface with a tabbed chat window. It too offers group chats, file transfers and other common IM features, such as emoticons and avatars, that you'd expect from any full-fledged IM client. *Gajim* lets you set status messages, manage contacts, customise appearance, log conversations and change themes and skins with ease. Head to Help > Features to get a list of all the supported features that can be used in your setup. If some of the features aren't available, the window will point you to the libraries you need to install to get that feature to work.

Make a noise and make it clear

You can also use *Gajim* to have voice and video chat sessions with your contacts. However, while *Gajim* works on Windows as well, the multimedia ferrying sessions work best if both clients are on Linux. One of the app's strong suits is its settings screen, which enables you to configure almost every single bit of the app such as preset messages, notifications, and the port used for file transfers.

Gajim also features an impressive list of plugins, including a version of the OTR plugin written in Python. For more security, *Gajim* also supports an experimental plugin for the OMEMO encryption, which *as per* the developers gives better encryption features than OTR. Another benefit *Gajim* has over *Pidgin* is that it lets you store passwords in *Gnome Keyring* or *KDE Wallet*.

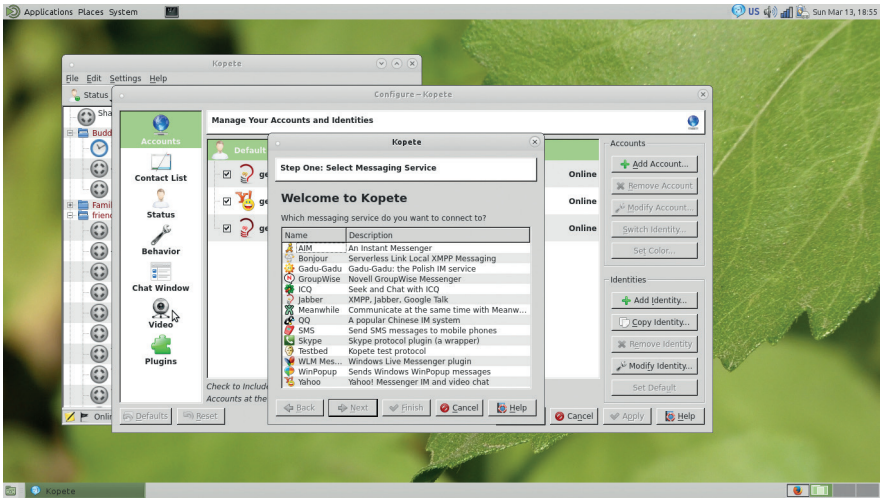


Gajim supports Bonjour/Zeroconf to discover other users on the local network.

VERDICT
A wonderfully feature-rich and versatile IM client especially for XMPP users.
★★★½☆

Kopete

Kollect kall.



Kopete lets you have video chats, but only over Yahoo's IM service.

Kopete supports a wide variety of protocols including AIM, ICQ, MSN, Yahoo, IRC and Jabber. The first step once you've installed the client is to add an account. This populates the interface with your list of contacts. The interface also has easily accessible buttons to change your status, add new contacts, and show all contacts. The chat window is also pretty much like the other clients. There are options to insert smileys and send files. If you're in a group chat, the window will show a list of all participants.

True to its name, *Kopete* is your typical KDE app and lets you customise virtually all aspects of the client. The panel for configuration of notification is overwhelming. Unlike the other clients, *Kopete* has a highly useful account management scheme. The app encourages you to first create identities for your different circles of friends, such as work, family, and friends. Next up, you should associate each identity with as many accounts on any of the supported networks. This scheme lets you sign into multiple accounts and add people from different services in such a manner so as to never crowd the interface.

Unlike *Pidgin* and *Gajim*, *Kopete* doesn't have many plugins, since the app itself is so tweakable. There's a plugin that does on-the-fly spell checking while another aggregates statistics about your conversations. There's also a Contact Notes plugin for adding personal notes to each contact.

Talking of contacts, *Kopete* can group different contacts that correspond to the same person on different networks into one single meta-contact. So if you have a friend who is on several networks, instead of him hogging the app's interface, you can group all his identities into one single entry. *Kopete* can also stop all notifications, except those of selected contacts. Then there's the Highlight plugin, which will call for your attention when a message in a group chat matches a specified regular expression. Geekier users will recognise the benefit of such a feature, especially in an IRC chat room.

In terms of security. *Kopete* can encrypt your conversations once you've enabled the OTR plugin. In *Kopete* the OTR plugin offers four different enabling policies. The default policy labelled Opportunistic works best for a majority of users and will only start the OTR session automatically but only if the user on the other end also uses the plugin. There's also the Privacy plugin, which helps filter certain messages. *Kopete* determines which messages to filter by dividing your contacts into a whitelist and a blacklist. You can also filter messages by specifying a list of words and ask *Kopete* to drop messages that support either some of these words or all of them.

VERDICT

A highly konfigurable IM client with exceptional contact management.

★★★☆☆

IM vs VoIP

Two side of the same coin?

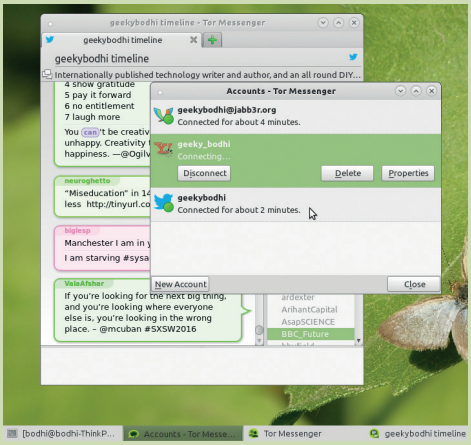
Functionally speaking, both instant messaging and VoIP appear the same. You can use both to exchange text messages and files and make audio and video calls. But in technical terms, the comparison is akin to comparing apples and oranges. To better understand the differences (and similarities) let's compare their protocols. The main IM protocol is XMPP, while SIP does the same job for VoIP.

Both XMPP and SIP are what are known as signalling protocols. They are designed to establish channels that allow two clients to communicate packets of data with each other. However, neither SIP nor XMPP technically carry the actual voice/video data. This is left up to other protocols, which are negotiated by the signalling protocol.

The one major difference between the two is in terms of their design – SIP is a peer-to-peer protocol and XMPP is client–server. Furthermore, the two protocols have evolved differently. SIP was designed primarily with just signalling as a goal, while XMPP was designed primarily with messaging and presence as a goal.

Over time both have gradually extended into each others' realm though: XMPP added an extension called Jingle for session negotiation and SIP added an extension called SIMPLE to support IM and presence.

All said and done, primarily due to their pedigree, SIP is particularly suited for telecommunications, and is offered by vendors for trunk services to and from the PSTN while XMPP is primarily used for federated IM.



Use the Tor Messenger to route OTR-encrypted messages via the Tor network.

qTox vs Wickr

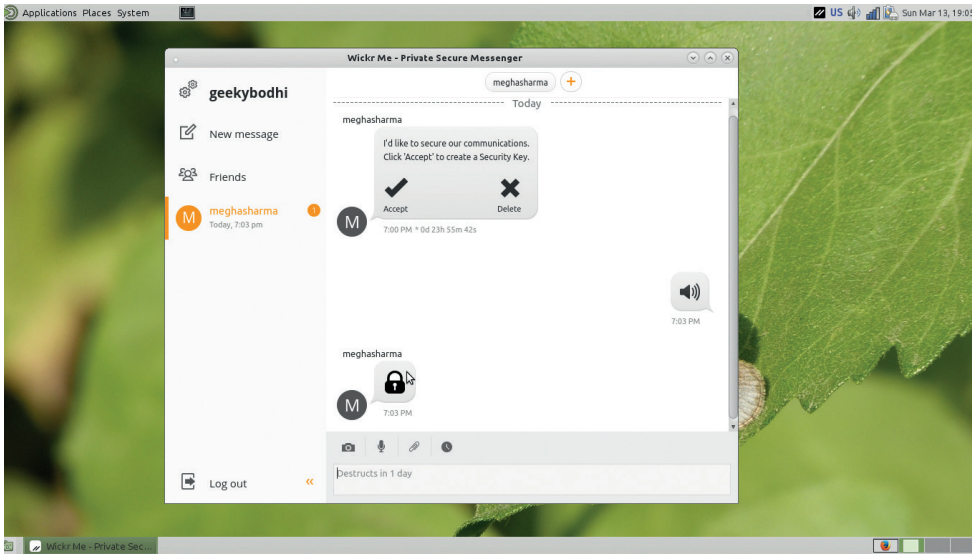
Messengers for the post-Snowden internet.

There are several apps designed to protect different facets of your online life from unwanted prying eyes. *Wickr* and *qTox* are two popular apps that help safeguard IM communication.

Wickr enables users to exchange end-to-end encrypted messages that besides text can include photos, videos, and file attachments. Unlike the other apps in this group test, *Wickr* enables users to set an expiration time for their encrypted communications. The app is available for all major mobile and desktop operating systems.

The service encrypts all communications locally on each device with a new key generated for each new message. Furthermore, although *Wickr* communication is routed through a server, the service is designed so that *Wickr* itself does not have access to your passwords, encryption keys nor the messages. For further privacy, *Wickr* strips metadata from all content transmitted through its network.

You can get started with the service without giving out your name or even your email address: just pick a username and password and you're good to go. But if you do give it your email address, the service can search your contacts for people you know who also use *Wickr* and automatically adds them to your contacts list. Besides one-to-one chats, *Wickr* lets you chat securely with a group of up to 10 users.



Wickr **gives each message a live countdown so you know how long you have until it disappears.**

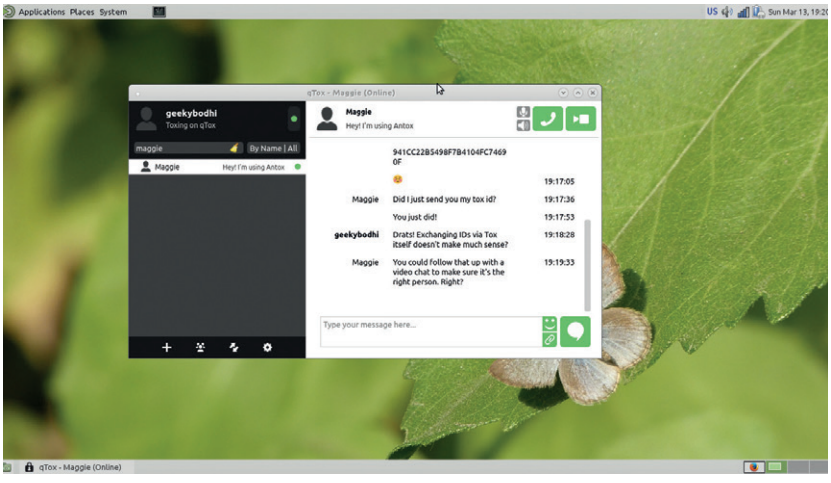
Like *Wickr*, the *Tox* protocol also does end-to-end encryption of text, audio and video messages. However, *Tox* addresses the shortcomings of *Wickr* – it's open source and doesn't route your messages via a central server.

Detox chatter

The *Tox* protocol uses the same peer-to-peer technology used by BitTorrent to provide direct connections between users. Instead of usernames, every user in a *Tox* network is represented as a string of bytes, which is their *Tox* ID. Also all chats are encrypted using the NaCl encryption library. Since there's no central server, users can simply fire up their *Tox* client and add friends without signing up with a service or configuring an account.

Tox offers a couple of different clients for all the popular desktop and mobile platforms. The basic usability is pretty much the same across the clients. You launch the client and assign yourself a nick that generates a *Tox* ID that you can then pass on to friends. When your friends add you, you get a notice, which you'll have to accept after verifying their *Tox* ID, before you two are connected. If your friends are using a *Tox* mobile client, you can save and send them a copy of the QR code image generated by your *Tox* client. The *Tox* mobile clients can scan the QR Code and add users automatically.

Not all *Tox*'s features are available across the clients. Text-based IM works across all supported platforms and their users can also securely share images and screenshots. Additionally desktop users across the clients can make audio and video calls to each other without any noticeable delay and can also host group chats with others users of the desktop clients.



The *Tox* protocol also supports the use of custom 'tox:' URIs to create links that'll launch a user's *Tox* client and automatically add you as a contact.

VERDICT

QTOX Ideal client for exposing the best features of the new P2P protocol. ★★★★☆

WICKR The best option for sending amnesiac instant messages. ★★★☆☆

OUR VERDICT

Instant messaging clients

There's very little to choose between *Pidgin*, *Kopete* and *Gajim*. All three are wonderful Jabber/XMPP IM clients and each has its own unique features. Many distros have replaced *Kopete* with *Telepathy*, but some like OpenSUSE continue to stick with the old but solid app that still works as advertised.

Similarly, *Pidgin* and *Gajim* are exceptionally well performing IM clients that can take on new features thanks to their diverse list of plugins. While *Pidgin* is convenient for connecting with friends on well-known public services, *Gajim* scores for being an all-round XMPP client.

Similarly, there's little to choose between *Wickr* and *qTox*. *Wickr* is closed source and uses a proprietary algorithm but comes from a developer with impressive security creds and has been audited by several reputable organisations. In fact, *Wickr*'s bounty for discovering a vulnerability in the app is still lying unclaimed. The app's only real shortcoming is its inability to do real-time audio and video conversations. But if you wish to send snapchat style self-destructing messages, there's no better secure option than *Wickr*.

On the other hand, the *Tox* protocol is open source and uses P2P technology instead of a centralised server. Like *Wickr*, *Tox* also has clients for desktop and mobile OSes. Unlike *Wickr*, however, you can use the *Tox* desktop clients to have voice and video calls.

That leaves us with *Jitsi*, which includes the best features of all the other clients. It's open source, supports multiple popular public networks and protocols including XMPP and SIP, and can be used to send files. On top of this, *Jitsi* can host individual as well as group voice and video chat sessions. The app also implements the ZRTP security protocol designed by the creator of PGP, Phil Zimmermann. *Jitsi* scales well and can be used for everything from short text-based chats to full-fledged multi-user video sessions.

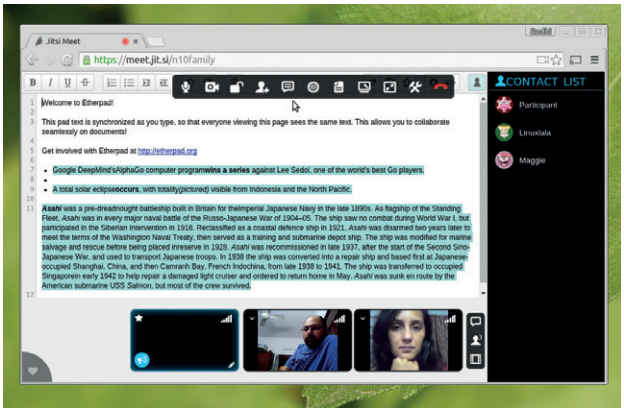
> Jitsi implements the ZRTP security protocol designed by the creator of PGP, Phil Zimmermann

Your own private IM server

If you're sold on the idea of offering an IM service to users on your network, it's best to roll your own IM server.

Just like the clients, there's no dearth of open source XMPP-based IM servers. Some of the popular ones include *Ejabberd*, *Prosody* and *Openfire*. All these servers offer IM services based on the XMPP protocol but come with their own unique set of features that make them suitable for different kinds of deployments. The feature-rich and very extensible *ejabberd* is one of the most deployed servers. If you're looking for a server for a small team, there's *Prosody*, which can even be run on a Raspberry Pi. The Java-based *Openfire* is resource-hungry but also offers its own cross-platform clients.

Deploying an IM server is pretty similar to deploying any other server software. The Turnkey Linux project also produces a ready-to-roll JeOS appliance for the ejabberd server (**https://www.turnkeylinux.org/ejabberd**). You can use this appliance to evaluate the server or even deploy it on a production environment inside your network or on virtual and internet-based infrastructures such as Docker and OpenStack.



You can host video conferences on your own infrastructure by deploying JitMeet.

1st Jitsi

Killer feature: ZRTP secured video calls.
https://jitsi.org
An exceptional IM client that works well for all kinds of users.

2nd qTox

Killer feature: Full featured peer-to-peer IM sessions.
https://github.com/tux3/qTox
The only IM client to avoid routing sessions via a central server.

3rd Wickr

Killer feature: Encrypted messages with an expiry date.
www.wickr.com
Designed for paranoid users who prefer to put their privacy over any other feature.

4th Gajim

Killer feature: Lightweight but with all the bells and whistles.
https://gajim.org
Implements the best features of XMPP but works well only when all parties are using the same client.

5th Pidgin

Killer feature: Supports a wide range of networks and protocols.
https://pidgin.im
Works well for traditional IM services, but getting audio and video to work is a black art.

6th Kopete

Killer feature: Helps curb account proliferation.
https://userbase.kde.org/Kopete
A tweaker's haven with its best days behind it.

Subscribe

shop.linuxvoice.com

Introducing Linux Voice, the magazine that:

- **Gives 50% of its profits back to Free Software**
- **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 100 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.

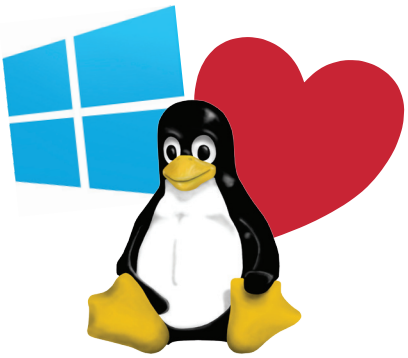
NEXT MONTH IN LINUX VOICE

ON SALE THURSDAY 26 MAY

SYSADMIN 101

If you use Linux, you're an admin – even it it's just one machine with one user. Join us, and discover the power at your fingertips.

EVEN MORE AWESOME!



MS + Linux 4 ever

The recent change in attitude from Microsoft towards Linux has left us confused, bemused and bewildered. Is it real love, or are we being used?

Simon Phipps

The former president of the Open Source Initiative, advisor to approximately 47% of all Free Software projects and all-round good egg pops in for a chat.

OpenBSD

For bulletproof security, a sensible filesystem and a pretty decent set of native applications, why not give this alternative operating system a try?

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, 2nd Floor, 5 Churchill Place, Canary Wharf, London, E14 5HU
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

ISSN 2054-3778

Subscribe: **shop.linuxvoice.com**
subscriptions@linuxvoice.com

FOSSpicks

Sparkling gems and new releases from the world of Free and Open Source Software

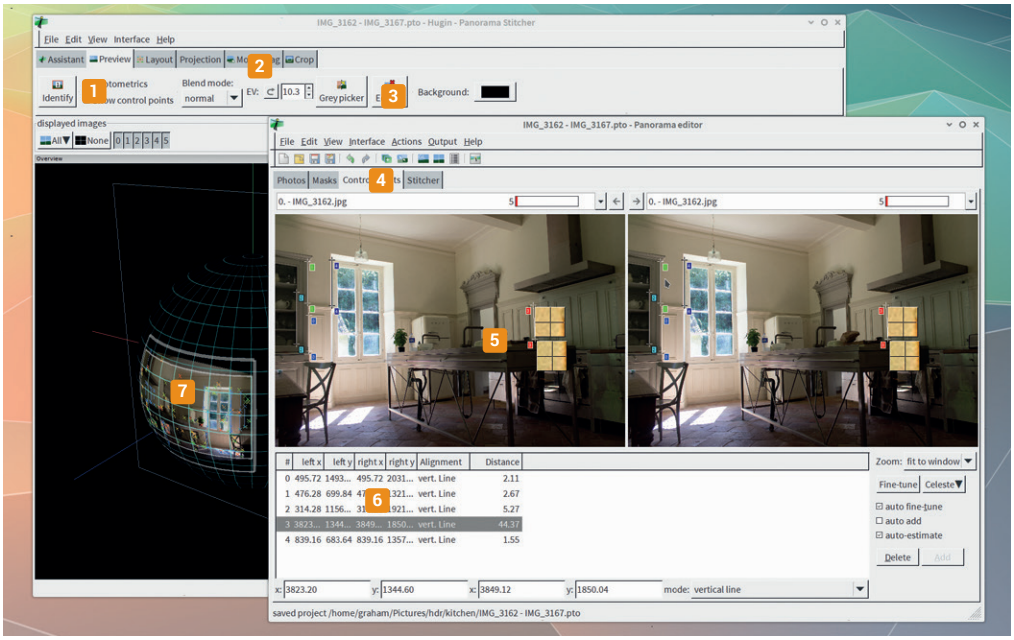
Our benevolent editorial overlord **Graham Morrison** tears himself away from updating Arch Linux to search for the best new free software.

Photo stitcher

Hugin 2016.0.0

We've been using *Hugin* for a long time. It's the best open source software we've found for stitching together multiple photos taken at an offset and for turning them into a single panoramic image. It does this brilliantly, working with 16 bits per colour channel to dynamically adjust brightness, colour, angle, vignetting (brightness and darkness at the edge of an image) plus various chromatic and visual distortions, creating an output that's not just seamlessly stitched together, but also of a higher quality than any one of the single images used to make the composite. *Hugin* is also capable of some rather clever projection mapping, such as with vertical control points and can generate high-dynamic range photography.

It does all this by offering dozens of control options via lots of different panels. But these options can be cleverly culled by allowing the user to select between Simple, Advanced and Expert views in the GUI. 'Simple' is never going to be as simple as the point-and-click of your smartphone's camera application, but *Hugin*'s output is always better, at least in our experience, especially when you zoom in and examine the details of a join. With a little tweaking in *Hugin*, you can't see them.



1 Simple Mode Import photos, click on Align, and mess around with the projection, crop and preview **2 Tabbed Windows** Assist Mode will help with importing your photos **3 White Balance** Each panel has a View Options – White Balance for the preview **4 Panorama Editor** More complex editing can be done from a different window, including masks and manual stitching **5 Control Points** Editable in either view, you can change how and were photos are welded together. **6 Photo List** *Hugin* works as well with two images as it does with 20, although everything takes longer **7 Projection** Even single images can be projected onto spheres.

A stitch in time

We've found that you can generate excellent panoramas by simply importing a group of images, playing slightly with the alignment, and generating the large output file, with no specialist skills necessary. In Simple mode, to stitch a couple of photos together into a panorama, use the Add Images button to insert your images and click on the Align button. This process should automatically detect and anchor control points within the overlapped parts of your images. You can then play with the projection and the cropping and generate the output.

Hugin is also an exceptional application if you want to dive into the details. There are major improvements in this version to the way colour profiles are managed and used by the many command line tools that augment *Hugin*'s GUI functionality. There are lots of bugfixes too, and this version is noticeably more stable when manipulating groups of massive images. We also really like the **align_image_stack** command for merging bracketed images, although the internal Align works just as well. Nothing comes close to *Hugin* for its stitching quality and output for panoramic or mosaic photo composition.

Project website
http://hugin.sourceforge.net

System information

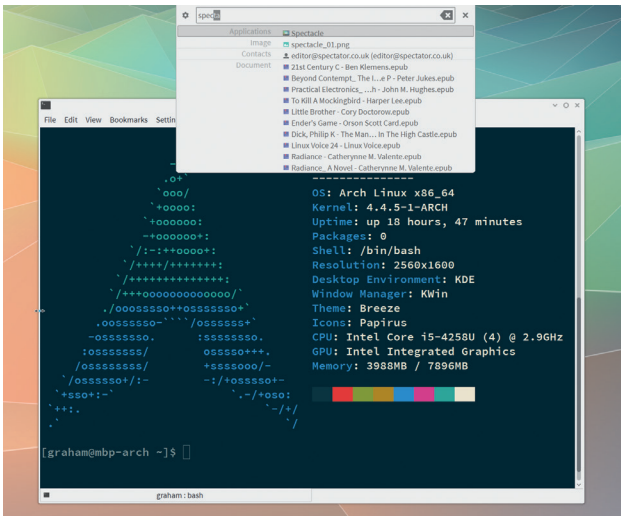
Neofetch 1.5

We often find ourselves using a variety of commands to retrieve basic system information, whether we're trying to work out which version of Ubuntu we're using, the type of kernel running, or how much RAM there is on the system.

There are lots of tools that provide this information, especially from the GUI, but it's more common for us to need these facilities from the command line, where we can probe the status of a machine remotely. *Neofetch* is one of the nicest of these commands we've found, providing all the information you need with just a simple command, and presenting the data it gathers in a very photogenic way. At its most basic, for example, typing **neofetch** will draw a logo for your chosen distribution, tell you about the kernel, RAM, CPU, GPU and your terminal's colour palette, but it's also capable of a lot more. It can tell you what music the machine is playing, for example, and even display the cover art (and other images) within the terminal.

The many arguments that can be used when launching are used to change almost everything about what and how the data is displayed, from the colour of the output and the kind of information displayed. If you need the same modified output every time you run the command, you can put all those arguments into a configuration file which will be used as the default when you run the command without further arguments. You can even change the order of the output and the formatting, as well as remove information from the output you don't need.

> Neofetch provides all the information you need in a very photogenic way



We know of no other command that will reveal which icon set you're using from the terminal!

Project website
https://github.com/dylanaraps/neofetch

Uber system monitoring

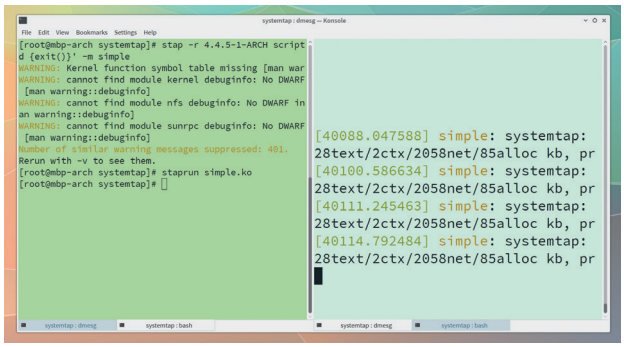
SystemTap 3.0

Techniques like probing the **/proc** virtual filesystem and tools like *Neofetch*, above, are great if you need quick and accessible information about your Linux system. But if you need more control and more data, you need a more ambitious tool.

SystemTap does this and more with the emphasis on 'complexity'. It's akin to Sun Microsystem's revolutionary (and complex) *DTrace*. *SystemTap* translates its own scripts into native C so that your system's C compiler can run the script as a kernel module. This means *SystemTap* can get completely inside the running state of your system, from network packets and process latency to the kernel and scheduler – like a series of nanorobots for your system. It's best supported on Fedora, although we had little difficulty getting the tools installed and running on Arch. In particular, you need to make sure you build the kernel module against your kernel version, using the **staprun** command, after which you can run the script embedded within the module with the **stap** command.

Power, absolute power!

It's perhaps best suited to a developer debugging their system applications, or an embedded Linux engineer, but it's also useful for systems running containers or many different Linux instances, as you can monitor system resources outside of their execution environments. Despite this complexity , you may never need to write your own scripts, as there's a brilliant community and some excellent online documentation. In particular, the official website contains 156 examples, and includes samples on how those scripts can be run and used, as well as how to interpret the output.

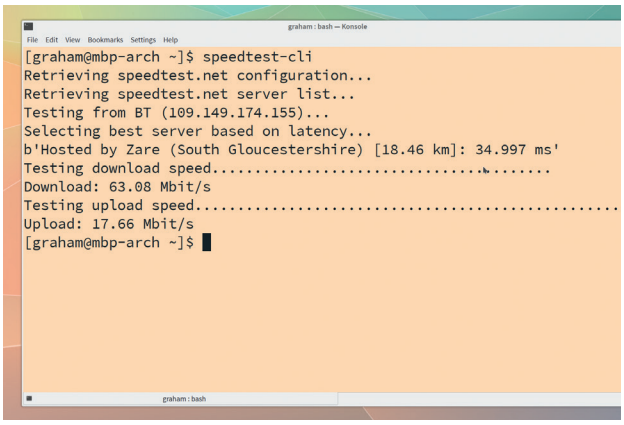


We could fill an entire issue on how to use *SystemTap* – it's capable of displaying almost every little thing about your system.

Project website
https://sourceware.org/systemtap

Performance monitor

Speedtest-cli 0.3.4



Use this simple command line utility to test your internet speed, and even automate speed checks over a period of time.

We would all like to think that, in this world of super-fast broadband, the specific speed you get isn't as important as it used to be more. Unfortunately, this isn't the case. Even if you've got the best connection possible, ensuring you get what you pay for is just as important today as it was when you had to listen out for the Hayes initialisation string to execute on your modem.

Your connection may be being throttled or the speed may be way off what you were sold. Bandwidth may also fluctuate throughout the day, or week, or month, and without proper monitoring, you have neither feedback nor recourse with your ISP. This is particularly true in the UK, where the hyper-competitive market for 'unlimited broadband' means ISPs often try to transparently adjust broadband speeds according to demand or usage levels.

The solution is to use a speed checker, and there are many. The most common are either web pages or apps for a mobile device, but *Speedtest-cli* is one you can run from the command line. It's as simple to use as typing its name, although a handful of arguments enable you to change the testing server, choose between bytes and bits and share your results via **speedtest.net**.

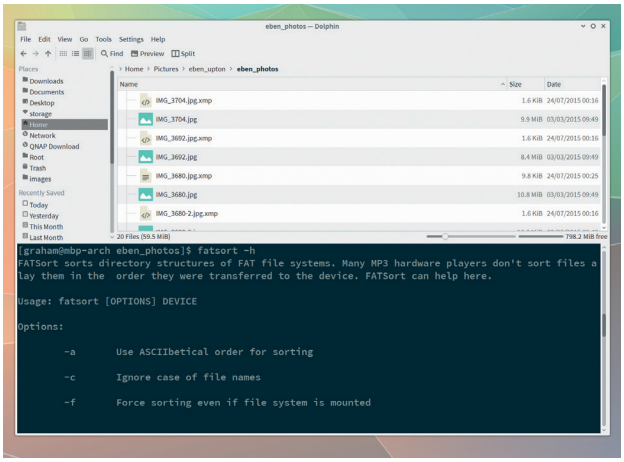
By default, after a few seconds (depending on the speed of your connection), you'll be presented with both a download and upload speed. But the best thing about this simple utility is that using nothing more than **cron**, you can automate this test and simply log these speeds so you can see how the bandwidth available to you changes over time.

> Speedtest-cli is a broadband speed checker you can use from the command line

Project website
https://github.com/sivel/speedtest-cli

USB file sorting

FATSort



There's not much to see, but *FATSort* will turn random lists of files into files properly ordered by file name.

This is a brilliant little tool, and we owe a huge thanks to james_olympus on our IRC channel for its discovery. He submitted this as a Find for our podcast, and it's one of those small, simple utilities that makes you wonder how you ever did without it.

It fixes a small but annoying problem with the way some devices read file names off FAT12, FAT16 or FAT32 partitioned storage. Mostly, that means USB sticks and SD cards being read by music players, such as those found in cars or home stereos. But it also applies to lots of internal storage found on MP3 players. The problem is that while FAT is widely used, thanks to its history with Microsoft Windows, the filesystem itself is very simple, and most implementations of the filesystem are even simpler. So simple, in fact, that many devices will read the filenames off a device in the order that they're written to the blocks on the device, rather than sorting their names into something more logical. This is the order your files were physically written or copied, rather than something sensible such as their actual file name.

This is particularly important with MP3 players, because many of us arrange our music collections into folders for each album, and prefix each track name with the number order of each track, such as 01, 02, 03. This means that when filename ordering is ignored, so too is the original ordering of the album. FATSort solves this problem, and all you have to do is enter the device name following the command itself – **fatsort /dev/sda**. Brilliant.

> We owe a huge thanks to james_olympus on our IRC channel for this discovery

Project website
http://fatsort.sourceforge.net

Tiling Terminal Emulator

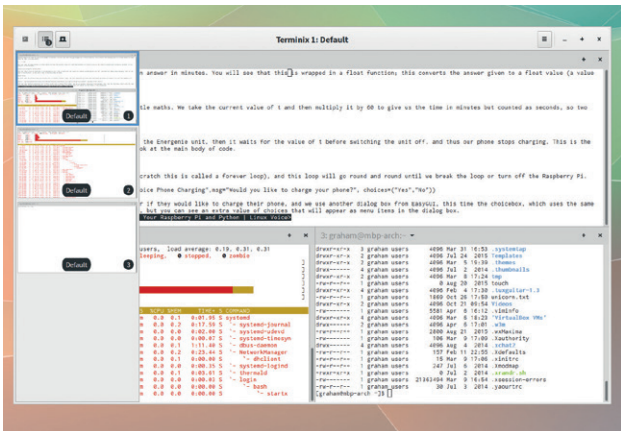
Terminix 0.56

We've noticed a resurgence in all things command-line. Perhaps it's to avoid the constant distraction of the internet via a browser, or the way command line tools are generally engineered to do one specific job. Either way, many of us are replacing gargantuan desktop applications with simple utilities rendered in 12-point Courier.

We too are more productive with these single-task tools, and because many have been around for so long they have unrivalled stability, support and third-party integration. *Terminix* is a terminal emulator for these tools that acts like a tiling window manager, enabling you to easily manage multiple terminal sessions in the same way you would using something like *Xmonad* on the desktop. Unlike *Xmonad*, though, *Terminix* is thoroughly modern and easy for any beginner to use. It's built atop *GTK 3*/Gnome 3, which makes it particularly well suited for desktop users looking for a powerful interface to the command line, and it can replace your standard terminal without causing you any transitional pain.

Two worlds collide

You can easily spawn new sessions, splitting the display either horizontally or vertically, all without remembering a single keyboard command, and all with Gnome's lovely transitions. Sessions themselves can be saved and restored via simple Json-formatted configuration files, and there's full support for terminal profiles, as well as light and dark themes for the remainder of the interface. A thumbnail overview enables you to skip between open sessions and everything works quickly and without clutter. If you find yourself using the terminal often, especially if you're a Gnome user, we can't recommend *Terminix* enough.



Forgive us for running a *GTK 3* terminal emulator under KDE, but it was all we had to hand.

Project website
https://github.com/gnunn1/terminix

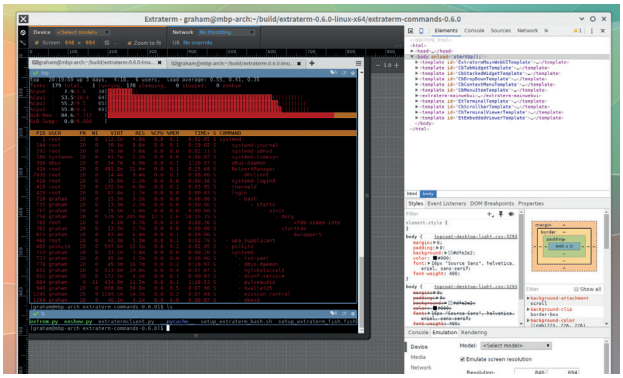
Super-powered terminal

Extraterm 0.6.0

Keeping with the terminal upgrade theme (see above), *Extraterm* is a new terminal that attempts to power-up the old model. Like *Terminix*, it supports splitting the view vertically and tabbed sessions, but this layout frippery isn't its only trick.

There's the selection mode, for instance: press Ctrl and space together and a blinking cursor appears. You can now move this around the console and use the Shift key to select areas of text. The usual shortcuts can be used to copy and paste, with the Ctrl+Space combination used to go back to normal edit mode. It reminds us of a similar mode on the Commodore 64 where you could use the shortcuts printed on the lower side of the keys to move the cursor away from its editing position – although you couldn't copy and paste back then.

There are also customisation options for *Bash*, *Zsh* and the up-and-coming *Fish* shells. These come packaged within a Zip file downloaded from the project's GitHub page, and you can load their settings by typing **source** followed by the name of the file corresponding to your shell – **source setup_extraterm_bash.sh**, for example. With this done, commands are now framed by coloured backgrounds, which is useful for clarity. You can also pop the output into a new tab, or close the output completely. There's also additional **show** and **from** commands. The first will prettify the display of a file, including syntax highlighting for code and embedded images for pictures, whereas **from** lets you easily use the output from one command as the input to another. It's all quite esoteric, but powerful too, and worth a look if you're considering a new terminal.



***Extraterm* is built using the Electron web platform, which means it's mostly JavaScript and you get all the debugging tools for free.**

> Extraterm is esoteric but powerful, and worth a look if you want a new terminal

Project website
https://github.com/sedwards2009/extraterm

Image viewer

PhotoQt 1.3

We know there are lots of photo viewers available for the Linux desktop. Many of them are very good. But like music players, we feel there's always a place for something that's different or attempts to fill a niche not covered by the alternatives.

PhotoQt is a viewer that's designed to be "good looking, highly configurable, yet easy to use and fast," according to its website. We downloaded, built and installed the latest version, and the first thing you notice when you run the application is that it's full screen! This isn't unusual for photo viewers, especially of the Adobe variety, but it's still a shock to our windowed brains, which by nature and training are fractured into different areas.

Fortunately, you can use the hover menu that appears in the top-right of the display to change this setting, along with plenty of other options. Full-screen or windowed (better without the window borders), *PhotoQt* is sublime. Move the cursor down to the bottom of the window, and the current folder's thumbnails appear. Move it to the left and you get the EXIF metadata for an image. Move it to the right, and a quick settings panel slides in.

Despite the plethora of photo viewers for Linux, we feel there's still space for a minimal, quick, and accurate tool like *PhotoQt*.

Pretty as a picture

You can also scale, flip and rotate images with an editable context menu that can also send the image to your pre-configured editor of choice, and you can easily set an image as wallpaper or trigger a slide-show.

> Full-screen or windowed (better without the window borders) PhotoQt is sublime

PhotoQt does everything you need in a quick, concise and visually appealing way, and leaves the largest part of its display area to the most important job – viewing your photos.

Project website
http://photoqt.org

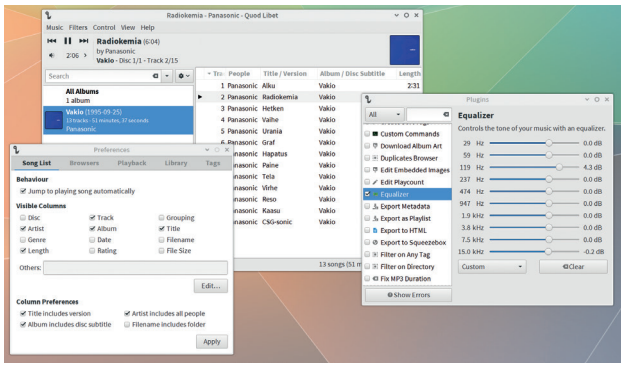
Music player

Quod Libet 3.6.1

This is a major update to a music player that's been around for a long time.

Picking up our dusty 'Latin for Dummies' book off the top shelf, it reveals *quod libet* is Latin for 'whatever you wish,' which also happens to be an old style of music containing more than one melody. All of which is a modest way of saying this is a music player and management application. It's designed to 'Just Play Music,' rather than get between you and your collection, and it does this mostly by being quick and easy to use.

On first launch, you need to say where your music files are stored. After your library has been scanned, you can start listening to music. We love the user interface. It's simple while remaining powerful enough for our needs. It's also album-centric, defaulting to playing an entire album in order rather than constantly adding to a dynamic playlist, but different views let you construct playlists, browse internet radio stations or use the powerful tag system to find the music you're looking for.



***Quod Libet* is one of the best players we've found for taking advantage of tags embedded within your music files.**

Power up with Python

The Python plugin system is especially impressive, listing dozens of different add-ons, including lyrics, equalisation, Squeezebox-export, song-lookup via acoustic fingerprint and lots more. Its main audio back-end is *GStreamer*, which makes it compatible with almost any file type and desktop, although it did have trouble with a few of our FLAC files.

> Quod Libet defaults to playing an entire album in order rather than a playlist

However, if you're looking for a new music player with few dependencies and a powerful minimal user interface, this latest version is highly recommended.

Project website
https://quodlibet.readthedocs.org

FOSSPICKS Brain relaxers

Gameplay streaming

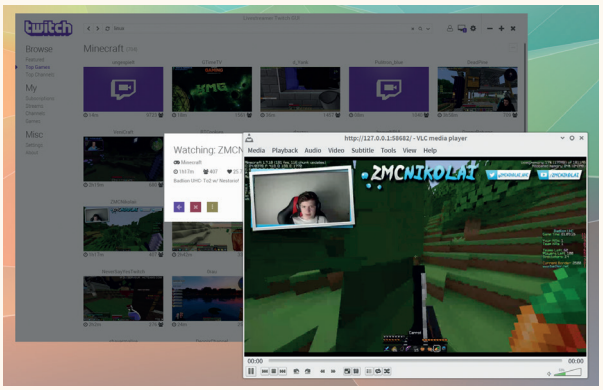
Livestreamer Twitch GUI 0.12

Back in the old days, before the internet, before anonymous trolling and the Snooper's Charter, friends would gather in a single place – not to talk conspiracies, but to watch and play video games with each another. Now that kids can no longer go outside, the modern analogy is watching other people playing video games online, and the most popular service that facilitates this video game voyeurism is called *Twitch*.

Twitch is actually quite compelling, with games like *Rocket League* and *Counter Strike* attracting the same online viewing devotion that you might expect at a football match. The only problem with *Twitch*, especially for us Linux users, is the lack of a non-Flash version on the website. *Livestreamer* is a command line tool that grabs audio and video data from websites like these and sends them on to your favourite video player, and *Livestreamer Twitch GUI* does exactly what it says – it's a simple interface to the content available on *Twitch* that when selected will open the stream in your favourite player.

It's brilliantly simple to use, and much, much easier than trying to do the same thing in your standard browser. With the video extracted from a browser and pushed into something like *VLC*, *Twitch* becomes much more responsive and far less demanding of your CPU. Whether watching other people watching video games is a good use of your own time, we wouldn't like to say!

> This is a simple interface to open streams from Twitch in your web browser



***Twitch* is great for watching video games, but some crazy people use it to stream live from their Linux desktop too (thanks to loangogo for this find!)**

Project website
https://github.com/bastimeyer/livestreamer-twitch-gui

Retro game player

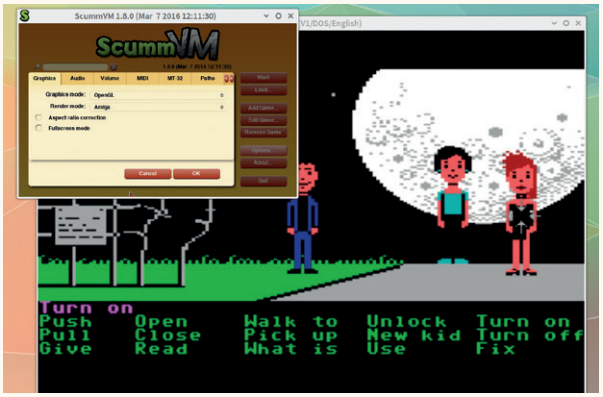
ScummVM 1.8.0

We know there can't be many people who haven't tried *ScummVM* already. It's a wonderful system that runs on almost everything – from a Raspberry Pi to any humble Android device, and it enables you to play many of the point-and-click classic games from the 80s and 90s that have never been surpassed.

Think of classics like *Monkey Island 2*, or *Day of the Tentacle*, or the free *Beneath a Steel Sky*, or the older *Kings Quest*, *Space Quest* and *Manhunter* series. You make your way by selecting items and verbs and clicking through the artwork. It sounds simple, but there are few genres that can compete with this form of gaming entertainment. Most importantly, these adventures are just as good today as when they were made, and have been lovingly re-framed to work within *ScummVM*, saving you from swapping floppy disks on a 30-year-old Amiga 500.

Like the developers of interactive fiction interpreters and arcade machines emulators, we're supremely grateful that these experiences aren't being lost to posterity. Which is why we're highlighting this release, the first major update for a year.

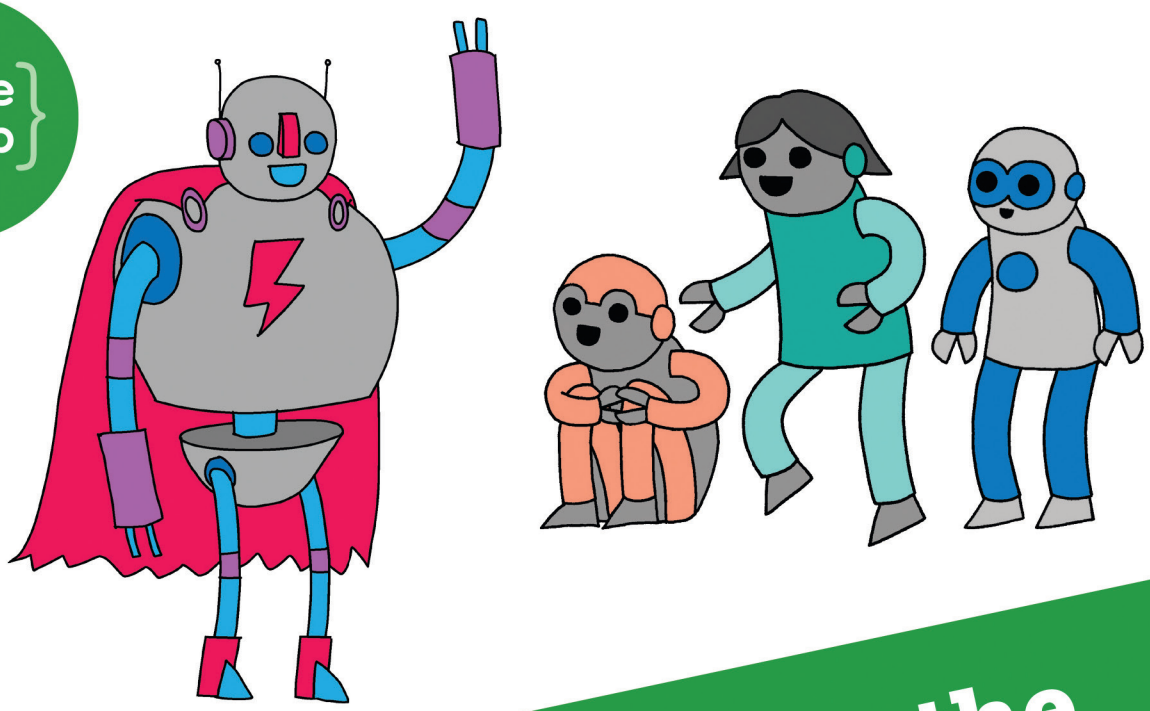
There's a graphical overhaul to the subsystems used to render games and support for 10 new titles, including '*The Lost Files of Sherlock Holmes*,' '*Broken Sword 2.5*' and two of the last *Zork* games. But of course, it still works brilliantly playing '*Maniac Mansion*,' a game which our friend Richard Cobbett once described as 'one of the most intricate and important adventure games ever made.'



The very latest version of *ScummVM* adds support for Myst. Now if only the game itself could be updated for virtual reality.

Project website
http://scummvm.org

code club



Can you help inspire the next generation of coders?

Code Club is a nationwide network of volunteer-led after school clubs for children aged 9-11.

We're always looking for people with coding skills to volunteer to run a club at their local primary school, library or community centre for an hour a week.

You can team up with colleagues, a teacher will be there to support you and we provide all the materials you'll need to help get children excited about digital making.

There are loads of ways to get involved!

So to find out more, join us at **www.codeclub.org.uk**

TUTORIALS

Warning: excessive Linux knowledge may lead to fun and more efficient computing.

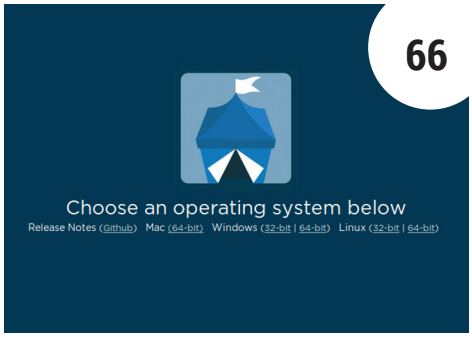
Ben Everard
Looks deep into the tea leaves and sees an open source future.

Every one of the world's major computing companies produces open source code, as do almost all the world's major governments. Almost every company uses FOSS to some degree, and soon, every major web browser will be open source. While people who use exclusively Free Software are still rare, almost every person uses it so some degree, whether consciously or not.

There wasn't some big moment that suddenly tipped the scales in favour of open source. The situation we have today is the result of hundreds of thousands of small decisions. Every time anyone makes the decision to use FOSS rather than some proprietary software, they tip the balance ever so slightly in the right direction. Every line of open source code written and every bug report filed has added to the cause and collectively, all these decisions have changed the world. As long as people keep choosing open source, and keep supporting the cause, the world will keep changing for the better. Victory is in sight – we just need to keep doing what we're doing and the future will be a freer place.

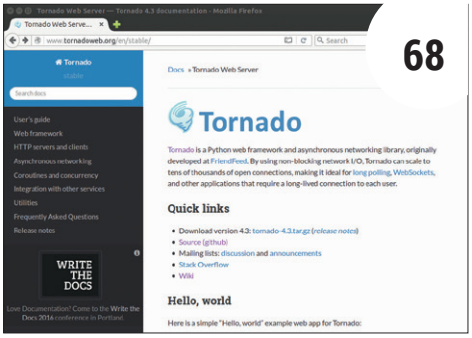
ben@linuxvoice.com

In this issue . . .



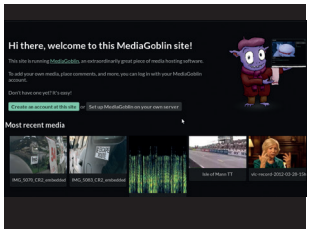
Decentralised online shopping

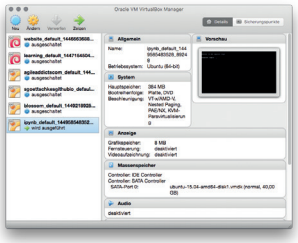
Peer-to-peer internet shopping promises trustworthy sites without centralised control. **Ben Everard** grabs his bag and goes browsing.



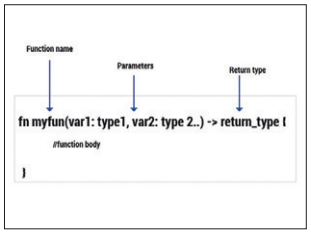
Personalised photo sharing for your events

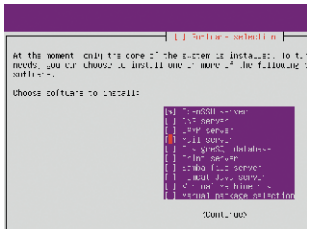
Building a photo sharing web app doesn't have to be complicated. **Ben Everard** starts a series on mixing Python with Bootstrap.





Coding





DECENTRALISED SHOPPING WITH OPENBAZAAR

Convert your Bitcoin stash into a collection of stuffed toys without the middle-man.

BEN EVERARD

WHY DO THIS?

- Buy stuff without lining PayPal's pockets.
- Make money.
- Keep the wheels of commerce turning.

Websites such as Amazon and Ebay enable small businesses to set up shops and sell through their channels. Because we can trust the large company not to lie to us, we can put some trust in the user ratings and reviews of these small businesses in a way we wouldn't be able to if they were on the business's own website.

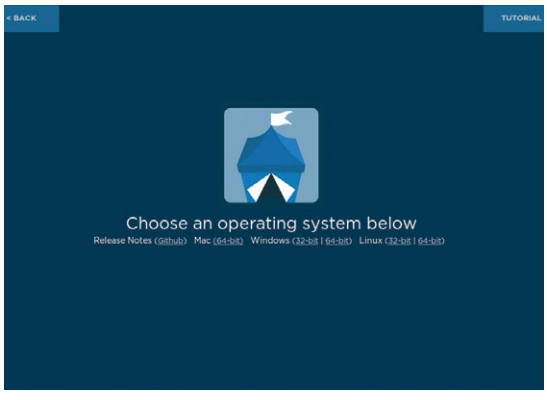
However, by allowing a company to control the ecosystem, we allow them to decide what can be sold (and what can't be), and to take a chunk of the money for themselves.

OpenBazaar is a decentralised online marketplace that enables us to find small shops and buy things without any centralised control. At the same time, if we trust the network, we can trust the ratings of a seller and avoid any malicious parties. In theory at least, this gives us the benefits of a centralised system without the control. Now let's buy some tat!

STEP BY STEP: BUY AND SELL

1 Install the software

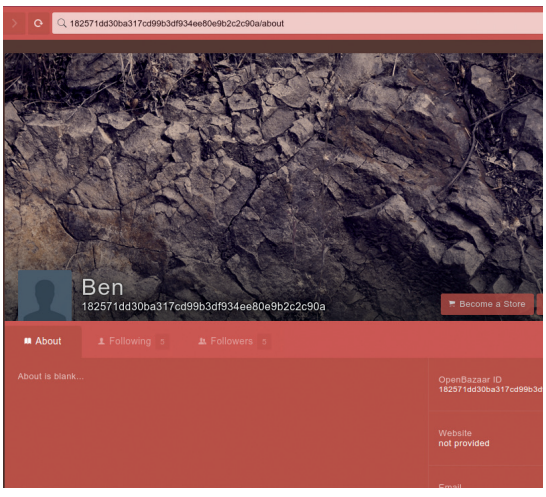
The first step, as always, is to grab the software. If you're using Ubuntu or any other Debian derivative, you can just download the Deb files from **https://openbazaar.org/download.html**. By the time you read this, there might also be install files for other distros there, so it's worth checking even if you're not on a Debian derivative. If aren't, it's going to be more difficult to set up. The easiest option here is to create a Debian or Ubuntu virtual machine and install *OpenBazaar* in there. If you want to build from source, you'll need two parts, the server (written in Python) and the client (running on Node.JS). Grab the server from **https://github.com/OpenBazaar/OpenBazaar-Server** and the client from **https://github.com/OpenBazaar/OpenBazaar-Client**. Both of these repositories include build instructions, but neither appears to have been well tested on any distro other than Ubuntu. Hopefully this will change as the project becomes more mature.



2 Create a profile

When you first go online, you'll need to create a profile. This can contain a lot or a little information about you. The very least it needs is your language and your timezone, but you can also build up a full profile including picture and biography if you want to. OpenBazaar is almost a combination of shopping platform and social network, so how much information you choose to put on will vary depending on how you intend to use the site.

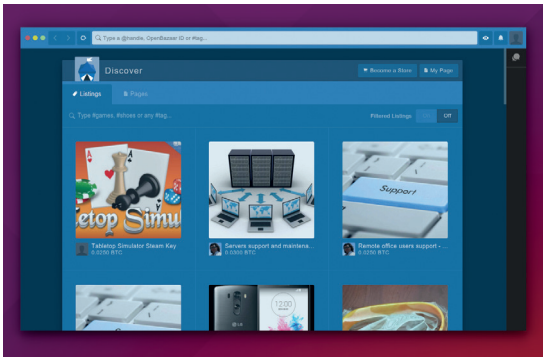
All the details of your profile are stored in the **.openbazaar** folder in your home directory. *OpenBazaar* is decentralised, and there's no one to help you if you lose this, so it's a good idea to keep this file backed up. If you want more than one account, just create a new user on your Linux system and you'll get a new user in *OpenBazaar* as well.



3 Find goods and services

OpenBazaar is structured with a lot of small stores, but you can also search all the products from a single search bar. Click on the eye in the top-right corner to enter Discover mode. Here you can type in whatever you want to find. By default, *OpenBazaar* only shows results from shops that you follow, and when you first start, this won't be anyone. Change Filtered Listings to Off and you can see results from all the stores. There aren't many now, but the number is increasing quickly.

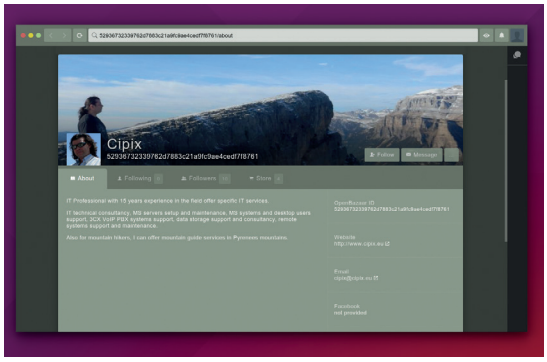
While *OpenBazaar* is decentralised, it's not anonymous, so while there are items of varying legality for sale, you are exposing your IP address if you buy them. Everything's charged in Bitcoin at the moment, so there's no problem with currencies around the world. If you put an address in your profile, you'll see shipping details for each product.



4 Research a seller

Shops on the internet trade on their reputation. If you see a store on the web, or on *OpenBazaar*, you need to know you can trust them before parting with any money. In time, sellers will be able to build up a reputation on *OpenBazaar*, and you can see reviews of products in the items page. However, in the early days, it will be hard to know who to trust. Do a little research to see what other people are saying about them.

The safest bet will be companies that exist outside of *OpenBazaar* and already have a reputation, so check the profile of anyone you're thinking of buying from to see if they link to a website (and make sure that the website also links back to the *OpenBazaar* page as there's no security on this). If you're still unsure about someone, you can use a trusted third party to moderate the transaction (See next step).

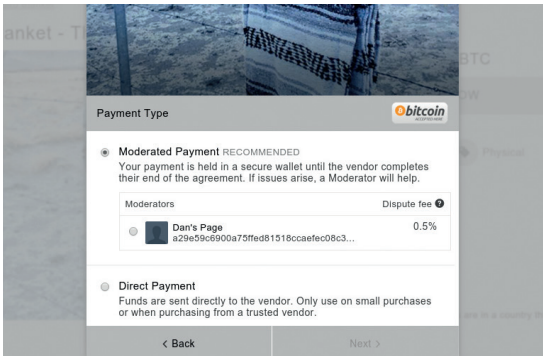


5 Buy something

Once you've found something you want to buy, the final step is actually purchasing it. There are no shopping carts, only the Buy Now button. First, you'll be asked if you have a Bitcoin wallet, and since all payments are in Bitcoin, it's essential that you get one before you can complete the transaction. The second step is to choose either a direct or moderated payment. If it's direct, the money goes directly to the seller; if it's moderated, the money is first held by a trusted third party.

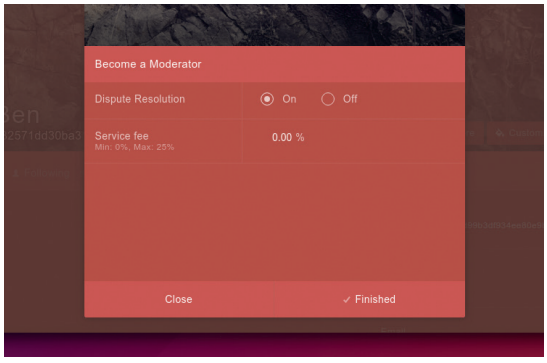
Anyone can set themselves up as a moderator, so make sure that you find one you trust before using this payment option. They each charge different fees to resolve disputes, so there's also a secondary market for trusted moderators that are cost efficient.

Finally, you just have to enter your payment details and shipping information.



6 Set up a store.

With *OpenBazaar*, anyone can set up a store, and that includes you! There are no setup fees (or, indeed any fees at all except optional moderator fees). Go to your profile (click the avatar icon in the top-right of the screen), and press Become A Store. You'll need to enter your store name and a description of yourself, then click Next. The final step is to select which moderators (if any) you want to trust. Once you're a store, you just need to enter the details of the items you're selling. Another option for the business-minded is to become a moderator. These are the people responsible for solving and disputes that arise between buyers and sellers. Click on your avatar and select Become a Moderator. The only information you need is the fee you'll charge to solve any problems. Once this is set up, sellers will be able to add you as an option for buyers.



BUILD A PHOTO WEB APP WITH TORNADO

Share photos without sending all the data to an advertising company.

BEN EVERARD

WHY DO THIS?

- Share your images with family and friends…
- …without sharing them with Mark Zuckerberg.
- Amaze your in-laws!

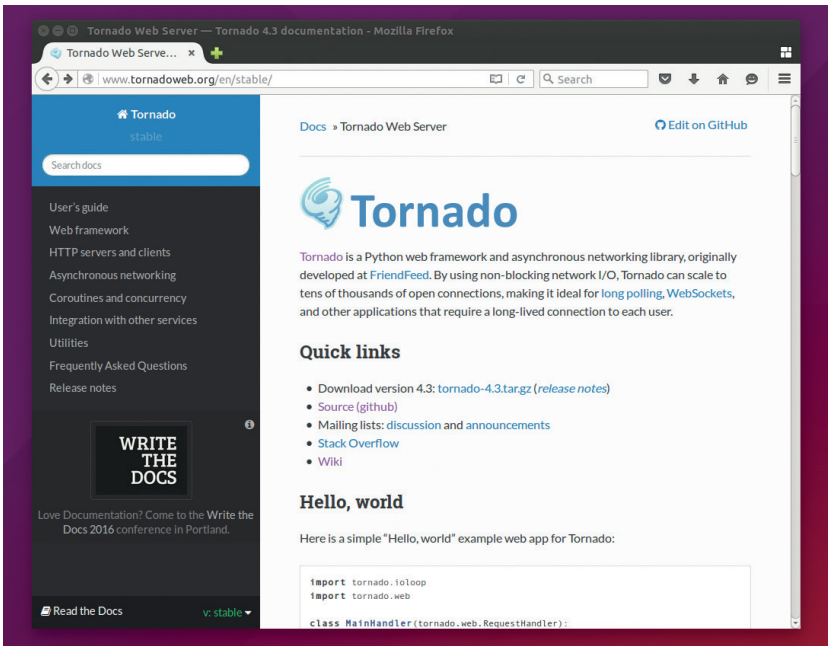
This summer I'm getting married! Naturally, we want guests to be able to share their photos of the day with us, and each other. There are proprietary options for this – such as Facebook – and there are ready-made open source options such as MediaGoblin (see page 76 for more on this excellent media platform). However, we wanted a web app personalised to us that enabled people to share their images as well as giving useful information about the day. The solution was obvious: we'd write our own.

Python is a great choice for this task, because it helps us develop quickly and easily. We don't want to worry about the nitty gritty, and with Python, we don't have to. The *Tornado* web framework will provide the HTTP server, and give us the tools to build our web app with just a few templates.

This is a three-part tutorial. In this first part, we'll look at the main web app that enables people to browse and upload pictures. In part two we'll look at making the pages look better, and in part three we'll build a photobooth that will push pictures straight to the web app from a camera with the help of a Wi-Fi-enabled SD card.

First, let's install *Tornado*. The easiest option is to grab it via *pip*, the Python installer.

Friendfeed (owned by Facebook) developed Tornado in 2009. Yes, we are aware of the irony of using this social media technology to avoid social media.




```
sudo pip install tornado
```

Tornado is an HTTP server and web app framework in one, so we don't need any additional software (as we would with, say, PHP), and this also means that everything is controlled from a single place.

A web app in *Tornado* consists of one or more classes, and each class is bound to a web address. When a browser requests that address, *Tornado* creates an object of the corresponding class, and this object serves the page. This method enables *Tornado* to create multiple objects to handle requests. A really simple website could be served by *Tornado* with the following code:

```
import tornado.ioloop
import tornado.web
class MainHandler(tornado.web.RequestHandler):
        def get(self):
        self.write("<h1>Hello world</h1>")
def make_app():
        return tornado.web.Application([
        (r"/", MainHandler),
        ])
if __name__ == "__main__":
        app = make_app()
        app.listen(8888)
        tornado.ioloop.IOLoop.current().start()
```

The **make_app** function creates a **tornado.web.Application** object with a list containing all the different web addresses we want our app to serve, and the classes that will be used to serve them. In this example, we only want one address (**/**), and it's served by the class **MainHandler**. The **MainHandler** class inherits most of the functionality it needs from the **tornado.web.RequestHandler** class, and the only thing we need to define are methods that correspond to the HTTP verbs we want that address to respond to. For normal web pages, this will just be **get**, which is called when a web browser requests a page, but we'll also use **post**, which is used to handle data sent to the web server through forms in the web page.

As you can see, this allows us to use Python to put together the web page we want to return. In this case, it just returns Hello World when you visit the root of the website. Start the server running with **python website.py**, then point your browser to **localhost:8888** to see the result.

If we just wanted to render the same website every time someone visited our site, we could just use a normal web server and not have to bother with Python. The power of *Tornado* (and other frameworks) is that it enables us to tailor the site depending on what the visitor does. We'll do this by passing arguments between the web browser and the web server (arguments are part of the URL that come after a question mark). If we want to expand our simple hello world example to greet visitors by name, we just need to change the **MainHandler** class to the following:

```
class MainHandler(tornado.web.RequestHandler):
        def get(self):
        name=self.get_argument("name","world")
        self.write("<h1>Hello "+name+"</h1>")
```

The **get_argument** method takes two parameters: the first is the name of the argument and the second is the default value. This will return a string containing the text from the URL. If you make this change, then restart the Python program, you can get a personal greeting from your web browser by going to **localhost:8888/?name=Ben**.

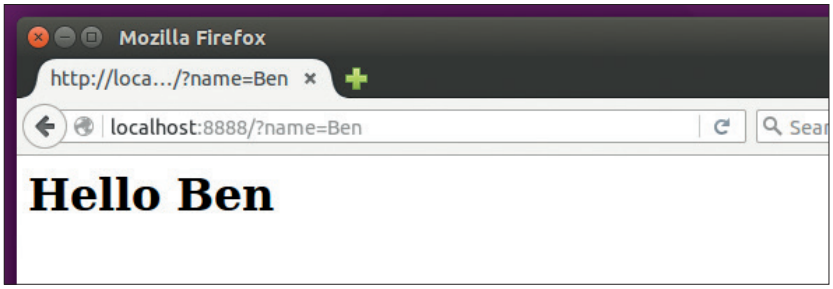
This is the basic mechanics of *Tornado* – we create a class to craft a web page for the visitor based on the arguments supplied in the URL. There is just one more thing to make the code a little cleaner. Most web pages are quite long, and often have well over a hundred lines of HTML. We don't want to include all that in our Python code, so we use templates. Templates are a way of combining HTML and Python in a way that's easy to maintain.

Templates are basically normal HTML files, but with bits of Python code interspersed. If you just want to insert a value from a variable, you can put the name of the variable inside double braces. The following example would give us the same result as the previous code:

```
<h1>Hello {{name}}</h1>
```

This same syntax can be used to put the result of any Python expression directly into the website.

Additionally, you can put more advanced bits of code such as loops inside **{% %}** blocks. Since HTML doesn't follow the same indenting as Python, you need to explicitly finish with **{%end%}**. Let's now look at how we want our web app to work. You'll see these work a little later.



URL parameters are the link between the browser and the server, and you can use them to create a personalised web page.

There will be three main pages: the front page, the gallery page and the upload page. The first one we need to get working is the upload page, because until this works, we won't have any images with which to test the other pages.

The **UploadHandler** class consists of two parts. The first is the **get** method, which returns a web page with a form to upload the picture; and the second is the **post** method, which takes the uploaded picture, saves a copy of it and also makes a thumbnail.

> The power of Tornado is that it enables us to tailor the site depending on what the user does

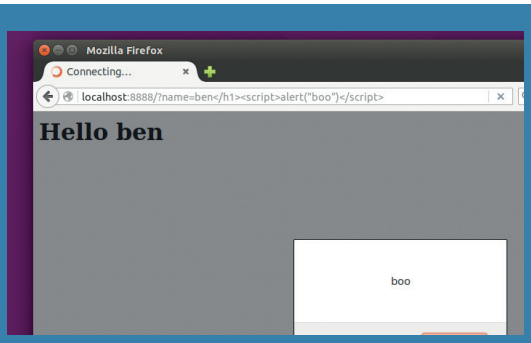
The code for this is:

```
class UploadHandler(tornado.web.RequestHandler):
  def get(self):
    self.render(web_root+"templates/upload.html", app_
title=app_title)
  def post(self):
    file_body = self.request.files['file1'][0]['body']
    fname = str(uuid.uuid4())
    img = Image.open(StringIO.StringIO(file_body))
    img.save(web_root+"images/"+fname+"."+img.format,
img.format)
    img.thumbnail((128,128))
    img.save(web_root+"thumbnails/"+fname+"."+img.
format, img.format)
```

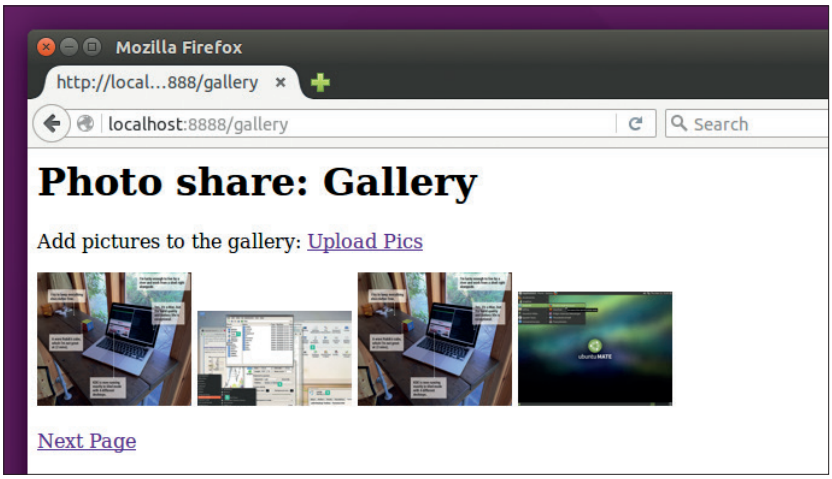
Security

Whenever you let a user send data to your web app, you need to be aware of the potential security risk. The first example, in which we took a user's name via a URL parameter, created a cross-site scripting vulnerability, because the user can send data that we include directly in the website, and this can include JavaScript. They could then use this to make your website appear however they wanted. If we put this on the public web, we'd need to add some checking before we passed the string on to the template.

File uploads can also present problems, because a hacker could upload a malicious file. In our program, we run the uploaded file through the Python Image Library which should filter out any malware.



Cross-site scripting vulnerabilities account for more reported vulnerabilities than any other type of attack.



It may not look good, but it works: our online photo sharing site. You'll need a public-facing web server to share this with the real world.

```
    self.redirect("/")
```

First let's look at the **get** method. This does no processing and just renders a template. If you want to use variables in templates, you have to pass them across from the main page, and in this case we pass **app_title** to a variable of the same name in the template.

The template stored in **templates/upload.html** is:

```
<h1>{{app_title}}</h1>
<form enctype="multipart/form-data" action="/upload"
method="post">
<input class="btn btn-lg btn-success" type="file"
name="file1" accept="image/*"/>
<br />
<input class="btn btn-lg btn-success" type="submit"
value="upload" />
</form>
```

For space reasons, we'll only include the body of the HTML templates we're using. These will render properly on most web browsers, but if you're using templates for real work, they should include the full HTML header as well.

We're storing all our data in the filesystem rather than a database, so at this point we need to define our directory structure to stop things getting lost.

> Our really simple photo sharing app is built and ready to go. It's simple, functional, and easy to use

Inside the main root directory, we'll have a **templates** folder, an **images** folder and a **thumbnails** folder. In *Tornado*, these don't have to correspond to the URLs that we use to serve the files (as these are defined in the **make_app** function), however, it can make it easier to see what's going on if the directories do match. For the image upload to work, you need to create the **images** and **thumbnails** folders.

We're using **uuid** to create a random name for the image. Used in this way, it's not completely guaranteed to be unique, but the chances of it generating the same name twice are tiny, so we're prepared to accept the small risk and save on the effort (and processing time) of checking to make sure this doesn't already exist.

The **self.requests.files** variable holds all the files that are uploaded by the user. In our template, we called the file-input-box **file1**, so this is the name we're looking for. All we need is the body of the first file uploaded in the box. You could put in more error checking at this point to respond sanely if there were a problem, but we'll omit this for brevity. The Python Image Library is then used to resize the file and save it to the appropriate places.

As well as this class, we need a couple of bits to make the upload work properly. First we need to import some new modules and set a global variable. All this can be done at the start of the file by adding the following lines below the import lines already there (we've put all the import and global variable lines here even though some won't be needed until later):

```
import Image
import StringIO
import uuid
import os
import math
app_title = "Photo share"
pics_per_page_gallery = 4
webroot = "/home/ben/weddingtutorial/"
```

Second, we need to change the **make_app** function to include the URL for the upload page.

```
def make_app():
        return tornado.web.Application([
        (r"/", MainHandler),
        (r"/upload", UploadHandler),
        ])
```

You can now restart the Python command running the website and go to **localhost:8888/upload** and upload an image. Once it's completed successfully, you'll be redirected to the main page (which should still display 'hello world'). To make sure that everything's gone successfully, make sure that the image has uploaded to both the **images** and **thumbnails** folders.

The next stage is building a gallery to display the uploaded images. Before doing this, upload about 10 images through the uploader to make sure there are some to display in the gallery. This time, let's look at the template for the gallery first:

```
<h1>{{app_title}}: Gallery</h1>
<p class="lead">Add pictures to the gallery: <a class="btn
btn-lg btn-success" href="/upload" role="button">Upload
Pics</a></p>
{% for pic in pics %}
<a href="/images/{{pic}}"><img src="/thumbnails/
{{pic}}"></a>
{% end %}
{% if page > 0 %}
<p><a href="/gallery?page={{page-1}}"
role="button">Previous Page</a></p>
```

```
{% end %}
{% if page < final_page %}
<p><a href="/gallery?page={{page+1}}" role="button">Next Page</a></p>
{% end %}
```

This needs four variables passed from the main code, **app_title**, **pics**, **page** and **final_page**. The first of these is simply used in the title, and we store it in a variable to enable us to easily change it across all pages of the web app.

The **pics** variable contains a list of all the filenames for the images to display. Images have the same names as their thumbnails, they're just stored in different directories, so the name of an item in **pics** refers to both.

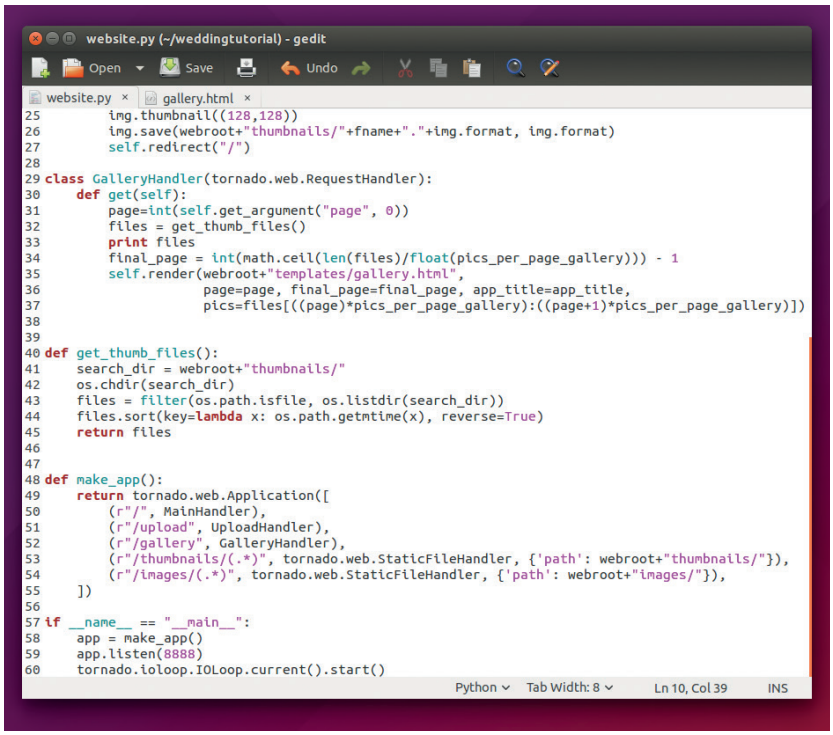
Tornado inserts the lines between {% for pic in pics %} and {% end %} once for every entry in the pics list. Therefore, this code block will create a thumbnail that links to the main image for every picture that we pass across in pics.

We could just send every uploaded image across, but this would mean the gallery web page quickly got very large. Not only would this make it hard to view, but it would increase the load on our server, because each thumbnail has to be sent. Instead, we'll only display a small number of images and enable the user to move forwards and backwards through the gallery. We'll do this using a URL parameter called **page**. To avoid the user scrolling out of the gallery, we need to hide the Next Page and Previous Page buttons when the user reaches the end of the gallery. This is done using the two **{% if … %}** blocks. Now, let's take a look at the code that makes all this work:

```
class GalleryHandler(tornado.web.RequestHandler):
  def get(self):
    page=int(self.get_argument("page", 0))
    files = get_thumb_files()
    final_page = int(math.ceil(len(files)/float(pics_per_page_gallery))) - 1
    self.render("templates/gallery.html",
          page=page, final_page=final_page, app_title=app_title,
          pics=files[((page)*pics_per_page_gallery):((page+1)*pics_per_page_gallery)])
```

This grabs the URL argument called **page** (defaulting to 0, the first page, if there is none). Then it calls a function called **get_thumb_files**. We'll create this below, but it gets a list of all the filenames of thumbnails with the most recently modified at position 0. The last bit of information needed is the total number of pages, which is calculated using the global variable **pics_per_page_gallery** (we set this to four to make testing easier, but will increase it before going live). The **pics** variable sent to the template is the appropriate section of the thumbnail files for the page the user is requesting.

Now, let's take a look at the **get_thumb_files** function. This links together a few features of the **os** module to find the files in the **thumbnails** directory. The sort uses a lambda function to extract the modified time for each file and perform a reverse sort based on this.



60 lines of Python is all it takes to share pictures without surrendering them to an advertising company.

In order to make this work, we need to add the appropriate lines to the **make_app** function:

```
def get_thumb_files():
  search_dir = "thumbnails/"
  os.chdir(search_dir)
  files = filter(os.path.isfile, os.listdir(search_dir))
  files.sort(key=lambda x: os.path.getmtime(x), reverse=True)
  return files
```

```
          (r"/gallery", GalleryHandler),
          (r"/thumbnails/(.*)", tornado.web.StaticFileHandler, {'path': webroot+"thumbnails/"}),
          (r"/images/(.*)", tornado.web.StaticFileHandler, {'path': webroot+"images/"}),
```

This sends the **/gallery** path to the **GalleryHandler**, but it also creates paths for **/thumbnails** and **/images**. In both of these cases, we just want to serve the images, so rather than create a new handler class, we can use the built-in **StaticFileHandler** class to serve up the files we've stored in the directory.

There we have our really simple photo sharing app built and ready to go. It's simple, functional, and easy to use. I showed it to my fiancée, and was told in no uncertain terms that if we were going to have a custom-written photo sharing website for our wedding, it'd have to look quite a bit better than this. Next month, then, we'll look at using Twitter's Bootstrap library to make things prettier.

Ben Everard is a sensitive troubador who also builds robots, writes books (including the best-selling learn Python with Raspberry Pi) and brews his own cider.

MASH UP MINECRAFT WITH SONIC PI

Combine Sonic Pi and Minecraft to build a world of music!

LES POUNDER

WHY DO THIS?

- Learn Sonic Pi.
- Learn Ruby.
- Use lists to organise data.
- Use tuples to store values.
- Store data in variables.

TOOLS REQUIRED

- Any model Raspberry Pi running the latest Raspbian release.
- All of the code for this project can be found at **https://github.com/lesp/Linux-Voice-27-Minecraft-Mashup/archive/master.zip**.

Minecraft is addictive – there, we said it. You have a world that can be shaped and changed using blocks which are one metre cubed. You can build a house, a bridge or a fire-breathing dragon! Anything is possible with a little patience and planning.

Sonic Pi is the live coding music phenomenon that has gripped coders of all ages and musical abilities. With Sonic Pi you can write any form of music using a simple-to-understand language based upon Ruby.

These two fantastic projects have one thing in common: they can both make computer science interesting to children and adults.

In a series of four tutorials we shall use both of these projects to create and shape *Minecraft* worlds and produce music, jingles and noise that interacts with the code we write.

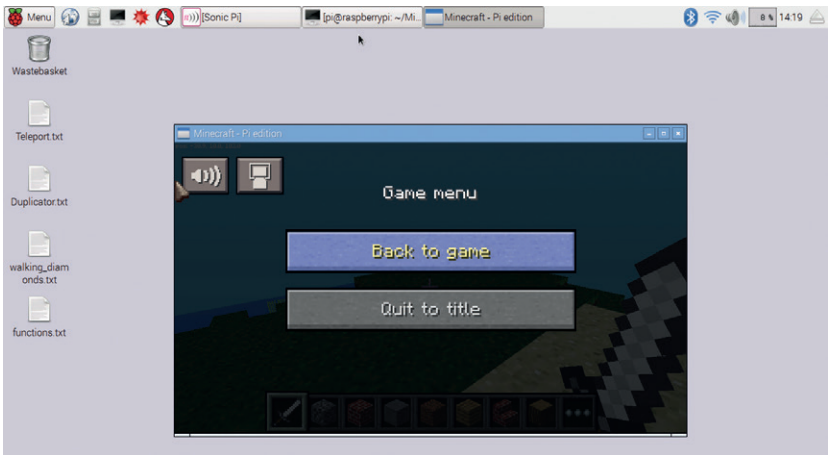
Over the course of the tutorials we will be introduced to coding concepts such as

- **Loops** To repeat the code sequences.
- **Variables** To store information such as individual co-ordinates.
- **Tuples** To store comma separated values such as x,y,z co-ordinates.
- **Functions** Used to group a sequence of code and recall it using a name.
- **Floats** Numbers that have a decimal place.
- **Integers** Numbers that have no decimal place.
- **Strings** Characters of text, including numbers that can be displayed on screen.

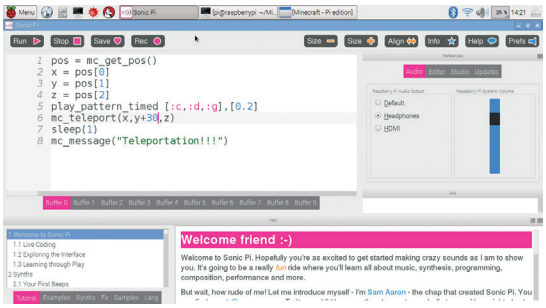
Sonic Pi is also a really clever way to teach musical composition. Musically minded children can use Sonic Pi along with music theory to compose pieces in any style of music. The fact that Sonic Pi uses MIDI (Musical Instrument Digital Interface) means that a learner can transfer that knowledge to and from the application.

Minecraft is also rather sneaky. 3D co-ordinates are not an easy subject to grasp in class but by using *Minecraft* as a delivery method kids can latch on and understand this confusing subject.

So let's get hacking!



When you first use *Minecraft* you will see the world through the player's eyes. To change this press Escape and click on the 3rd person button.



Unlike some code editors, Sonic Pi has a clean interface that enables users to concentrate on their work. It is highly configurable and very easy to learn.

Project 1 – Teleport Jingle

Teleportation is no longer the preserve of *Star Trek*; in *Minecraft* we can also send our avatar to anywhere in the world, instantly. This simple project serves as an introduction to using Sonic Pi with *Minecraft*. Before we start you will need Sonic Pi open and *Minecraft* should be open and your player in a world.

In Sonic Pi use any empty buffer. We shall start by finding out our position in the world. To do this we use the **mc_get_pos** function, which will return three values (our x,y and z co-ordinates) which tell us where we are in the world. We shall save these co-ordinates to a tuple, a list of comma separated values, called **pos**.

```
pos = mc_get_pos()
```

Now we need to split the values in **pos** into separate x,y,z variables, as it makes them easier to work with. We use slicing to precisely remove the values that we need from the tuple.

```
x = pos[0]
```

```
y = pos[1]
z = pos[2]
```

With the co-ordinates split into their corresponding axes we shall now play a little jingle that will indicate that transport is under way. The jingle is comprised of three notes played in rapid succession, with 0.2 seconds between each note. We play the notes C, D and G as they are complimentary to each other.

```
play_pattern_timed [:c,:d,:g],[0.2]
```

Now we shall use the co-ordinates that we learnt earlier. We shall keep the x and z co-ordinates as is. But the y co-ordinate we shall change by 30 blocks, roughly 30 metres, so that our avatar is teleported into the air, ready to fall back to the ground.

```
mc_teleport(x,y+30,z)
```

Our last two lines for this project create a delay of one second before printing a message to the *Minecraft* chat window.

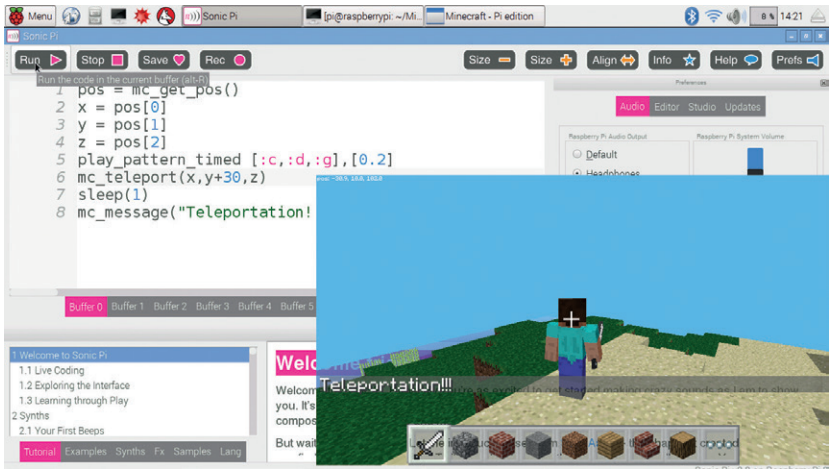
```
sleep(1)
mc_message("Teleportation!!!")
```

With the code complete, click on Run in the top-left of the Sonic Pi window. Now quickly switch to *Minecraft* to see the poor little character fall to the ground with a little jingle.

Project 2 – The world at your feet

Have you ever wanted to have a *Minecraft* block follow you around, but above your head? Well in this project we shall do just that. For this project it would be prudent to use another buffer in Sonic Pi.

Our goal for this project is to replicate the block that we are standing on, but have it hover above our head. This shows that we can detect the type of block at our feet, so we can "get" the block type and then "set" the block above us.



Teleportation is the "Hello World" of Minecraft hacks. It offers something more interesting that simply printing text. How high can our little man fly?

We start by using a **live_loop**, which is a way of running multiple loops in Sonic Pi. Each of these loops can be run simultaneously. A **live_loop** needs a name, and we called our first loop "duplicator" as it handles duplicating the blocks.

```
live_loop :duplicator do
```

Just as in project 1, we shall use a tuple to store the location of our player before saving each of the co-ordinates as a separate variable.

```
pos = mc_get_tile
x = pos[0]
y = pos[1]
z = pos[2]
```

So now that we know where we are, let's create a variable that will store the block type we are standing on, which is 1 block beneath us; in other words "y-1", so to find out the block type.

```
below = mc_get_block(x,y-1,z)
```

So now that we know what the block beneath our feet is, let's change the block that is above our head to match. Our player is two blocks tall, so to give us a little space let's have the block appear one block above us. To do this we "set" the block and pass it four arguments: the type of block and our three co-ordinates in the world, but with our y co-ordinate altered by three blocks. We then add a short sleep to pace the speed of the project. Finally for this section we close the loop using **end**:

```
mc_set_block(below ,x,y+3,z)
sleep 0.1
end
```

For our next **live_loop** we will create a simple melody to play as we walk around the world. Again we name the loop, this time we use the name **beat**.

```
live_loop :beat do
```

For our beat we shall use one of the many instruments built in to Sonic Pi. In this case the **blade** instrument replicates the violin drone heard in the *Blade Runner* movie theme.

```
use_synth :blade
```

Now that we have the instrument, lets write some

What is Sonic Pi?

Sonic Pi is the creation of Dr Sam Aaron from the University of Cambridge Computer Laboratory. Sam is a gifted coder and musician who regularly plays to audiences across the world. Sam teamed up with the Raspberry Pi Foundation's Education Pioneer, Carrie Anne Philbin, to produce an application that was easy enough for children with no coding experience to use, while remaining flexible for professional musicians to use.

Sam's knowledge of code and music enabled him to add features that benefited all users. One area that is particularly hard for children to grasp is indentation, a vital method to learn for languages such as Python, and so Sonic Pi has an alignment tool that will scan your code and correct your alignment. Sam has also written a 30,000-word book that is located in the help system, enabling quick reference to coding concepts, language syntax and a plethora of example scripts that can be copied and pasted into a blank buffer for a quick hit of inspiration.

But the biggest feature of Sonic Pi is the live coding element. Live coding is where you use the computer as an instrument – rather than manipulate an instrument, you manipulate code in real time. In our projects we used the **live_loop** to have code repeat inside of a loop, and any changes made to the code inside the loop can be instantly updated by clicking on Run during the playback cycle. This is great fun for instant feedback on your composition.

Duplicating blocks above our head is a great way to understand how to detect the block that you are standing on and then change the block above your head.

music. We are going to use the **play_pattern_timed** function and repeat the C,D,G pattern from before, but now we shall alter the timings for each note, giving us a subtle change to the music.

```
play_pattern_timed [:c,:d,:g],[0.3,0.1,0.1]
```

Finally for this project we use a sleep to delay the pace of the code and then end the **beat** loop.

```
sleep 0.5
end
```

With the code complete, click on Run to launch the code and you should hear the beat play. Switch to *Minecraft* and go for a walk; you will see blocks above your head. If you have a desert in your world, make sure you walk very quickly.

Project 3 – Walking through the air

The goal for this project is to have a block appear at our feet, enabling us to walk through the air and never get our feet wet.

To start, use a new buffer. We are going to reuse the **live_loop** structures from Project 2, but this time we shall call the loop **diamond** as we shall be walking on a bridge made of diamonds.

> Four projects in Minecraft and we've covered quite a few coding concepts with very little code

```
live_loop :diamond do
```

We will now get the tile position of the player. This is a very coarse value when compared to **get_pos**, which provides precise values using floats. **get_tile** returns three integers that are rounded to the nearest tile position.

```
pos = mc_get_tile
```

Next we split the values stored in the **pos** tuple so that we have individual values.

```
x = pos[0]
y = pos[1]
z = pos[2]
```

Next we create a variable called **below** that will store the block under our feet.

```
below = mc_get_block(x,y-1,z)
```

For the last section of this loop we set the block under our feet to be a diamond block – yes, we can have diamonds fall at our feet! We will then use a sleep to delay the code before closing the loop.

```
mc_set_block(:diamond_block ,x,y-1,z)
sleep 0.1
end
```

For our final loop in the project we reuse the **beat** loop from Project 2 but change the synth instrument to **fm**, a suitably synthetic noise. We also changed the notes played in the pattern to give a different sound when walking in the air.

```
live_loop :beat do
use_synth :fm
play_pattern_timed [:a,:e,:g],[0.3,0.1,0.1]
sleep 0.5
end
```

And with the code complete, click Run and the switch back to *Minecraft*. Go for a walk, soar into the sky by double-tapping the Space bar. To go higher, press and hold the Space bar; to go lower, press the Shift key while moving forward.

Project 4 – Walking randomly

For our fourth and final project we will go a little further and use random choice to change aspects of our code. The goal of this project is to play random notes as we fly through the air on a path of diamonds or wood.

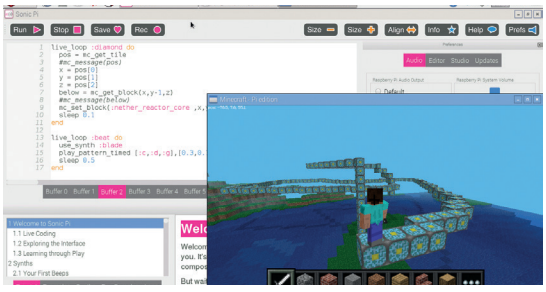
Use a new buffer in Sonic Pi. We start by upping the tempo of our music. We will now be playing our beats at 120 beats per minute (BPM).

```
use_bpm(120)
```

We are now going to do something new – we're going to create a series of functions, groups of code that can be run just by calling their name. Our first function is called **bell**, and we start by defining its name

```
def bell;
```

In the previous project we used instruments to inject a little noise into our code. But Sonic Pi also comes with a series of samples – pre-recorded audio files – that can be inserted into our project. In this function we shall use an electronic bell sample. We can control the rate of playback for a sample, with 1 being normal speed, 2 double speed and -1 being



Building a bridge through the air is a fun project that shows how easy it is to automatically build anything in *Minecraft*.

Block types

Minecraft is an ordered world and has some similarities to our own planet. First we have gravity, an attraction between two bodies. In the instance of project 1, our player is attracted to the world and will fall towards it. The world is made up of a bedrock mantle with grass, deserts and bodies of water upon the crust of the world. Water flows following the path of least resistance, as does lava, with both being capable of causing great destruction to a world.

There are many different types of blocks that can be used to build with; **stone_brick**, **wood_planks** and **glass** can be used to build a home. There are even special blocks to add that personal touch to a home such as **bed**, **door_wood** or **door_iron**. Rare blocks such as **Nether_Reactor_Core** and **Obsidian** are typically mined at great expense, but with a little Sonic Pi code we can generate many instances of these blocks on a whim.

Blocks such as **sand**, **water_flowing** and **lava_flowing** can be quite troublesome to use as they obey the law of gravity, meaning that if they are placed high up in a world, they will inevitably flow around the world. Try using these blocks in project 2, duplicator, to see how powerful they can be.

This is your *Minecraft* world and while it may obey certain laws of science, using Sonic Pi and Ruby we can change our perception of this world, bending and shaping it to our will. How will you wield that power?

reverse playback at normal speed. It would be cool to set the rate of playback, but perhaps we should use a little random choice to mix it up? The rates to be used are stored in a list (identified by **[ ]**). By adding **.choose** to the end of the command we can let the Raspberry Pi choose the rate of playback.

```
sample :elec_bell, rate: [-1,1.5,0.5].choose
```

Finally we add a delay to our code before ending the function.

```
 sleep 0.1
end
```

For our second function we use the **fm** synth to play a randomly chosen note from a list of D5, C5 or G5. We then sleep for a brief time before ending the function.

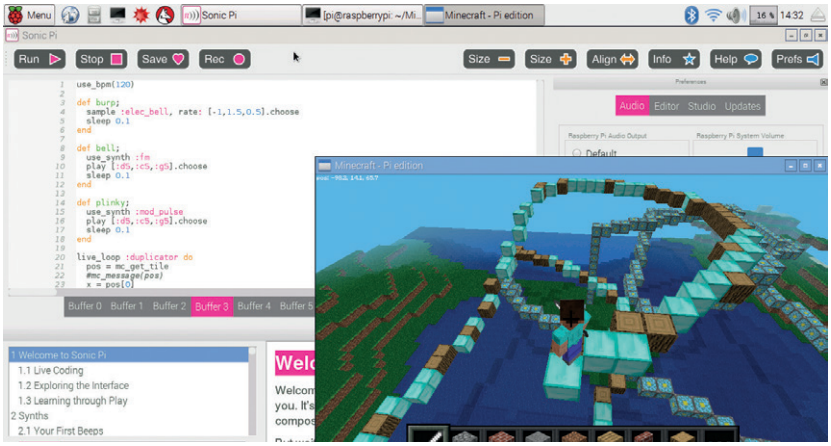
```
def fm;
 use_synth :fm
 play [:d5,:c5,:g5].choose
 sleep 0.1
end
```

For our third function we replace the single-note instrument with a modulating note; again, we pick a note at random and play it for only a tenth of a second before the function ends.

```
def plinky;
 use_synth :mod_pulse
 play [:d5,:c5,:g5].choose
 sleep 0.1
end
```

We reuse the duplicator code from Project 3 to create a path at our feet. Again the code is contained in a **live_loop** to constantly run the code, but enabling us to edit the code on the fly, if we so wish.

```
live_loop :duplicator do
 pos = mc_get_tile
```



After running project 4 for a while your world will be littered with lines of blocks – try changing the block type to **flowing_lava** for a little destruction!

```
 x = pos[0]
 y = pos[1]
 z = pos[2]
 below = mc_get_block(x,y-1,z)
```

So far the code is the same as Project 3, but now we add a new line, which will choose the block to be used as our pathway. In this project we only used two block types in our list, but your list can be as long as you wish. In fact adding more items to the list will yield more random results.

```
mc_set_block([:wood,:diamond_block,].choose ,x,y-1,z)
```

Lastly for this loop we control the pace with a sleep and then close the loop.

```
 sleep 0.2
end
```

We now create another **live_loop** that will control the music playback. In this loop we call each of the functions that we created earlier. Feel free to change the instruments, synths and samples used in the functions to make the music your own.

```
live_loop :beat do
 plinky
 bell
 fm
end
```

With the code complete, click on Run and you will hear lots of random noise, which sounds almost like an amusement arcade. Switch to *Minecraft* and go for a walk. The blocks that fall at your feet will be randomly chosen from the list of blocks that you entered in the loop.

So there we have it: four projects in *Minecraft* and we've covered quite a few coding concepts with very little code – oh, and we had some fun making lots of noise and hacking our own *Minecraft* world.

Use this code as a basis for experimentation, change the instruments and co-ordinates and see what happens.

Les Pounder divides his time between tinkering with hardware and travelling the United Kingdom training teachers in the new IT curriculum.

GNU MEDIAGOBLIN: YOUR MEDIA, YOURTUBE

Share and stream your media without losing control.

MAYANK SHARMA

WHY DO THIS?

- Learn a platform designed for sharing media.
- Supports a variety of popular formats and file types.
- Offers the convenience of popular media sharing platforms without relinquishing control.

Edward Snowden's NSA leaks saw a meteoric rise in the demand for and usage of privacy-first software. The *GNU MediaGoblin* project was able to capitalise on this and managed to fund itself through two successful crowdfunding campaigns. The project is spearheaded by two veteran Free Software advocates, Deb Nicholson and Chris Webber, who wanted a platform that enabled the discerning user to host, and more importantly share, their media without agreeing to the fine print on the popular centralised platforms.

GNU MediaGoblin (*GMG*) is written in Python and can be deployed easily on any Linux server. Once it's up and running, you can upload and share videos, images, audio, PDF documents and other types of digital media. One of the key features of *GMG* is its federation layer, which enables you to sync and share content across different *GMG* installations.

Everything you need to deploy *GMG* is available in your distro's official repositories. We'll be setting up the server atop an Ubuntu Server installation, but these instructions will work on any other distro with appropriate tweaks.

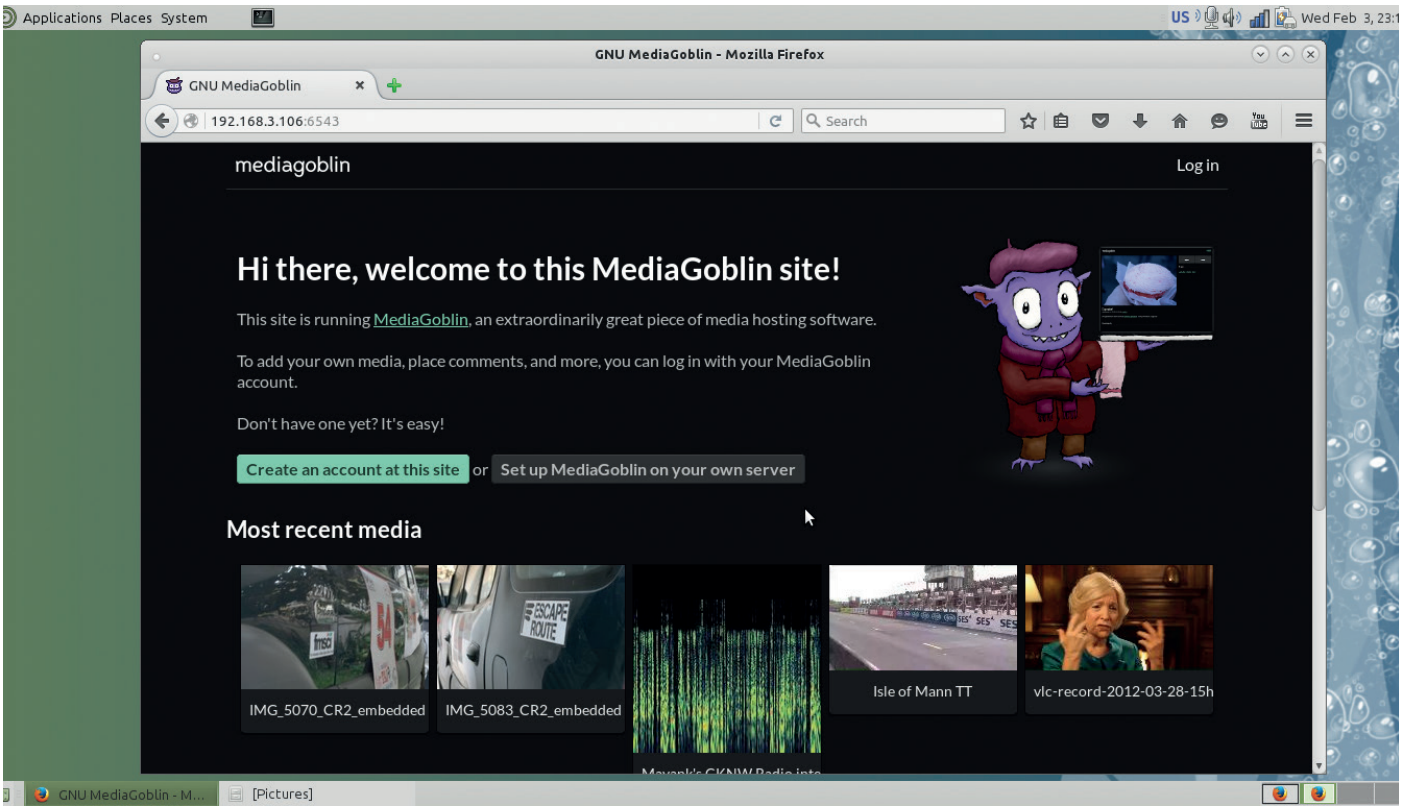
Begin by installing the dependencies with

```
sudo apt-get install git-core python python-dev
python-lxml python-imaging python-virtualenv npm
nodejs-legacy automake
```

Besides Python, we've installed the **python-lxml** package to enable Python to parse XML ,and the Python Imaging Library for processing images. There's also **git** for downloading a copy of *GMG*.

GMG houses its data inside a database and supports both *PostgreSQL* and *SQLite*. The latter is the default option and works well for smaller installations. If you are planning to have more than a couple of users uploading and viewing content, it's best to switch to *PostgreSQL*, which is what we'll use in this tutorial. You can install the database server with

```
sudo apt-get install postgresql postgresql-client
python-psycopg2
```



Users can create a *MediaGoblin* account to upload media or comment on existing ones.

The installation process will automatically create a new system user named **postgres** for managing the database. However, to keep things streamlined and secure, we'll create a new database user with restricted privileges, named **mediagoblin**, and a new database named **mediagoblinDB** owned by the new **mediagoblin** database user for our *MediaGoblin* instance.

First, switch to the **postgres** system user with

```
sudo su - postgres
```

and then create a new database user with

```
createuser -A -D mediagoblin
```

After bringing the user to life, create the database with

```
sudo -u postgres createdb -E UNICODE -O mediagoblin
mediagoblinDB
```

Next we'll create an unprivileged system user named **mediagoblin**, which we'll use for the sole purpose of running *MediaGoblin*. The user named **mediagoblin** can be underprivileged, because *MediaGoblin* doesn't need any privileges to run. Controlling the server via an underprivileged user also helps make the system secure. The command

```
sudo useradd -d /var/lib/mediagoblin -m -r -g www-data
mediagoblin
```

will create a user named **mediagoblin** and assign it to a group that is associated with the web server (**www-data**). This will ensure that the web server can read the media files (images, videos, etc) that users upload. No password will be assigned to this account, and you will not be able to log in as this user, but you can switch to it.

Next up, you'll need to create a working directory for *MediaGoblin*. This is where the Git repository will be downloaded. Create the directory and give it the right permissions with:

```
sudo mkdir -p /srv/mediagoblin.example.org
sudo chown -hR mediagoblin:www-data  /srv/
mediagoblin.example.org
```

We're now all set to clone *GMG*s online repository into this folder. First switch to the **mediagoblin** user we created earlier with

```
sudo su mediagoblin -s /bin/bash
```

and then change the directory to the working directory we just created with

```
cd /srv/mediagoblin.example.org
```

and mirror the latest stable release with

```
git clone git://git.savannah.gnu.org/mediagoblin.git -b
stable
```

This will create a directory named **mediagoblin**. Switch to this folder (**cd mediagoblin**) and then initialise the repository with

```
git submodule init
```

followed by

```
git submodule update
```

After it's done we'll deploy *GMG* via the bootstrap scripts, which make use of the **virtualenv** tool to create isolated Python environments. First make sure you're in the correct directory (**/srv/mediagoblin.example.org/mediagoblin**) before issuing the

```
./bootstrap.sh
```

command. When it's done type **./configure** and finally, compile all components with the **make** command. While you are here, create a directory named **user_uploads**, which is where we'll be storing all the uploaded media files with

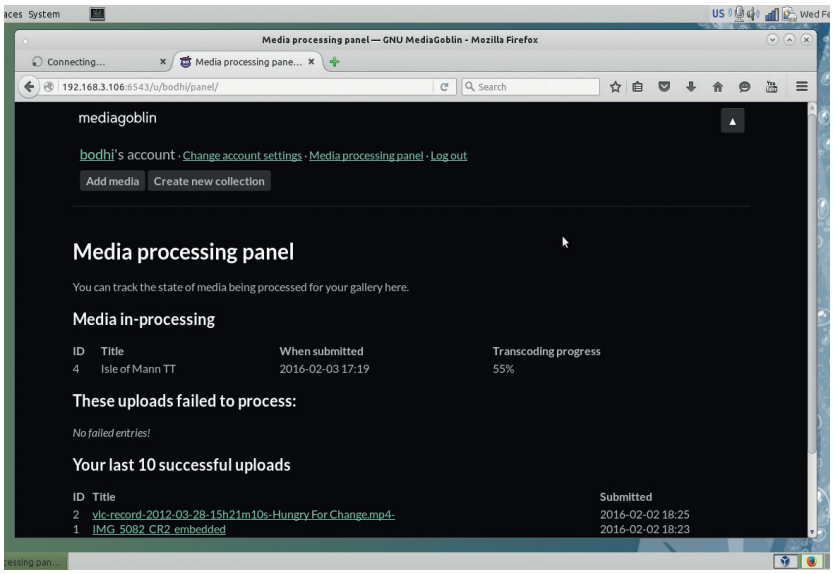
```
mkdir user_uploads
```

Also remember to make sure it has the proper permissions with

```
chmod 750 user_uploads
```

The *MediaGoblin* developers recommend using the FastCGI protocol to route requests from the web server to *MediaGoblin*. This is done via a Web Server Gateway Interface (or WSGI) and one of the most popular ones is the Python module named Flup. The latest version of Flup seems to only support Python 3 and newer versions, which is why we'll install an older version of the library with

```
./bin/easy_install
flup==1.0.3.dev-20110405
```



By default, *GNU MediaGoblin* will automatically transcode all videos to WebM format. Refer to the documentation to disable this behaviour and save time and processing power.

PRO TIP

If *Nginx* complains while installing, before anything else make sure you don't have another web server (such as *Apache*) running on port 80.

Get set GMG

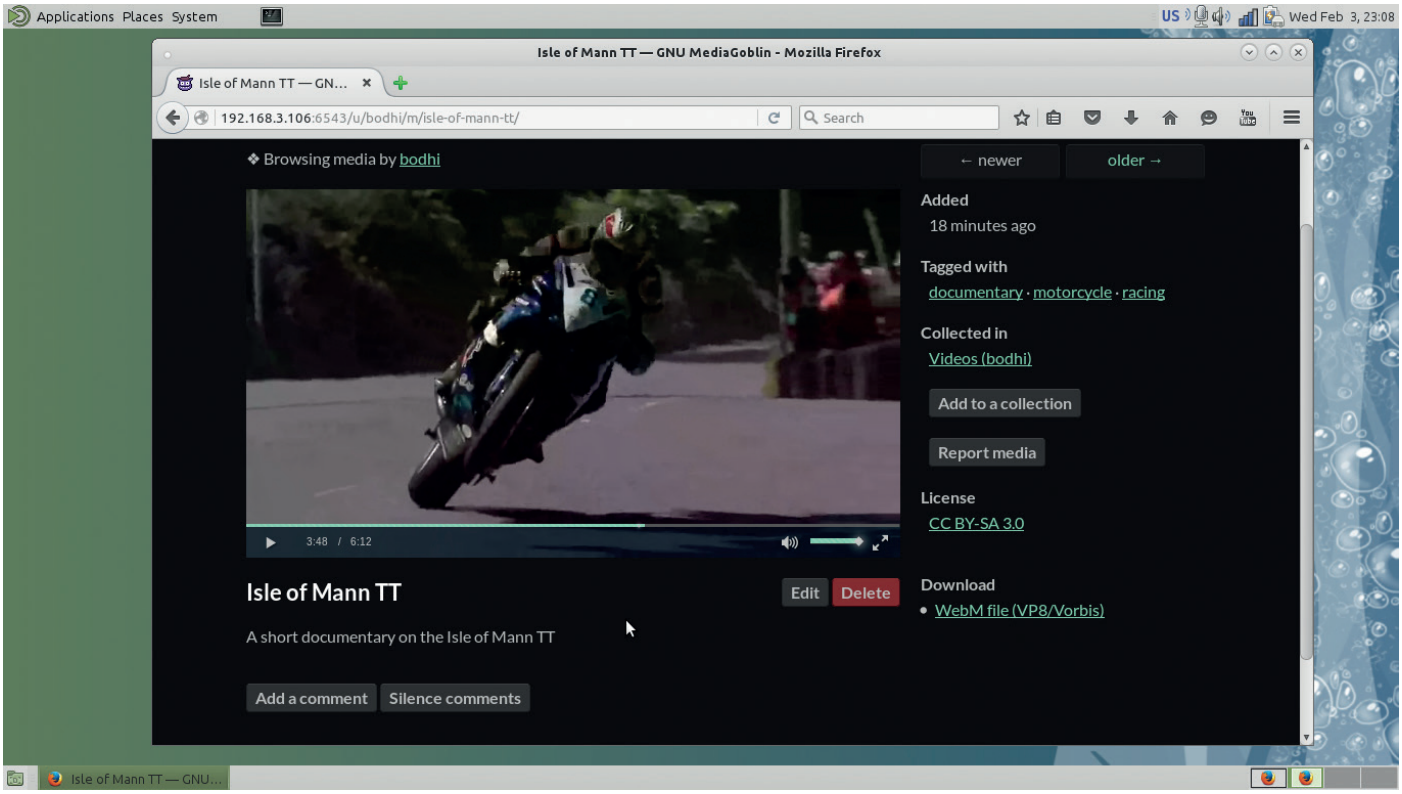
Now that *GMG* is installed, we will edit its configuration file as per our requirements. The main configuration file is named **mediagoblin.ini** and is housed under the **/srv/mediagoblin.example.org/mediagoblin** directory. Begin by making a copy of the file called **mediagoblin_local.ini** in the same directory. Then open the file in a text editor and scroll down to the **[mediagoblin]** section. Add the following line here to instruct *GMG* to use the *PostgreSQL* database we've set up for the server.

```
sql_engine = postgresql://mediagoblin@
localhost/mediagoblinDB
```

That's all for now. Save the file and exit the editor. Before proceeding further, populate the data with

PRO TIP

For easier management, turn the *MediaGoblin* deployment into a service with these init scripts (https://github.com/joar/mediagoblin-init-scripts).



You can browse media uploaded by other users and arrange them along with your own uploads into collections.

```
./bin/gmg dbupdate
```

That's it. Now bring the server online with

```
./lazyserver.sh --server-name=broadcast
```

You should now be able to connect to the *GMG* installation by pointing the browser to **localhost:6543**. You can also replace **localhost** with the IP address of the server to connect to the *GMG* server from other computers on the network. Advanced users can refer to the documentation on the project's website and hook up *GMG* with a webserver like *Nginx* or *Apache*.

Before you start using the *GMG* installation, you'll have to create an account. When you do so, the server will display the link to authenticate the account on the terminal. Click on it to verify your account. You can also manually add users from the command line by issuing the

```
./bin/gmg adduser
```

command from under the **/srv/mediagoblin.example.org/mediagoblin** directory.

PRO TIP

To minimise spam disable user registration by changing the allow_registeration = true line to allow_registeration = false in the mediagoblin_local.ini file.

Add media

You can now log into your *GMG* account and start adding media. Click the button labelled Add Media and use the Browse button to point to the file you wish to upload. Then define the title of the media, add a brief description and a comma-separated list of tags. Finally, use the pull-down menu to select the appropriate licence for this media file before you click the Add button to upload it.

By default, the *GMG* server only allows image uploads; support for other media types is bundled as plugins. Enabling them is a two-step process. First you'll have to install the required dependencies. For example, to upload audio files, install the necessary components with

```
sudo apt-get install python-gst-1.0 gstreamer1.0-plugins-{base,bad,good,ugly} gstreamer1.0-libav python-numpy python-scipy libsndfile1-dev libasound2-dev
```

Once the dependencies have been installed, open the **mediagoblin_local.ini** configuration file, scroll down to the **[plugins]** section and in a new line add **[[mediagoblin.media_types.audio]]**. Now save the file and apply the changes with the

```
./bin/gmg dbupdate
```

command. The audio plugin also needs a Python library to display spectrograms for the audio files. You can install it with

```
./bin/pip install scikits.audiolab
```

and then run the **dbupdate** command again.

Similarly, to enable support for video files, first install the dependencies, which are pretty the same as the ones for the audio plugin. Then add **[[mediagoblin.media_types.video]]** under the **[plugins]** section in the **mediagoblin_local.ini** file and run the

```
./bin/gmg dbupdate
```

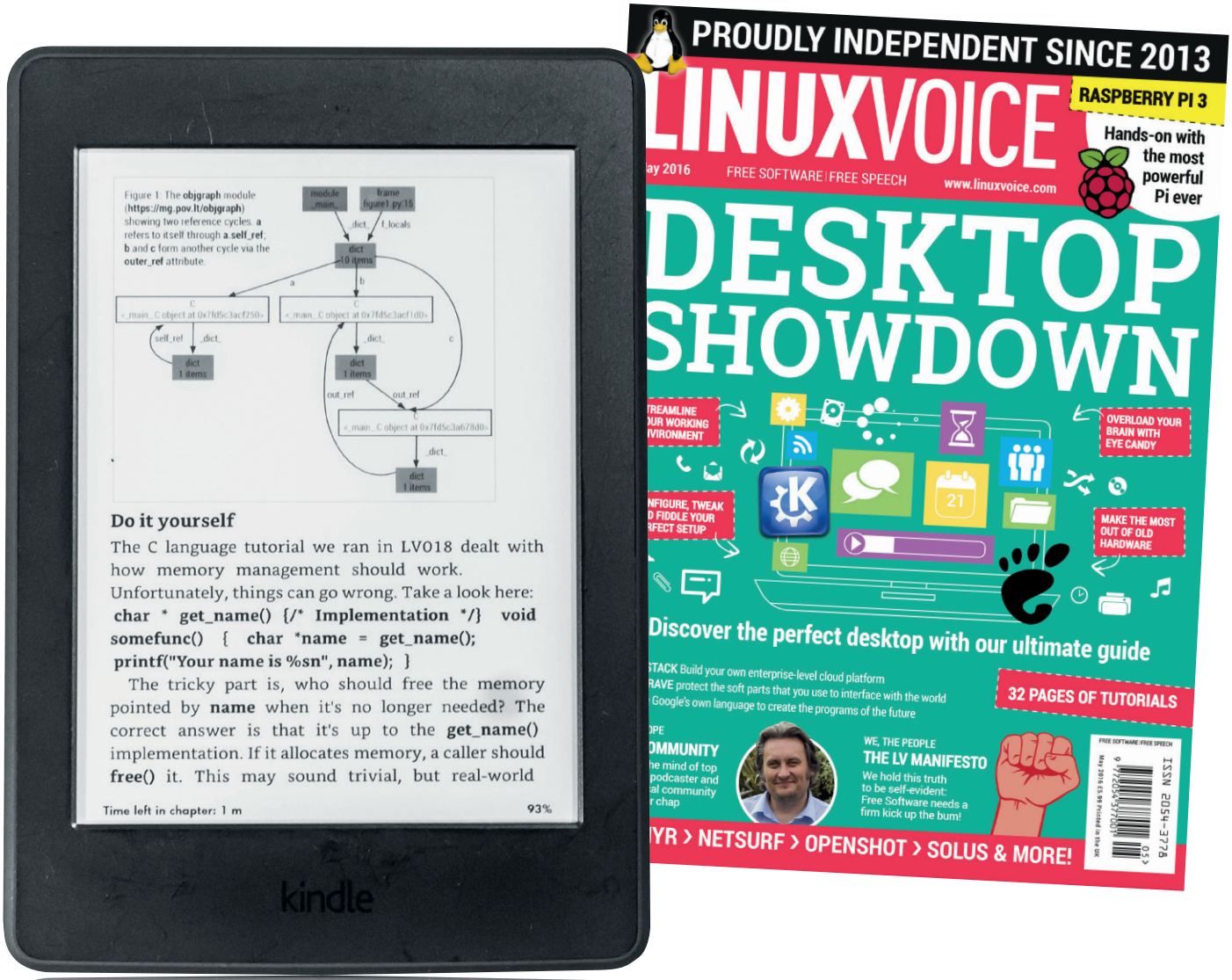
command.

That's all there's to it. You should now be able to upload audio and video files to the *GMG* installation. Refer to the documentation on the project's wiki (**https://wiki.mediagoblin.org**) to further customise and expand your *MediaGoblin* instance to host other kinds of media files as well.

Mayank Sharma has been messing with technology for decades, and spends his spare time burning rubber/getting lost in his own personal *Wacky Races*.

LINUXVOICE

The only Linux magazine available as DRM-free PDFs and ePub



shop.linuxvoice.com

GET NUMPY WITH JUPYTER AND VAGRANT

Run Python 3, IPython and NumPy in your browser all without installing them!

SEBASTIAN GÖTTSCHKES

WHY DO THIS?

- Have a clean development environment ready in minutes.
- Write Python code and run it, all in your browser.
- Install different versions of programming languages and libraries.

Getting started with Python is fairly easy if your OS has Python pre-installed. But what about using Python 3 instead of 2, installing various libraries that may conflict with other projects you are working on, and software not available for your distro?

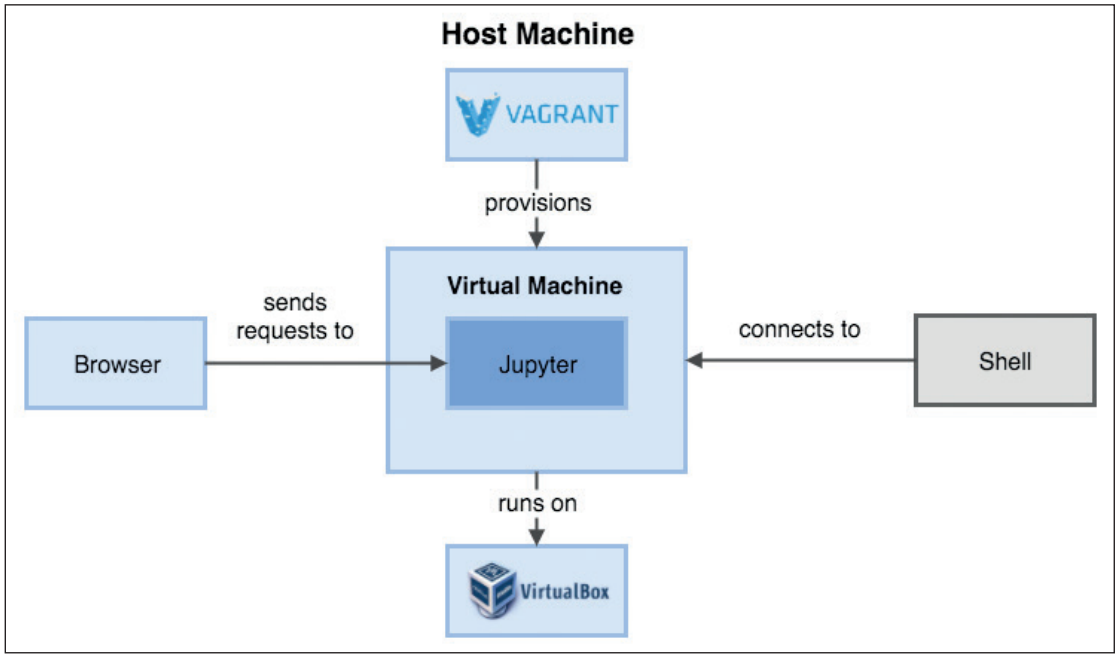
Installing all those pieces can be a daunting task and, depending on your OS, might require a good deal of knowledge. With *Vagrant* you can put all those parts into a virtual machine and not care about them anymore. This VM does not need to run any specific OS, so it can be any OS which makes it easy to install the software you need (like Ubuntu does). You don't need *Vagrant* to boot up a virtual machine, install Ubuntu and add the software you need, but *Vagrant* handles the tedious parts of virtual machines by providing base images, taking care of booting them correctly and provisioning, which means installing the software you want onto the machine. As you script all steps needed, they can be executed over and over again and also shared, making it possible to have the same environment available on different machines.

IPython (**https://ipython.org**) is a tool for running Python interactively that's more powerful than the standard Python interpreter. When combined with *Jupyter* (**https://jupyter.org**), you get a web-based Python environment that lets you easily experiment with Python code and share the results in HTML.

To run both *IPython* and *Jupyter* with Python 3 without touching the host system, we'll first set up a development environment inside a virtual machine with *Vagrant* and then use the *IPython* notebook in our browser.

Vagrant is configured using one file containing the details about the virtual machine, the base image to be used and the code to be run after the machine is booted. This file can be shared and used to make (almost) identical virtual machines on many hosts. You don't need to mess with *VirtualBox* manually any more or copy virtual machine images to different physical machines.

To get started, you need to have *Vagrant* installed as well as one virtualisation provider. *Vagrant* can be installed from the project homepage at **www.vagrantup.com** or through your distro's repository. The most prominent virtualisation provider is *VirtualBox*, which can be installed through the repositories as well or downloaded from **https://www.virtualbox.org**.



Both *Vagrant* and *VirtualBox* run on your host, with the Browser and a shell accessing the virtual machine using http and ssh. Inside the VM, the *Jupyter* server runs the *IPython* notebook.

Afterwards, you should be able to run

```
vagrant --version
```

in your terminal and get back the version. For this tutorial, version 1.7.4 is assumed, but everything should run with *Vagrant 1.5* and higher. The installed *VirtualBox* version is 5.0.8, but *Vagrant* should be able to work with Version 4 just as well.

Now, let's create a folder for our project called **python-nb**. Open up a terminal, navigate to this directory and run

```
vagrant init bento/ubuntu-15.04
```

This tells *Vagrant* to create a configuration file (which is called **Vagrantfile**) that has the base box **bento/ubuntu-15.04** already mentioned. Base boxes are images of virtual machines created with the provider you are using (in this case, *VirtualBox*). *Vagrant* downloads them automatically so they are available for usage and imports the image to create a new virtual machine. You can either use pre-build base boxes (like we do in this tutorial) which you can find at **https://atlas.hashicorp.com/boxes/search**, or create your own (the *Vagrant* documentation has a good introduction at **https://docs.vagrantup.com/v2/boxes/base.html**).

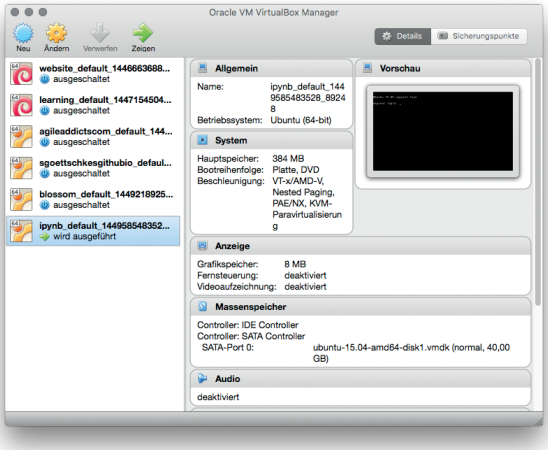
The next step would be to boot that box by executing

```
vagrant up
```

and waiting while *Vagrant* is downloading the base box, importing it and setting everything up for you to use. Afterwards you can access your virtual machine by running

```
vagrant ssh
```

which opens an SSH connection into your VM. You could go ahead and install all the software you need by hand now, but if you do this your setup is not reproducible, and next time you remove the box and create a new one, you'll just get the base image again. That's why *Vagrant* comes with provisioning, which means you can write down the steps you want to execute inside the virtual machine and *Vagrant* executes them.



You can see your virtual machines in the *VirtualBox* GUI, and even interact with them if you like.

We can use a sequence of shell commands to get us from base image (the Ubuntu base box) to our final state. The provisioning part inside a **Vagrantfile** looks like this:

```
config.vm.provision "shell", inline: <<-SHELL
  apt-get update
  apt-get install -y build-essential python3-dev python3-pip
  pip3 install ipython jupyter numpy
SHELL
```

Put this before the last **/end** statement in your **Vagrantfile** and run **vagrant provision**. What happens is that *Vagrant* executes every statement in the provision block. After updating the **apt-cache**, it installs a few **apt** packages we need for *Jupyter* and *NumPy* and afterwards uses **pip** to install the packages from **pypi**. It's important to point out that by default, *Vagrant* executes those commands using **sudo**. This is fine for us as both **apt** and **pip** need **sudo** and it saves us a few characters.

If you SSH into your box (again, using

```
vagrant ssh
```

and run the *iPython* notebook server with

```
ipython notebook
```

you'll see the server starting and listening on localhost:8888. If you go to this address using your favourite browser, you'll not have any luck. Remember, this isn't running on your localhost but inside a virtual machine, so there are two things needed to make it accessible from outside.

First, we need to tell *Vagrant* to forward the port 8888 from our host to our guest. This can be done with one line in our **Vagrantfile**:

```
config.vm.network "forwarded_port", guest: 8888, host: 8888
```

Place this line somewhere below the

```
config.vm.box = "bento/ubuntu-15.04"
```

line inside your **Vagrantfile**.

We can forward any port on the host to any port on the guest. They do not need to match, but it's often easier to forward to the same port so you don't get confused. We need to restart the virtual machine so

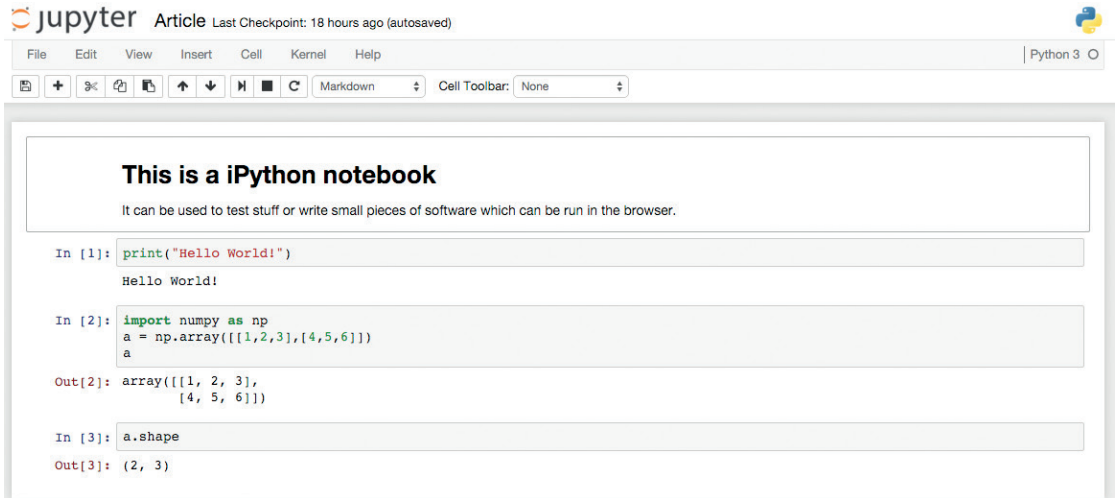
Vagrant vocabulary

Vagrant comes with a set of commands used to interact with a virtual machine. It's a good idea to know what they mean:

- **vagrant up** Boot the virtual machine. If the box doesn't exist yet, it'll be created as well.
- **vagrant provision** Run the provisioning scripts. This is done automatically if the box is booted for the first time.
- **vagrant ssh** Establish an SSH connection with the VM. This will take care of SSH configuration (the port and SSH key) automatically.
- **vagrant halt** Shut down the virtual machine.
- **vagrant reload** Reboot the virtual machine. This is the same as a **vagrant halt && vagrant up**.
- **vagrant destroy** Remove the virtual machine completely. This will also destroy the virtual hard drive used, so all data inside the VM is removed!

There are more commands to interact with *Vagrant*. You can view them using

```
vagrant help
```



The final *IPython* notebook in a browser window.

Vagrant can pick up these changes. Run

```
vagrant reload
```

on your host to do that.

The second thing we need to do is tell the *IPython* notebook server to run on host 0.0.0.0, which is essentially telling it to accept connections from anywhere. This is important because "localhost" on the guest means that the server only listens to requests from inside the VM and even with the port forwarded, the request is still coming from your host machine.

Let's SSH into the VM once more and run

```
ipython notebook --ip=0.0.0.0 --no-browser
```

and try to go to **http://localhost:8888** in the browser. You should see the *IPython* notebook homepage, which means you can now run Python 3 with *NumPy* (and any other package you install) in your browser by creating a new notebook (click on New in the top-right corner and select Python 3 under the Notebooks heading) and writing code.

There is still one problem though: say you have written some awesome code and you want to share your notebook (it's a text file intended to be shared after all). It's stored inside the VM in the same directory you were in when you started the *IPython* notebook server, which is most likely:

```
home/vagrant
```

This path isn't accessible from your host. Worse, if you destroy your VM, your notebooks are gone as well!

Vagrant comes with shared folder support, which means you can mount folders from your host machine inside the guest and use those to exchange files. It's common to mount a directory containing the project code into the virtual machine. On your host, create a folder called **notebooks** inside your project directory. Add the following code to your **Vagrantfile** below the

```
config.vm.network
```

line:

```
config.vm.synced_folder "notebooks", "/opt/notebooks"
```

This tells *Vagrant* to mount the folder notebooks on our host (this is a relative path to our **Vagrantfile**) at **/opt/notebooks** inside the virtual machine. Do a

```
vagrant reload
```

on your host once more to restart the virtual machine so *Vagrant* can mount that folder. Execute

```
vagrant ssh
```

again and **cd** to **/opt/notebooks**. You should see an empty directory. If you run the command to start the *IPython* server from above again and create a new notebook, you'll see it turn up both inside your VM and on your host.

If you're done for the day, you can halt the VM using

```
vagrant halt
```

If you want to start all over, run

```
vagrant destroy
```

which removes the virtual machine and start from scratch with

```
vagrant up
```

Of course you can tailor your environment on your guest machine just as you would a physical machine. To try this, let's create an alias for starting the *IPython* notebook server with the correct parameters already set so you don't need to remember them. On Ubuntu, this can be done by writing into the **~/.bash_aliases** file. Let's add another shell provisioner to our **Vagrantfile**:

```
config.vm.provision "shell", privileged: false, inline:
<<-SHELL
 echo 'alias ipy="ipython notebook --ip=0.0.0.0
--notebook-dir=/opt/notebooks/ --no-browser
--port=8888"' > /home/vagrant/.bash_aliases
SHELL
```

This time, the shell command is run as the **vagrant** user without **sudo** as we pass the **privileged** parameter. We do this so the **.bash_aliases** file has the correct permissions and because it's not needed to run this command as root. To provision an already existing VM use:

```
vagrant provision
```

This will execute all provisioner steps once again. If you have any shell command that fails the second

time they are run (eg creating a symlink), you will have to change them. It's a good idea to write provisioning scripts in a way that they can be run any number of times and only execute code that's needed. This can be done by checking whether a symlink already exists before symlinking, for example.

The complete **Vagrantfile** (with all comments removed for readability) would look like this:

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

Vagrant.configure(2) do |config|
 config.vm.box = "bento/ubuntu-15.04"
 config.vm.network "forwarded_port", guest: 8888, host: 8888

 config.vm.synced_folder "notebooks", "/opt/notebooks"

 config.vm.provision "shell", inline: <<-SHELL
  apt-get update
  apt-get install -y build-essential python3-dev python3-pip
  pip3 install ipython jupyter numpy
 SHELL
 config.vm.provision "shell", privileged: false, inline: <<-SHELL
  echo 'alias ipy="ipython notebook --ip=0.0.0.0 --notebook-dir=/opt/notebooks/ --no-browser --port=8888"' > ~.bash_aliases
 SHELL
end
```

After this setup works, we can now go ahead and work with a new *IPython* notebook. After you have SSH'd into your VM and run **ipy** to start the iPython notebook server, go to **http://localhost:8888** and create a new notebook on the right by clicking New and selecting Python 3. You are redirected to a new screen which represents your notebook. Start writing some code in the textfield next to **In []**:

```
print("Hello World!")
```

Clicking the Run button will execute this line of code, show you the result and move the cursor to the next cell.

All modules that are available in Python will also work inside an *IPython* notebook. As we've installed *NumPy* already, we can use it without further changes.

Add version control

Both the **Vagrantfile** and the provision scripts are text files, which make them perfect for storing in a version control system. You can use Git, Mercurial or SVN to hold the versions of the files that build your virtual machine. This makes it possible to destroy a virtual machine; go back to a previous commit and build the box using those instructions. You can also share your environment using a code repository.

Just make sure to exclude the **.vagrant** folder as it contains information about your host system. Having this inside version control will mess things up for others checking out your code!

Different ways to provision

In this tutorial, the only provisioner we use is the shell provisioner, which executes shell commands or scripts. There are many more provisioners built into *Vagrant* that have advantages over using shell commands. For example, they'll take care of all the details by providing modules or plugins to interact with **apt**. They might also offer a templating system so you are able to provide templates for configuration files which are automatically put in the right place inside your virtual machine.

If you want to start using a more sophisticated provisioning provider, try *Ansible* (**www.ansible.com**). This tool keeps things simple while still providing powerful features. *Ansible* is a tool to execute commands on remote systems, so it's possible to set up your staging and production environment and your development machine using the same tool and achieve a similar configuration.

Put this in your second cell:

```
import numpy as np
a = np.array([[1,2,3],[4,5,6]])
a
```

After running the cell, you'll see the *NumPy* array printed. The *IPython* interpreter takes care of outputting the result of the last line automatically without the need to put a **print** statement. In fact, the interpreter outputs many datatypes much more nicely than a simple **print** statement would.

We can use the variable **a** in another cell as well. In Cell 3, let's try putting

```
a.shape
```

and see if this works. It does and prints **(2, 3)** to tell us that the shape of the multi-dimensional *NumPy* array is 2 rows, 3 columns.

The great thing about *IPython* notebooks is that you can add Markdown cells and mix those with your code. Use the Cell Type Dropdown in the toolbar to change the type of a cell from Code to Markdown. Markdown is a simple way to annotate your text with just a little bit of styling while maintaining readability of the text at the same time. Let's add a headline as well as some text:

```
# This is a iPython notebook
It can be used to test stuff or write small pieces of software which can be run in the browser.
```

If you hit run again, you can see that the layout changes a bit and the hash sign is gone. The markdown has been converted to HTML, with the headline now being a **h1**. Move the cell up to the top by clicking it and then clicking the up arrow in the toolbar until the cell is at the top.

Now that you've seen how to create a virtual machine for *IPython* using *Vagrant*, you can apply the same techniques to manage almost any server-side software. Doing this, you'll gain the benefit of having reproducible machines that are easy to manage.

Sebastian Göttschkes felt the pain of different development environments early in his career and, looking for a solution, found *Vagrant*. He now runs a VM for every project!

WRITE A PROGRAM TO TEE OUTPUT USING RUST

Learn the latest technology to come from the clever chaps and chapesses at Mozilla.

AMIT SAHA

WHY DO THIS?
- Reduce segfaults.
- Get a new job in Silicon Valley.
- Banish dangling pointers!

The Rust programming language (**www.rust-lang.org**) is a systems programming language which aims for three things – safety, speed and concurrency. If you are not familiar with it, one way to look at it is considering using it where perhaps the C programming language would be otherwise suitable, but you want something safer. A few ways it leads to safer programming is by disallowing usage of uninitialised variables, variables defaulting to being immutable and the notion of ownership. Remarkably, all of these are achieved at compile time.

In this article, we will write a program that will perform functionality similar to the **tee** program (**http://linux.die.net/man/1/tee**). Figure 1 illustrates the functionality of the **tee** program. During the course of writing the program, we will learn about some of Rust's features I mentioned earlier. In addition, we will learn about *Cargo*, Rust's project management tool. Let's get started!

Installation and setting up

Before we can write our first program, we will need to install the Rust compiler (**rustc**) and **cargo**. We will be installing Rust 1.7, the latest stable release. Download the Rust static binary for Linux from **https://static.rust-lang.org/dist/rust-1.7.0-x86_64-unknown-linux-gnu.tar.gz** (or if you are using 32-bit Linux, the 32-bit version from **https://www.rust-lang.org/downloads.html**) and extract the gzipped tarball and run the **install.sh** script:

```
$ tar -zxvf rust-1.7.0-x86_64-unknown-linux-gnu.tar.gz
$ cd rust-1.7.0-x86_64-unknown-linux-gnu
$ ./install.sh
...
Rust is ready to roll.
```

Let's verify that we have installed the tools we need correctly:

```
$ rustc --version
rustc 1.7.0 (a5d1e7a59 2016-02-29)
$ cargo --version
cargo 0.8.0-nightly (28a0cbb 2016-01-17)
```

Hello world

We will now write our first program in Rust. We don't need to use **cargo** for a hello world program in Rust, however it is a scalable approach, especially when we are working on non-trivial projects. Hence, we will get familiar with the absolute basics of **cargo** in this tutorial. The first step is to create a project:

```
$ cargo new helloworld --bin
```

The **new** sub-command creates a new project, with the **--bin** switch specifying that we want to create an executable crate, rather than a library crate. (We discuss crates later on in the article.) When you execute the command above, a new sub-directory,

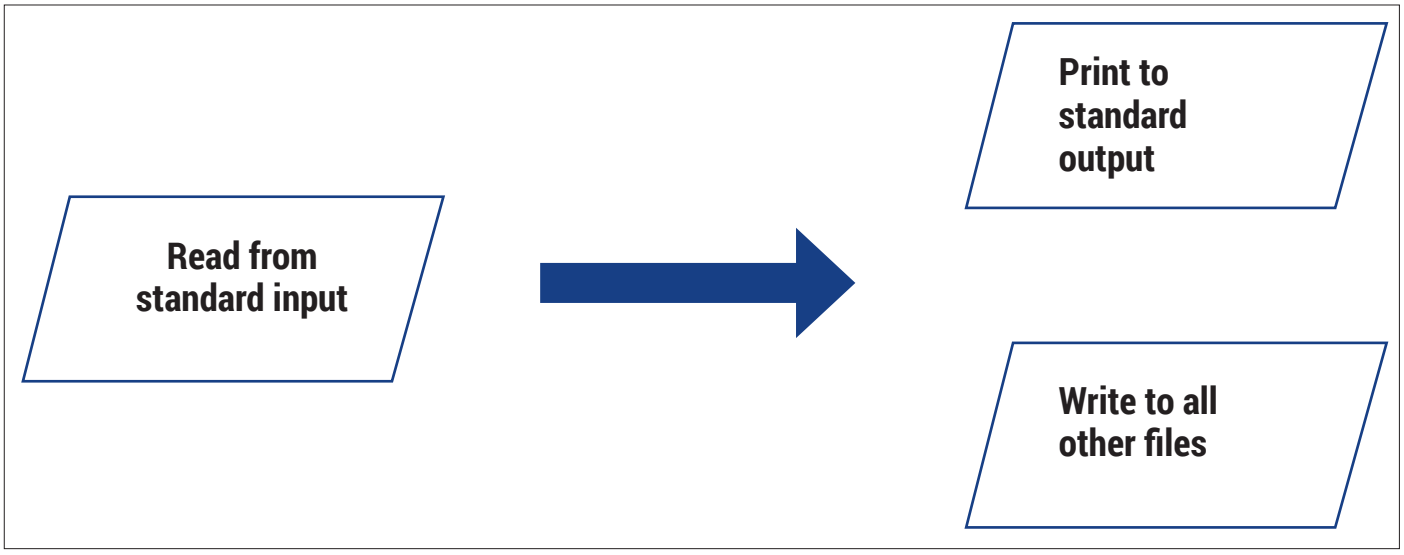


Figure 1: Illustration of the functionality of the tee command

helloworld, is created having a **src/** sub-directory and a **Cargo.toml** file:

```
helloworld/
├── Cargo.toml
└── src
└── main.rs
```

You will notice that **helloworld** is initialised with git version control. Thus, when the time comes, you can easily push your code into a remote repository. We will ignore **Cargo.toml** for now, since it gets interesting only when our project is dependent on others, or we plan to publish our library. Let's see what is generated in the **src/main.rs** file:

```
fn main() {
    println!("Hello, world!");
}
```

Since Rust is a compiled language, we first have to compile it. To do so, we will use the **build** sub-command of the **cargo** tool from the **helloworld** directory:

```
$ cargo build
..
```

Now that our program is compiled, we can run it, using the **run** command:

```
$ cargo run
..
Hello, world!
```

Congratulations, you have successfully run your first program. In the **project** directory, you will find a new file, **Cargo.lock** and new sub-directory, **target** has been created. The **.lock** file created becomes important when we are using external libraries, so once again we will ignore that for now. The **target** sub-directory contains the executable for our program (**target/debug/helloworld**) along with other files.

Let's now understand the program we wrote above. The first line declares the **main()** function – the entry point to our program. The **main** function doesn't take any parameters, nor does it return any data. A function that accepts arguments or returns any result will be defined differently (Figure 2).

Curly braces ahoy!

The function body starts with an opening brace, (**{**) and ends with a closing brace (**}**). In the function body, we have a single line: **println!("Hello, world!");**, which prints the string "Hello, world!" to the standard output followed by a new line.

println!() is a macro that prints a string to the standard output. For our purposes here, we will focus on just using this macro and not get into what a macro means in Rust and how you can define your own. In general, however, when you come across a function call where the "function name" has a trailing **!** mark, you should know that it is a macro.

Note how the line above is terminated by a semicolon? Most of our lines will be terminated by a semicolon. In Rust, every line is an expression and returns a value. When we don't care about the returned value, we use a semicolon (**;**) to terminate the expression and the expression is referred to as an expression statement. The other type of statement is a declaration statement, which we will get to in our next program.

User input, data types and handling results

In our second program, we will write a program which upon execution will take a line of input from the user and simply print it back.

Create a new project using **cargo new input_string --bin**, which will generate a **main.rs** file in the **input_string/src sub-directory**. Next, we will type in the following program and save it.

> We will write a program that upon execution will take a line of input from the user and simply print it back

```
use std::io;

fn main() {
    let mut line: String = String::new();
    let stdin: io::Stdin = io::stdin();
    let res: io::Result<usize>;
    res = stdin.read_line(&mut line);

    if res.is_ok() {
        // Unwrap the result to extract the value
        let nbytes: usize = res.unwrap();
        println!("{}({} bytes read)", line, nbytes);
    } else {
        res.unwrap();
    }
}
```

There are a bunch of new things we've introduced above. Let's start off with the first line, **use std::io**. To understand what we are doing here, we have to very briefly discuss 'crates' and 'modules' in Rust.

A crate in Rust is equivalent to a package in Python or a library in C. Each crate in turn consists of a main module and other modules. A crate can be either a binary crate or a library crate. We will only be discussing binary crates in this article. In addition, all our crates consists only of the main module. The statement, **use std::io;** thus tells the Rust compiler that we will be using the **io** module from Rust's standard (**std**) library in our program.

The first statement in our **main()** function:

```
let mut line: String =String::new()
```

declares a new mutable variable binding, line of the type **String** and its initial value is an empty **String** (which we can then append to later). By default, variables in Rust are immutable, meaning you cannot assign a value to it more than once. To be able to do

PRO TIP

Using 'cargo run', we can combine the building and running steps into one. For example, we can build and run the above project using 'cargo run'.

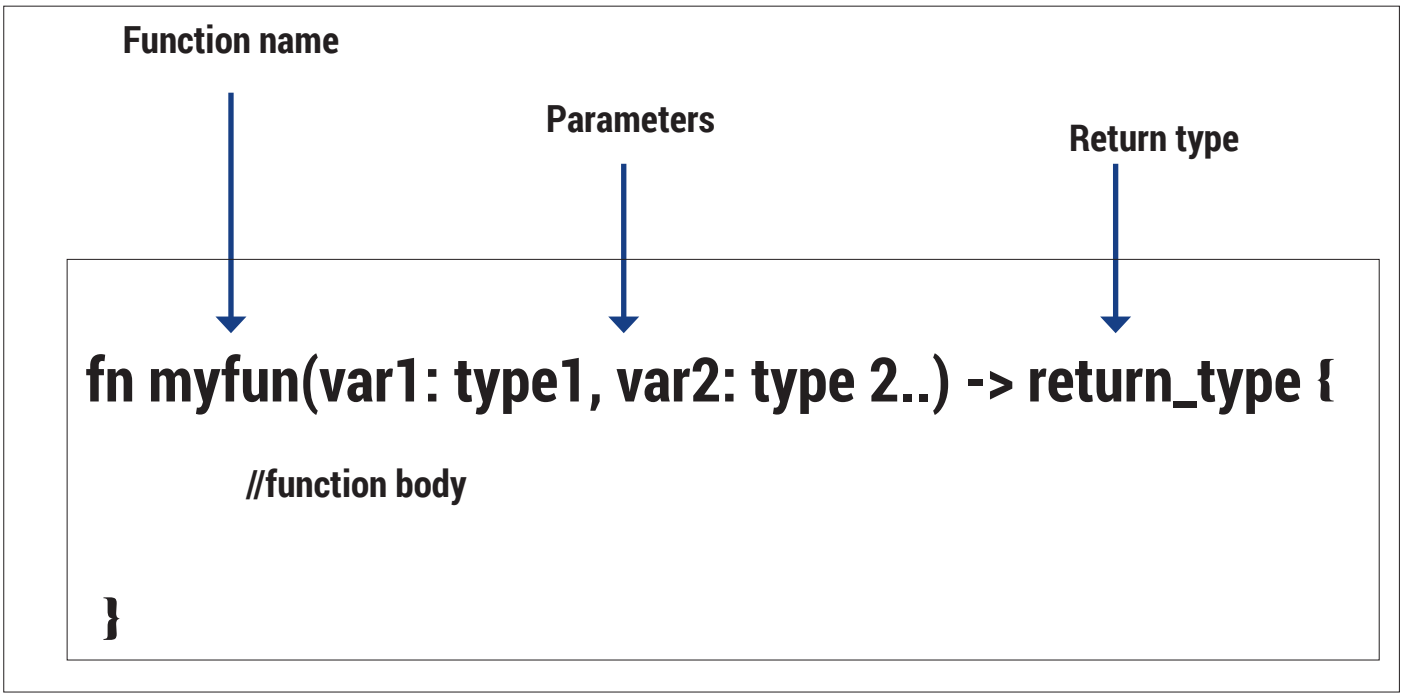


Figure 2: A function in Rust (please refer to the section on functions in the Rust reference: **https://doc.rust-lang.org/reference.html#functions** to learn more).

so, we need to declare it as mutable. Rust supports type inference, which means we could have omitted the type annotation from our earlier statement for brevity.

In the second statement, we obtain a reference to the standard input for reading using the **stdin()** function in the **io** module. This function returns a reference of **Stdin** type defined in the **io** module, hence we declare a variable binding **stdin** to refer to this returned reference.

The next statement declares a new variable binding, **res of io::Result<usize> type**. The **Result** type defined in the **io** module is used to return the result of any IO operation which may result in an error. It is worth mentioning here that the Result type is an enumeration with the two variants, **Ok** and **Err** representing success and error respectively. A successful result encapsulates a data item used to store the result of the operation. Thus, when we declare a variable binding of this type, we also need to specify the data type of this result. This data type will depend on the result we want to store in this variable. In this case, it turns out to be **usize**.

PRO TIP

The "let" keyword in Rust is used to declare a new variable binding. It's generic syntax is illustrated in Figure 3.

The **read_line()** function reads a single line of input from standard input and returns the number of bytes represented as data item of **usize** type. The bytes it read will be stored in **line**. Note how we pass a mutable reference to **line** when we call the **read_line()** function. This is our way of telling that we allow the **read_line()** function to modify what is stored in **line**.

Now that we have a result in **res**, we need to check if the call to **read_line()** was successful. We do so using the **if** statement. The **is_ok()** function returns **true** if the result corresponds to a success and **false** if there was an error. If the **read_line()** function returns successfully, we execute the following statements:

```
// Unwrap the result to extract the value
let nbytes: usize = res.unwrap();
println!("{}({} bytes read)", line, nbytes);
```

The first statement in this block is a demonstration of how we can write a comment in Rust. Rust also supports block comments:

```
/* This is a comment split over
multiple lines
*/
```

The next statement in the above block extracts the number of bytes returned as a result returned by the **read_line()** function using the **unwrap()** function and assigns the value to the **nbytes** variable. Next, we print the line we read and the number of bytes we read using the **println!()** macro. We call the macro with a string that has two "placeholders", **{}**. This will be substituted by the value of the variables that follow the string in order.

Let's now build and run the above project:

```
$ cargo run
Hello
Hello
(6 bytes read)
```

We can of course even pipe the output from another command to our program and it will print the first line it read:

```
$ cat src/main.rs | cargo run
..
Running `target/debug/input_string`
use std::io;
(13 bytes read)
```

Using pattern matching to handle errors

Using the **if** statement as above to check if the result was a success or an error works, but Rust has something more elegant for such tasks: the **match**

expression. Let's create a new project, **input_string_match** and then type in the following code into the **src/main.rs** file:

```
use std::io;
fn main() {
  let mut line: String = String::new();
  let stdin: io::Stdin = io::stdin();
  let res: io::Result<usize>;
  res = stdin.read_line(&mut line);
  match res {
    Ok(nbytes) => {
      println!("{}({} bytes read)", line, nbytes);
    }
    Err(err) => {
      println!("{}", err);
    }
  }
}
```

We can see that the only difference between our previous code and this is how we handle the result returned by the **read_line()** function. Here, we use the **match** expression to perform different actions based on the result. If the result returned is a success, it matches the **Ok()** block, else it matches the **Err()** block. The **match** expression also takes care of creating the variable binding to the data returned upon success or an error – **nbytes** and **err** respectively.

Vectors and iterators

Vectors in Rust is a growable list of elements of the same type and is created using the **vec!** macro. Let's see an example – create a new project using **cargo new vectors --bin** and replace its **src/main.rs** by the following program:

```
fn main() {
  let mut v = vec![1, 2, 3];
  v.push(4);
  for elem in v {
    println!("{}", elem);
  }
}
```

The first statement **let mut v = vec![1, 2, 3]** creates a new mutable variable binding, v to vector, [1, 2, 3]. Note how we did not declare any type of the elements that will be in the vector, but it was inferred from the elements we initialised it with. We create a mutable variable binding so that we can add and remove elements from the vector later on. In the next statement, we use the **push()** function to append another element to the vector.

Next, we use a **for** loop to iterate over the vector, **v** and print each of the elements, **elem**. When you build and run the program, you will see the following output:

```
1
2
3
4
```

Command line arguments

The most basic functionality we need to have in a command line program like **tee** is to be able to read user input supplied as command line arguments. We will create a new project, **read_cmdline_args** using **cargo new read_cmdline_args --bin** and replace the generated code in **main.rs** with the following:

```
use std::env;
fn main() {
  for arg in env::args() {
    println!("{}", arg);
  }
}
```

The command line arguments supplied to a program upon execution can be accessed using the **args()** function in the **env** module of the standard library. Hence, we start our program with the **use std::env** statement.

In the **main()** function, we directly iterate over the result of calling the **env::args()** function using a **for** loop and print each argument.

If we run the project using **cargo run arg1 arg2 12**, it will print the path to the executable followed by **arg1**, **arg2** and **12** on separate lines:

> The most basic funtionality in a command line program is the ability to read user input specified as arguments

```
$ cargo run arg1 arg2 12
..
arg1
arg2
12
```

If we don't supply any arguments, it prints only the path to the executable:

```
$ cargo run
target/debug/read_cmdline_args
```

Creating and writing to files

There is one missing piece left before we can write our program to implement the functionality of the **tee** program – writing data to files.

We will create a new project using **cargo new file_write --bin** and write the following program into the **src/main.rs** file:

```
// Write some data into a file
// Usage: cargo run <file path> string
use std::io::Write;
use std::fs::File;
use std::env;
fn main() {
  let args: Vec<String> = env::args().collect();
  if args.len() < 3 {
    panic!("Usage: cargo run <file path> string");
  }
  let file_name = args[1].clone();
  let contents = args[2].clone();
  match File::create(file_name) {
    Ok(mut f) => {
```

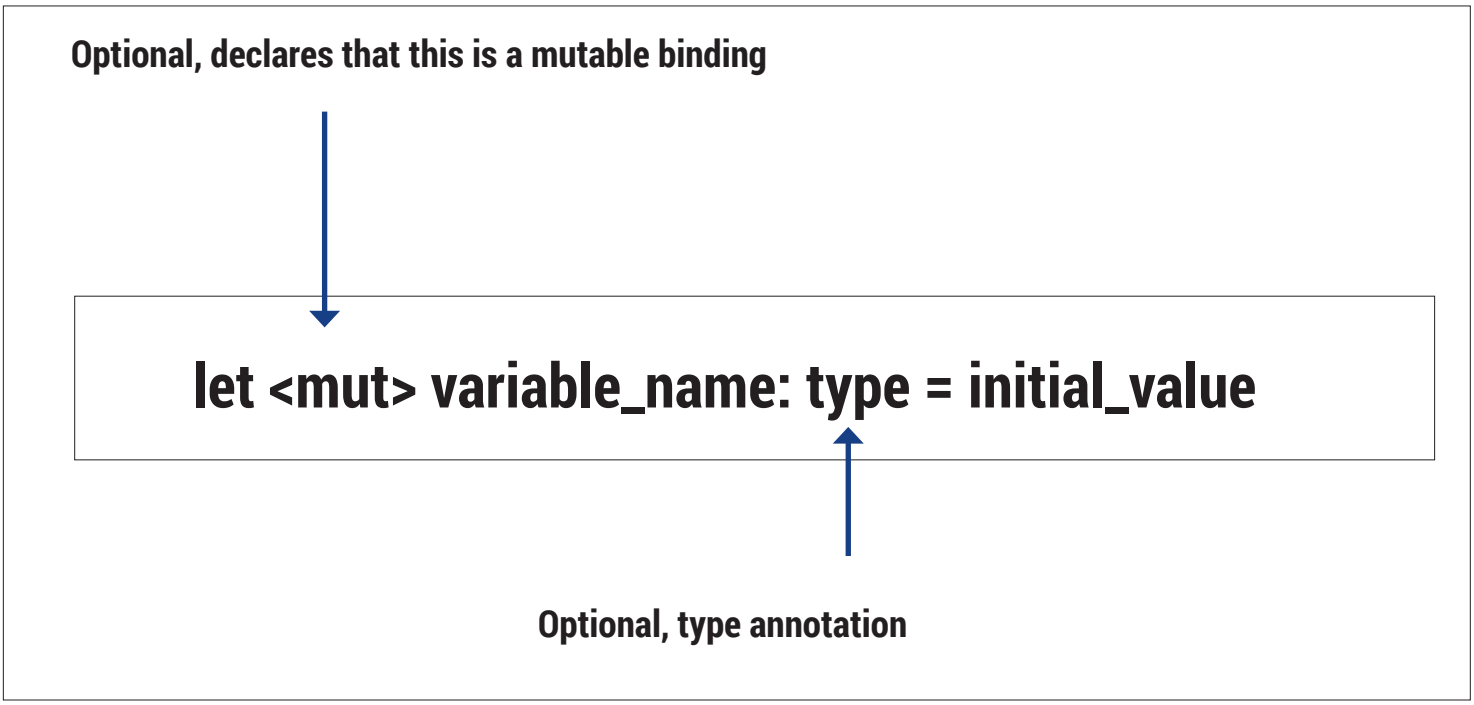


Figure 3: Generic syntax of declaring a variable binding in Rust.

```
        match f.write_all(contents.as_bytes()) {
            // _ is a throwaway variable since we don't
            // have anything to do if write_all() was
successful

            Ok(_) => {}
            Err(error) => {
                println!("Error writing to file: {}", error);
            }
        }
    }

    Err(error) => {
        println!("Error when creating file for writing: {}",
error);
    }
  }
}
```

In the first statement of the program, we import the **Write** trait from the **std::io** module. A trait in Rust is roughly equivalent to a behaviour certain types share.

Next, we import the **File** type from the **std::fs** module and the **std::env** module.

In the first statement of the **main()** function, we are creating a vector of Strings from the command line arguments passed to the program. We do so using the **collect()** function.

Next, we check the length of the vector using the **len()** function, and if it is less than 3, we want to print a message and exit. We do so using the **panic!()** macro. This prints the supplied string and exits the program.

Next, we create two new variable bindings, **file_name** and **contents** corresponding to the second and third command line arguments.

We then use the **create()** function to open the file for writing. If the file specified exists, it is truncated. If the file was opened successfully, we write to it using the **write_all()** function. Since the **write_all()** function expects the data as bytes instead of a string, we convert the string to bytes using the **as_bytes()** function.

```
$ cargo run /tmp/file.txt hello
Running `target/debug/file_write /tmp/file.txt hello`
```

The file **/tmp/file.txt** should contain the string **hello**.

Note how we don't worry about closing the file explicitly in the above program. Rust automatically closes the file when all references to it go out of scope.

The tee program

Now using all that we have learned so far, we are ready to write the program to implement the basic functionality of the **tee** command. You can find the entire program at **https://github.com/amitsaha/linux_voice_2/blob/master/tee/src/main.rs**. We will discuss the program in two snippets.

We have a number of **use** statements at the top of the program – where we import different modules and types from the standard library. Next, the first block of the main function opens the standard output for writing and any additional files that were specified as command line arguments:

```
let mut stdout = io::stdout();
// Process any additional files specified
let args: Vec<String> = env::args().collect();
let mut files = Vec::new();

if args.len() > 1 {
  for file_name in args {
    match File::create(file_name) {
      Ok(f) => {
        files.push(f);
      }
      Err(error) => {
```

Resources

- The Rust programming language book **https://doc.rust-lang.org/book**.
- Crates and Modules **https://doc.rust-lang.org/book/crates-and-modules.html**.
- Cargo **http://doc.crates.io/guide.html**.
- Primitive types in Rust **https://doc.rust-lang.org/book/primitive-types.html**.
- Standard library modules **http://doc.rust-lang.org/std/#modules**.
- Reference **https://doc.rust-lang.org/reference.html**.
- Rust by example **http://rustbyexample.com/index.html**.
- Rust for C/C++ programmers **https://github.com/nrc/r4cppp**.
- Error handling **https://doc.rust-lang.org/book/error-handling.html**.
- Linux coreutils written in Rust **https://github.com/uutils/coreutils**.
- Rust learning **https://github.com/ctjhoa/rust-learning**.

```
            println!("Error creating file for writing: {}", error);
        }
    }
  }
}
```

The first statement in the above block calls the **stdout()** function in the **io** module to obtain a reference to the standard output. A mutable variable binding, **stdout** is created to refer to it later on.

Next, we initialise an empty vector using the **new()** function and create a mutable variable binding, with files to refer to it. Next, we check if any files were specified. We open each of these files for writing, and if successful, we save the opened file object to the files vector.

The next block of code reads from the standard input endlessly, and for each line it reads, writes it to the standard output and any of the additional files that was specified:

```
// Read from standard input and write to all the files
let mut line = String::new();

loop {
  match io::stdin().read_line(&mut line) {
  Ok(n) => {
    // Have we read all the lines, if yes, break
    if n == 0 {
      break;
    }
    // Write to standard output
    match stdout.write_all(line.as_bytes()) {
      Ok(_) => {}
      Err(error) => {
        println!("Error writing to stdout: {}", error);
      }
    }

    // Write to any additional files
    // We obtain a mutable reference to the file from the vector
    for f in &mut files{
      match f.write_all(line.as_bytes()) {
        Ok(_) => {}
        Err(error) => {
          println!("Error when writing to file: {}", error);
        }
      }
    }
    // Clear the line we just read so that we don't keep
    // appending
    line.clear();
  }
  Err(error) => {
    println!("Error reading from stdin : {}", error);
    }
  }
}
```

An infinite loop in Rust is created by enclosing the loop body within **loop {}** as follows:

```
loop {
  // This statement will be executed endlessly
}
```

The **read_line()** function we use to read from standard input keeps appending the lines read into the variable, line, hence we call the **clear()** function above to empty the string.

To execute the code for our **tee** program, clone the repository from **https://github.com/amitsaha/linux_voice_2**, change the directory to the **tee** directory, and use **cargo run** to run the program. Of course, we can specify any additional files as command line arguments. For example, the following will execute the **who** program, print the output to the standard output and to the file **/tmp/who.output**:

> We have a number of use statements at the top of the program – where we import different modules and types

```
$ who | cargo run /tmp/who.output
```

Conclusion

In this article, we learned just enough of the Rust programming language while writing a program to **tee** output. However, our program lacks one major feature – the option append to existing files. You can perhaps consider that an exercise to try next.

You can find all the Rust projects we created in the article at **https://github.com/amitsaha/linux_voice_2**.

The resources section above lists a number of resources where you can learn more about Rust including and beyond all the features we have learned about and used in the article.

Amit Saha is the author of *Doing Math with Python* (No Starch Press) and a software engineer. He blogs at https://echorand.me, tweets @echorand and can be reached via email at amitsaha.in@gmail.com

GET INTO VERSION CONTROL WITH **GIT**

Git is taking over! Don't get left behind: run your own server for fun and profit.

GRAHAM MORRISON

WHY DO THIS?

- Control all aspects of your computer setup.
- Make it easier to chare code (and work on other people's).

Version control (also known as source control management) systems perform two essential tasks. First, they let you track changes to a file so that mistakes can be reversed. Second, they enable lots of people to edit those files at the same time, while maintaining an independent track of all those changes. Mostly, those files and projects revolve around programming and development, but development needn't be the only case.

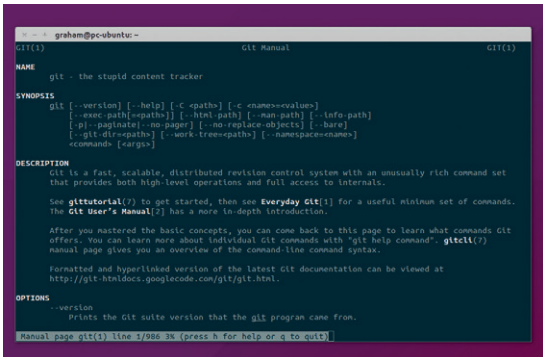
In this issue's interview, for example, old school hacker 'r0ml' is convinced that version control could be the enabler, allowing you to tinker and change things, whether that's a live website or your own configuration files. Being able to change things is one of the principles behind open source, and there's no reason why this magical ability shouldn't be confined to just programmers. There are even tools that put your entire **/etc** folder under version control, for instance, enabling you to roll forwards and backwards through any system changes, which is perfect for precarious servers with their finely balanced *Apache* and *Postfix* configurations. We're going to install, configure, serve and start using Git, most popular version control system around.

STEP BY STEP: SET UP GIT

1 What is Git?

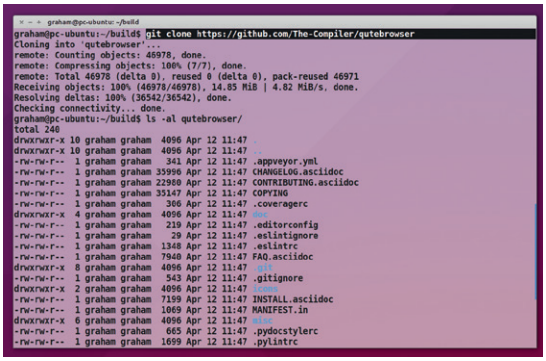
Even if you don't have any interest in development, you must have noticed that Git has taken over the world of version control. Before Git, there was a fragile ecosystem where Subversion, BitKeeper, Perforce and even the venerable CVS co-existed in quiet harmony. Git changed that, especially for open source projects.

When Linus Torvalds wrote Git his goal was to create something to replace the proprietary BitKeeper while keeping BitKeeper's main advantage. That advantage was being decentralised so that 1,000 developers didn't have to wait for a server while orchestrating a kernel release. This is what makes Git different from Subversion; in Subversion, the server is the gatekeeper to progress. With Git, every developer gets their own copy, or repository, and they work on this independently. The clever bit happens when all these parts are merged together.



2 Client Installation

Most distributions will either include the Git client or make it very easy to install, and this is all you need to communicate and collaborate with any other Git repository. Without a doubt, the most popular online repository is now GitHub, which has become a huge social network for developers. As of April 2016, it contains 35 million repositories serving 14 million project collaborators who use it both for code collaboration and for its wider infrastructure, such as issue tracking and documentation. You can make your own copy of any public project using the **git clone** command directly followed by the project's URL (**git clone https://**). This will copy all the files into a new local folder with the same name as the repository on the end of the URL, and you'll be able to explore its contents, build the project and make your own edits to the code, all without accessing the server again.



3 Terminology

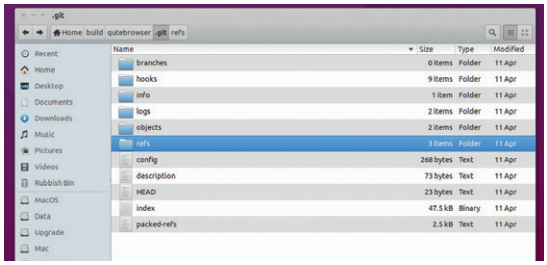
Before we can start playing with this Git revolution, we should first cover some of the words used by Git to explain its functions. If you've used some other version control system, these terms can seem counterintuitive, so here are 10 of the one that we feel are the most important:

- **branch** In Git, this is a label used to delineate a revision of a repository.
- **clone** As we've just shown, this is a copy of an existing Git repository.
- **commit** The process of saving your changes to your local copy.
- **fetch** Grabs changes from the online repository without merging them.
- **master** The definitive branch for a project, usually used for releases.
- **merge** Combines the changes from one repository with those in another (usually your local copy).
- **pull** Grabs changes from the online repository and merges them with your local repository.
- **pull request** Mostly used collaboratively to notify the maintainer of a repository to merge a developer's new change.
- **push:** sends your local changes to the online repository.
- **staging** This is a half-way point between a commit and a simple save, useful for partially implemented work.

4 File structure

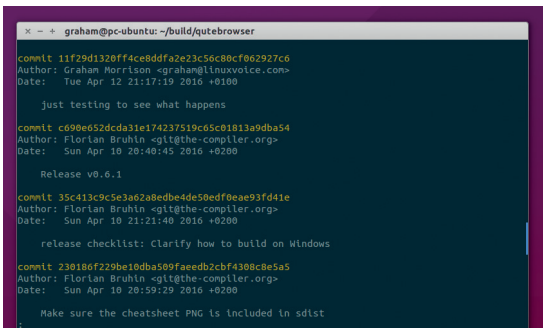
Linus wanted Git to remain simple, and while its scope and capabilities have become complicated, the way it works is still straightforward. Most of the magic happens with the help of a **.git** folder hidden within any repository you download. This folder contains everything Git needs to keep track of changes. **config** holds the local configuration, for example; **HEAD** will contain a reference to the branch you're currently working in; while **Index** contains data on files you've saved to the staging area.

The **objects** directory holds links to everything Git is managing, while **refs** and **tags** both hold references to branches and tags, with **tags** being more like a snapshot of a branch at one moment in time rather than an isolated developmental revision. By default, the **hooks** folder contains sample scripts that can be modified to trigger events after a commit and becomes very powerful when you link Git against your other systems.



5 How Git tracks changes

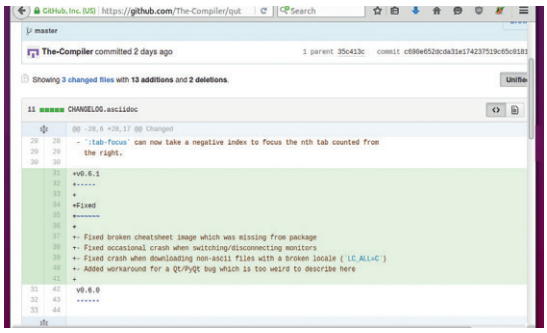
You may have noticed we've made lots of references to what we call 'references'. These references are at the core of how Git works. Unlike other systems, Git doesn't store changes specifically. Instead, it acts more like a filesystem for all your variations, with the key to each of those changes being a checksum of the content. This checksum is a SHA-1 hash, and is generated whenever you use **git commit** to finalise or stage your changes. You can see an ordered list of these hashes by typing **git log**, a command that shows the various commits made on your repository up to that point. You can show the details of any specific commit by typing **git show** followed by the first eight or so characters of the hash – whatever is enough to uniquely identify it. GitHub displays these values too, and the output from any change is the same as the output from the **diff** command.



6 The anatomy of a 'diff'

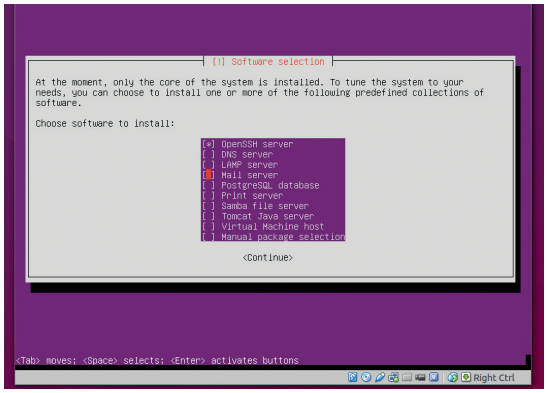
The output of **git show**, with its lines starting with '+++','---' and '@@', can be confusing. This is because the output uses the unified format of the GNU/Linux stalwart **diff** command, where **+** is used to denote additional lines, **-** is used to show removed lines, and **@** outputs the chunk of code that's been affected by each change. You can even see this command being executed early within the Git output (look for **diff --git a/ b/**) before the **diff** changes start to appear.

You can see these changes much more clearly in GitHub, where clicking on a commit's SHA-1 reference displays the differences for each file; this output is coloured and grouped together to show the differences in each file. If you ever really want to get into the details, we'd recommend using a graphical **diff** tool, such as *Kompare*, to check the differences between any two files.



7 Set up a server

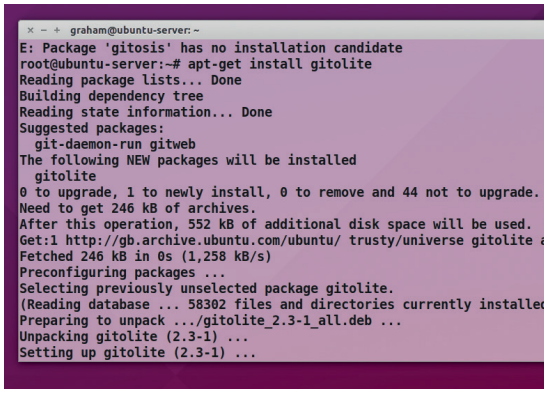
One of the biggest hurdles to using Git is getting practical experience of its commands and processes without breaking anything. You could set up your own experimental online repository with a service such as GitHub, but we've found the best option is simply to install a Git server locally and play with it yourself. That way you can set up a repository, import some code, make all the changes you need, and then look at how this affects both the server and the local storage. If you can do this with one of your own projects, even better, but you could just as easily import another project or simply play around by editing text files. We installed Ubuntu Server 14.04 into a virtual machine with SSH already running, but you could use a VPS or real hardware too.



8 Installing Git-o-lite

Installing the **git-core** packages on Ubuntu is almost enough to get you a working server, but it won't give you enough control or enough capabilities to manage different users and repositories, which is the main reason for installing a Git server in the first place. For that reason, we'd recommend installing **gitolite** instead. This makes it much easier to get complete control over your new system and turn your server into a fully fledged secure source code repository with multiple projects and users. On the Ubuntu Server, the main packages can be installed by typing **sudo apt-get install gitolite**, and we need to add a user to run the server, which can be accomplished by typing:

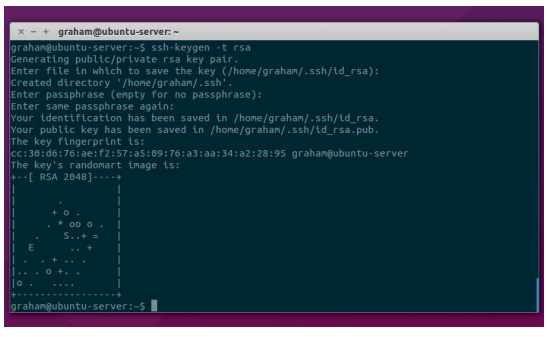
```
sudo adduser --shell /bin/bash --system --group --home /home/git git
```



9 Setting up SSH

Most configurations use SSH to deliver access credentials to a Git server. We've covered SSH quite a bit before, but essentially, typing **ssh-keygen -t rsa** on your client will generate a key pair, where one key is private and the other is public. Renaming the public key to something closer to your username will help with gitolite permissions configuration.

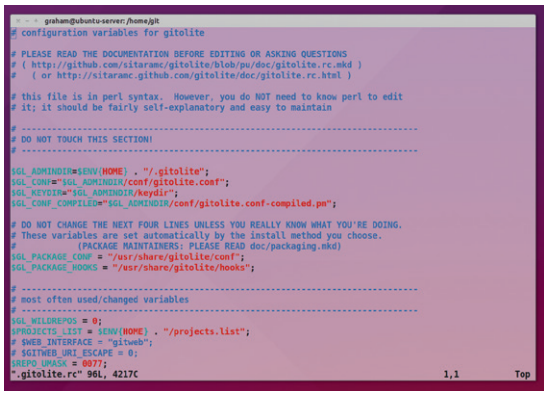
Administrator authentication needs the public part copied onto the server – use **sftp** if you have trouble with this – and we need the public key to complete the gitolite configuration. With the **.pub** key accessible on the server, type **sudo su git** to switch to the **git** user we created earlier, and type **gl-setup id_rsa.pub**. Change the path and filename so it points to the location and name your public key. By adding the key in this way, you can use the account with the private key as the gitolite administrator.



10 Configure gitolite

The configuration process continues and first asks where to to put the configuration file before then asking you for your favourite text editor. This is important because the following step will open the configuration file in this editor, and you'll need to be able to navigate the text, save and close, so don't be tempted to choose *Emacs* unless you know *Emacs*.

It's worth looking through the file, but we got a working configuration without changing any of the options. Save or exit to quit back to the command line as the **git** user, and exit from this too. You should now be able to access gitolite from the account and machine that generated the SSH key pair, which is where we need to go next to instantiate gitolite's own Git-based management interface.



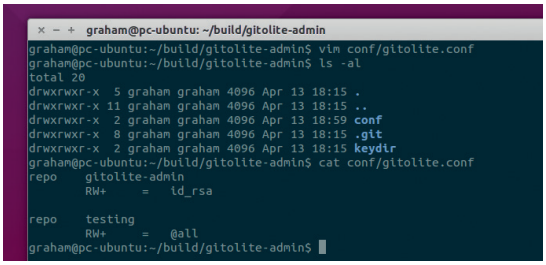
11 Checkout the configuration

The great thing about gitolite is that itss configuration is managed by Git itself, and your first step should be to check out this configuration from the account whose public key you've used by entering the following (changing the IP address to match that of your server):

```
git clone git@192.168.1.82:gitolite-admin.git
```

This will download a configuration file within its own folder, which can now be changed locally, and another folder called **keydir** which can be used to add the public keys of any other users you'd like to give access to. Any changes you do make to either will need to be committed (staged) and pushed (uploaded) before they can become active, and you can do this by first adding new files, committing them and pushing them back to the online repository:

```
git add keydir/graham.pub
git commit -m "adding grahams key"
git push
```

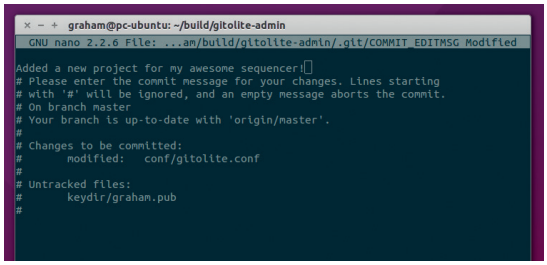


12 Creating a new repository

The configuration file within the **conf** folder needs to be edited to add a new repository, and each new addition needs to be entered in the following format:

```
repo meeq
    RW+     =   graham
    RW      =   andrew
    R       =   mike
```

The format is easy to understand, and gitolite's permissions system is powerful, allowing for groups and fine control over branches and tags. Our repository is for a project called 'meeq', which Graham, Andrew and Mike have various degrees of access to. The **+** allows a user to go backwards through the commits, while the unused **-** symbol can deny access. Thse usernames need to correspond to the names of the public keys you need to add to the **keydir** folder – Andrew's should be **andrew.pub**, for example, allowing him to access the repository. For users, the private key needs to be the first one selected by SSH.

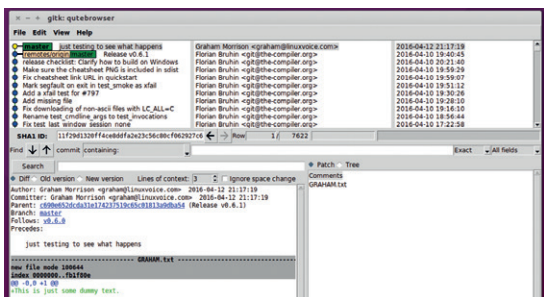


13 Initialise your repository

To make sure you've got the correct rights, it's easier to use a different account with a separate private key. Even if the public key is **graham.pub**, its private key can be the default **id_rsa** as long as it's the first chosen by SSH. If everything is working correctly, you can check out the new and empty repository with **git clone git@192.168.1.82:meeq.git** and switch to this new empty directory. You can now import your code, not forgetting to use **git add** so that files can be tracked, and create a new file for your new project. As shown earlier, typing **git commit -m "comment"** will add those changes to the local staging environment with a comment on the update, with **git push** uploading them to the server. When you now edit those files, you just need to **commit** and **push** for your modifications to be tracked and uploaded. If you want to see the differences between your last two edits, just use the **git diff** command without any arguments.

14 Start playing with Git

You can now start experimenting with your Git configuration and projects, safe in the knowledge that you won't be affecting any other server installation. This means you can learn to fork, tag, release and update your code, or even experiment with projects that aren't code, such as monitoring your configuration files or even part of your home directory. If you want to make things a little easier, try using a GUI Git client, such as gitk. Typing this in a project folder will open a GUI that lets you browse the branches and commits, including details like the SHA-1 hash. If you want to take this further, there are several tools. Our favourite is git-cola, which should be easily installable from your distribution. When it starts, select Open and point the requester at one of your Git project folders. Not only will it make visualising differences much easier, but you can perform almost all the same actions as Git with just a click of a menu. LV



Valentine Sinitsyn develops high-loaded services and teaches students completely unrelated subjects. He also has a KDE developer account that he's never really used.

CORE TECHNOLOGY

Prise the back off Linux and find out what really makes it tick.

Bash beyond the basics

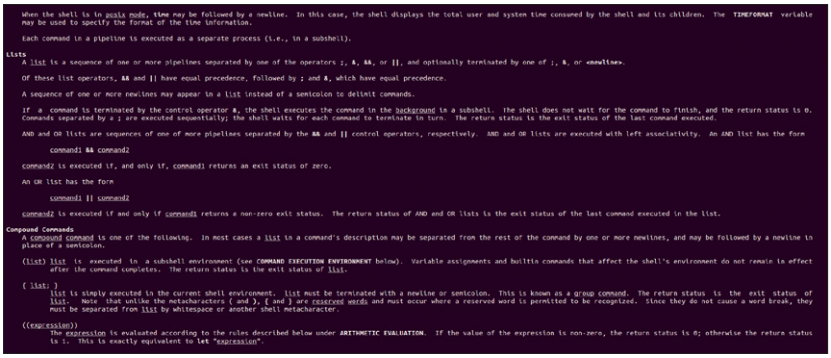
Imagine you just typed something at a shell prompt and pressed Enter. First, the shell splits your input into words. Expansion is the next step, and it comes in several forms.

One expansion type you already know is the pathname expansion. Recall **ls *.txt**? If there are files that end with **.txt** in the current directory, *Bash* puts their names instead of the wildcard. If there are none, ***.txt** remains as is. So, if you forget to escape an asterisk in a command like **find . -name *.txt**, it may or may not work as expected. Besides *, *Bash* also understands **?** (matches a single character) and **[a–z]** (matches any character in a group). With the **extglob** shell option enabled, you get some modifiers as well. Say, you can negate an expansion: **ls !(*.txt)** lists all files that **do not** end with **.txt**.

Brace expansion is like pathname expansion, although "file names" do not have to exist. Curly braces expand to the list of their comma-separated contents. So, **diff -u /etc/foo.conf{,.new}** shows the difference between current config and a new (perhaps, package-provided) one. Curly braces may also enclose a sequence expression:

```
$ echo file_{a..e}.txt
file_a.txt file_b.txt file_c.txt file_d.txt file_e.txt
```

{1..5} would also work, and you can use another pair of dots to specify an optional step increment. Brace expansions occurs before any others, and they can be nested. For example, **ls /usr{/bin/{*.py,*.pl},/sbin/{*.py,*.pl}}** lists Perl and Python scripts under **/usr/bin** or **/usr/sbin**.



When the shell is in posix mode, time may be followed by a newline. In this case, the shell displays the total user and system time consumed by the shell and its children. The TIMEFORMAT variable may be used to specify the format of the time information.

Each command in a pipeline is executed as a separate process (i.e., in a subshell).

Lists

A list is a sequence of one or more pipelines separated by one of the operators ;, &, &&, or ||, and optionally terminated by one of ;, &, or <newline>.

Of these list operators, && and || have equal precedence, followed by ; and &, which have equal precedence.

A sequence of one or more newlines may appear in a list instead of a semicolon to delimit commands.

If a command is terminated by the control operator &, the shell executes the command in the background in a subshell. The shell does not wait for the command to finish, and the return status is 0. Commands separated by a ; are executed sequentially; the shell waits for each command to terminate in turn. The return status is the exit status of the last command executed.

AND and OR lists are sequences of one of more pipelines separated by the && and || control operators, respectively. AND and OR lists are executed with left associativity. An AND list has the form

command1 && command2

command2 is executed if, and only if, command1 returns an exit status of zero.

An OR list has the form

command1 || command2

command2 is executed if and only if command1 returns a non-zero exit status. The return status of AND and OR lists is the exit status of the last command executed in the list.

Compound Commands

A compound command is one of the following. In most cases a list in a command's description may be separated from the rest of the command by one or more newlines, and may be followed by a newline in place of a semicolon.

(list) list is executed in a subshell environment (see COMMAND EXECUTION ENVIRONMENT below). Variable assignments and builtin commands that affect the shell's environment do not remain in effect after the command completes. The return status is the exit status of list.

{ list; }
list is simply executed in the current shell environment. list must be terminated with a newline or semicolon. This is known as a group command. The return status is the exit status of list. Note that unlike the metacharacters (and), { and } are reserved words and must occur where a reserved word is permitted to be recognized. Since they do not cause a word break, they must be separated from list by whitespace or another shell metacharacter.

((expression))
The expression is evaluated according to the rules described below under ARITHMETIC EVALUATION. If the value of the expression is non-zero, the return status is 0; otherwise the return status is 1. This is exactly equivalent to let "expression".

The **bash(1)** man page is a long read, but also an ultimate guide to the powers of *Bash*.

Parameter expansion is perhaps the most frequently used expansion in *Bash*. Does the **$PATH** expression look familiar? It expands to the value of **PATH** variable, and is really parameter expansion in its simplest form. You may enclose parameter name in curly braces. It's optional here, but consider this:

```
$ hello=world
$ echo ${hello}_1
world_1
$ echo $hello_1
```

Braces are often seen in more complex expansions, like **${cmd:-/bin/bash}**. This expands to the value of **cmd** if it is set and non-empty. Otherwise, **/bin/bash** is used. **${cmd:=/bin/bash}** is similar, but it also assigns to **cmd** the value specified. Both constructs are common across command line arguments processing code. Two others, **${parameter#word}** and **${parameter%word}** remove the matching prefix or suffix, as in pathname expansion. They may look counter-intuitive, yet are easy to remember. Hash marks usually come before numbers, so they are for prefixes; percentage signs comes after, and they are for suffixes. By default, the operation is non-greedy (a shortest match is removed); use **##** and **%%** to ask for the longest match instead.

Not all of these may sound practical, so here are some examples. **${path%/*}** is like a **dirname**: it strips everything after the last slash, including it. **${path##*/}** is a reverse: it strips everything up to the last slash, like **basename**. If the path is **/boot/grub/grub.cfg**, the former yields **/boot/grub**; the latter is **grub.cfg**. As you've probably guessed, these expansions are common across path handling code.

Sometimes, you want not to strip, but to substitute some part of the parameter value. Imagine you need to rename all **.JPG** files to **.jpeg**. The **rename** tool will do that, but you may not have it installed. A simple **for** loop does the same in pure *Bash*:

```
for name in *.JPG; do
  mv ${name} ${name/.JPG/.jpeg}
done
```

Here, we look for a constant substring, but it could be a pattern as well.

There are many other expansions in *Bash*. Say you can get a substring instead of substituting it. Or, it is possible to embed basic arithmetic expressions with **$((...))**. *Bash* doesn't support floating point, and if you need it, consider using external tools like **bc**. With command substitution (**$(...)** or just backticks), you can grab **bc** output into a variable: **sine_pi_2=$(echo 's(2. * a(1))' | bc -l)**. Remember not to put spaces around equals sign, or *Bash* would get confused. Many more possibilities are detailed in **bash(1)** under the **EXPANSIONS** section.

Terms and conditions

Sequential scripts are good for the simplest tasks. Anything more or less advanced demands conditional execution and branching. Many general-purpose languages provide one or two constructs (say, **if** and **switch**/**case**) for that purpose. *Bash* offers somewhat more idioms to explore.

Bash branches on whether a command executed successfully (exit code 0) or not:

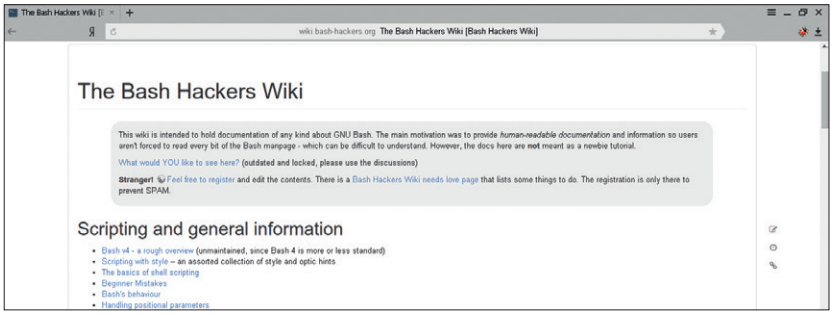
```
if grep -q Linux file.txt; then
  # Linux is in file.txt
fi
```

From here, it feels natural to introduce a specific command that tests conditionals and returns 0 (true) or 1 (false). This command is **test**, and it also has a well-known synonym, **[**. Both are *Bash* built-ins, but you may also find the **[** executable lying around for compatibility reasons:

```
$ whereis [
[: /usr/bin/[ /usr/bin/X11/[ /usr/share/man/man1/[.1.gz
```

When you encounter a construct like **if ["$x" = "yes"]; then ... fi**, remember that brackets aren't special syntax. They are just an ordinary command, albeit **[** forces you to supply the closing **]** (**test** doesn't).

You specify conditions to check as **test** command-line arguments. All basic comparisons are understood, and you can combine expressions with parenthesis **()**, logical **AND** (**-a**) and **OR** (**-o**). **!** negates the expression's value, as in C. Remember that parenthesis have special meaning in the shell, so you'd probably want to escape them (**\(**). Also remember that strings and numbers use different comparison operators: **=** and **!=** (strings) vs **-eq**/**-ne**/**-gt**/**-lt** etc for integers. Perl borrows these semantics, but if your background is in some other language, it could be the source of hard-to-find bugs. Another subtle detail is that you should quote variables. If **x** is unset or empty, **test $x = "yes"** sends **test** two arguments: **=** and **yes**; the first operand is missing. With **test "$x" = "yes"**, it will receive an empty string, **=** and **yes**, which is just false. Should you want to test if something is empty (or non-empty), use **-z "$x"** or **-n "$x"** (remember it like this **"length **z**ero/**n**on-zero"**).



If you are serious about *Bash* programming, remember to bookmark **http://wiki.bash-hackers.org**.

Bash is a special-purpose language, so **test** also provides numerous filesystem-related operators. You can check if the file exists (**-e**) and whether it is a regular file (**-f**) or a directory (**-d**). Other operators check whether the file is readable (**-r**), writeable (**-w**) or executable (**-x**). You can find a complete reference in the man page, but be careful: **test(1)** refers to the external **test** command, whereas *Bash* uses a built-in described in **bash(1)**.

Let's have some examples. **[$i -lt 10]** checks that **$i** (which must be set) is less than ten. Another way to do it is **$((i < 10))**, but it expands to **1** (true) or **0** (false) literally, while **test** yields an exit code. **[-n "$file" -a -x "$file"]** checks that **$file** stores a non-empty string which refers to some executable. For a wrap-up, **["${file#.*}" = "zip"]** checks that the file's extension is **.zip** (case-sensitive). Note that spaces after **[** and before **]** are required.

Often, you want to check that the file exists, and complain if not. While certainly doable with **if**, *Bash* favours a somewhat more compact idiom for such one-liners. The **&&** and **||** shell operators are short-circuit: they only execute operands required to compute the final value. Now, consider **[-e $file] || exit 1**. If the file exists, **[** yields 0, and regardless what **exit 1** will return, the expression will be true. So *Bash* won't execute **exit 1** unless the existence test fails. Similarly, **[-d "$backups"] && cp *.bak $backups** copies backups to some directory, if it exists. **&&** are left associative, so **expr1 && expr2 || expr3** works like **if-then-else**.

Besides the **test** built-in command, *Bash* also supports a conditional expression, **[[...]]**. It's called "new test" sometimes. It also evaluates expressions and returns the status of 0 or 1. So, what's the

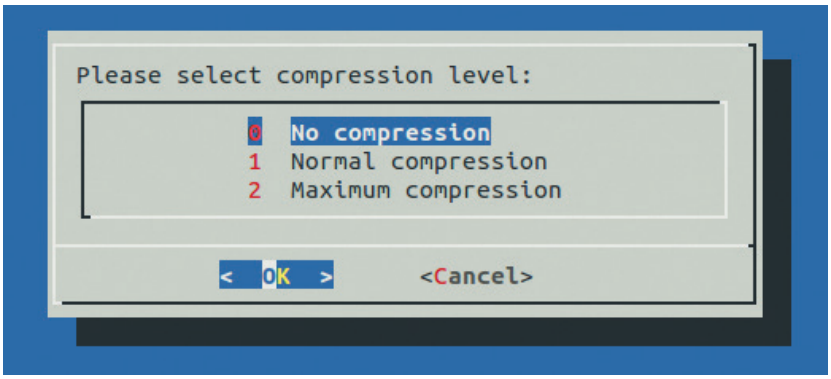
What's the $#@*?

Bash has many predefined variables and special parameters with cryptic names that you may find useful in your scripts. Here are some highlights:

- **$0** | Expands to the name of the shell or shell script.
- **$1**, **$2**, ... **$n** | Positional parameters. Expand to the script or function arguments (see "Functions").
- **$#** | The number of positional parameters.
- **$***, **$@** | All position parameters together. The difference is that the quoted "$*" expands to single word, and "$@" to multiple ones.
- **$?** | Last foreground command exit status.
- **$$** | The shell's process identifier (LV023).
- **$HOME** | The current user's home directory.
- **$PS1**–**$PS4** | Command prompts templates.
- **$RANDOM** | Expands to a random number. Unset and re-create to make it a normal variable.

PRO TIP

The `readonly` keyword marks variables (and functions) immutable at or after creation. This helps to prevent accidental modification of important constants.



Bash scripts aren't solely about the command line – you can make dialog-based interfaces as well.

difference? First, **[[...]]** is not portable – there are shells that don't understand it. Second, it's not a command. It has its own parse context, so you don't have to quote arguments anymore. Third, conditional expressions are more advanced; for example, you can do pattern-matching with the **=** operator: **[[$file = *.zip]]**. Even regular expressions are available with **=~**. If the supplied regular expression is syntactically incorrect (as per **regex(7)**), the conditional expression returns status code 2.

Functions

As in many programming languages, *Bash* lets you wrap reusable pieces of code as functions. Each function stores a sequence of shell commands to execute, and acts much like a command by itself. For instance, the syntax for calling a function and running a command is exactly the same. You supply function arguments the same way you supply arguments for the command. Moreover, positional parameters and **$#** (see the boxout) in a function are overridden to reflect the values and the number of arguments passed to it.

Defining a function in *Bash* is straightforward:

```
function read_pid()
{
  local pidfile
  # TODO: Check $# first
  pidfile=/var/run/$1.pid
  if [ -r "$pidfile" ]; then
    # do something
  else
    return 1
  fi
}
# usage: read_pid apache2
```

There are a few things to note here. First, the leading **function** keyword is optional; **read_pid() { ... }** works just the same. Second, one can define function-local variables with the **local** keyword to prevent name clashes. Local variables aren't shared between the function and outer context, even if they have the same name. Third, the **return** keyword is used to terminate the function; the value you pass to it is returned as a status code. If there is no **return**, the function exits successfully (status code 0). Note that *Bash* functions (as well as commands) can't return strings or any other types. To emulate this behaviour, you can **echo** in a function and grab its standard output with command substitution, like this:

```
say_hi() { echo hi; }
if [ "$(say_hi)" != "hi" ]; then
  # oh, really?
fi
```

Some functions are specific to the containing script; others are more general-purpose. The latter are often organized as libraries. In *Bash*, a library is a mere shell script which doesn't happen to contain anything except functions and (maybe) some bootstrapping code. There are no namespaces, imports, header files or whatever. Your system is likely to have a number of *Bash* libraries lying around. For example, check if **/lib/lsb/init-functions** exists, and peek into it. You'll see functions like **start_daemon()** or **pidofproc()** defined inside. Traditional Unix initscripts rely on these to start daemons and perform other tasks in a consistent fashion. Try **grep -R init-functions /etc/init.d** to see how the library is included. *Bash* provides the **source** keyword for that, but an idiomatic way is to use **. /path/to/script.sh**, which is a synonym.

Bash also calls some predefined functions in response to various events. If you ever worked with Ubuntu, you may recall a friendly note appearing when you type a command that is not currently installed:

```
$ emacs
The program 'emacs' can be found in the following
packages:
 * emacs24
 * ...
Try: sudo apt-get install <selected package>
```

(yup, I don't have *Emacs* on my laptop). When *Bash* is unable to find a command in **$PATH**, it invokes the **command_not_found_handle()** function, passing the original command line as the function arguments. In Ubuntu, it calls into the **command-not-found** tool (see it yourself with **type command_not_found_handle**), which does all heavy lifting. It's easy to author a poor man's variant though:

```
$ command_not_found_handle() { echo "Sorry pal, you
```

Shell options

Bash understands quite a few options affecting the shell's operation. We already met one of them, **extglob**, at the beginning of this Core Tech. Its cousin, **dotglob**, includes hidden files (they start with .) in pathname expansion, so wildcards like * cover them as well.

Other options pertain to usability. **cdspell** tries to correct minor spelling errors (like missing or transposed characters) in directories you **cd** into. With **autocd**, you can omit **cd** altogether: typing just directory name will be enough. **progcomp** enables programmable completion you call with the Tab key.

To control all these options, use the **shopt** built-in command. When called with no arguments, it prints valid options and their statuses (on or off). You can also see which options are enabled for the current session in the **$BASHOPTS** variable. To enable (or set) an option, use **shopt -s optname**; **shopt -u optname** does the reverse.

```
don't have $1 installed"; }
$ emacs
Sorry pal, you don't have emacs installed
```

Your script can even define custom signal handlers. *Bash* provides **trap** built-in for these purposes. The syntax is like this:

```
handle_sigusr1() { echo "Got SIGUSR1"; }
trap handle_sigusr1 SIGUSR1
```

To see which handlers are currently defined, use **trap -p**. Handler could be any shell command, not necessarily a function, although the latter is a common choice. The trap mechanism extends beyond Unix signals. **DEBUG** is breakpoint-like trap which executes before each command in a script. You may use it to watch how selected variables change during the script's lifetime. **RETURN** is called when a function returns. **EXIT** – you guessed it – runs on the shell exit, which is handy for cleanup tasks.

Arrays

Believe it or not, *Bash* also supports arrays. Arrays in *Bash* are one-dimensional, yet *Bash 4* introduced support for associative arrays (or hashes). Numeric array indexes start at 0, as you'd expect. Associative arrays are indexed with arbitrary strings. There is more than one way to create an array in *Bash*:

```
x[0]="The first one"
declare -a y=("And this is the second" "And the 3rd")
declare -A z=([roses]=red [sky]=blue)
```

Explicit declaration is required only for hashes, indexed arrays are usually created in place. To retrieve an element, use **${x[0]}** or **${z[roses]}**; braces are required. **"$y[*]"** and **"$y[@]"** expand to all elements in the array: either as one word (the former) or multiple words (the latter):

```
$ for i in "${z[*]}"; do echo $i; done
red blue
$ for i in "${z[@]}"; do echo $i; done
red
blue
```

Similarly, **${!y[@]}** expands to all keys (or indexes) in the array. **${#z[@]}** yields the number of elements.

Want someone to explain what the command does? **www.explainshell.com** is here to answer.

Arrays aren't something you do in *Bash* every day. You don't want an array when you just need to iterate over a set of elements: use a string and let *Bash* do word splitting for you. Yet you may find arrays useful in more complex scenarios. The following example comes from a fictional backup system which asks the user to choose compression level:

```
declare -a comp_opt=("tee" "bzip2" "xz")
tempfile=$(mktemp)
dialog \
 --menu "Please select compression level:" \
 10 50 8 \
 0 "No compression" \
 1 "Normal compression" \
 2 "Maximum compression" \
 2> "$tempfile" || exit 1
choice=$(<"$tempfile")
tar --create $src | ${comp_opt[$choice]} >$backup
```

dialog is a common tool to create interactive text-mode dialogs in shell scripts. In this case, it prints either 0, 1 or 2 to **stderr**. We redirect this stream to a temporary file and read it back as **dialog** finishes to learn the user's choice. Finally, we call **tar** and pipe its output to the desired command. **tee** is used to pass the tarball as is if no compression was chosen. **$src** and **$backup** are presumed to have their values set outside this snippet.

Command of the month: shellcheck

How do you hunt for a bug in a *Bash* script, if it doesn't behave the way intended? Perhaps the most popular answer would be "with **echo**s", but *Bash* also has some debugging aids and source code checkers. You already know about the **DEBUG** trap, but your first stop should be **set -ex**. **set** is a built-in command to tweak various low-level options. **-e** asks *Bash* to exit as soon as an error occurs (so you'd be certain where it was). **-x** prints source lines of your script expanded. Expansions could be tricky, but with **set -x** you'll know what *Bash* tries to run. Just add **set -ex** to the beginning of your script, and you'll be able to trace its execution from *Bash*'s point of view.

You'd probably agree that preventing a disease is better than curing it. Many languages have static analysers or similar tools which detect "smells" in your code. These could be common mistakes, or constructs that are likely not to work the way you think, code that is known to break under certain conditions and so on. *Bash* also has one: *ShellCheck*. It's written in Haskell, and you may have tough times compiling it. Better use your package manager or give the whole thing a try online. Visit **www.shellcheck.net**, paste your script, push the button – and you'll get detailed analysis within seconds! Now you have one less excuse for not quoting your variables properly.

/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!

Normally I blather on here about how other people are idiots and they are doing things wrong. Maybe it is sleep deprivation, or the very nice gin I picked up in South Africa, but I think I want to break with tradition and blather on about things that people have been doing right!

I have been playing around with the Particle (formerly Spark) Photon for a while and while I like it (a lot) it does annoy me a tiny tiny bit. The Photon, in case it passed you by, is basically a tiny Wi-Fi enabled Arduino-alike. In fact if anyone can be said to have made a "better Arduino" this is probably it.

Cunning firmware means, through its cloud portal, you can do Wi-Fi updates to your device, which is very handy if your device now happens to be in the attic somewhere under a box where a scary spider lives. But I will still have to brave the spider and find the right box when it comes to change the batteries. Boo!

The problem with Wi-Fi is that it uses power. You don't have to have it on all the time, but even turning it on and off to send data to the cloud and listen for updates wastes precious ergs. That's why I got a Photon-sort-of-compatible Redbear Duo too (**https://goo.gl/jlzCfX**).

A Bluetooth LE radio in addition to Wi-Fi means I can send data on minuscule amounts of power, and still turn on Wi-Fi to reflash the device if I want. The similarish Bluz DK (**http://bluz.io**) does away with Wi-Fi altogether, and uses a separate gateway device (basically a Redbear Duo) to sync with the Particle cloud. It may be years before I have to see that spider again...

MY LINUX SETUP
PETE CAVE

Readers, send us words and pictures of your Linux dens!

Q What version of Linux are you currently using?

A Linux Mint, Manjaro and OpenSUSE 13.1.

Q And what desktop are you using at the moment?

A Mate, Xfce, and I loved #crunchbang [which used the Openbox window manager].

Q What was the first Linux setup you ever used?

A Ubuntu 2004. Can't remember why I didn't stick with Knoppix.

Q What Free Software/open source can't you live without?

A *LibreOffice*, *Gimp*, *Bluefish*, *Filezilla*, the *Firefox* web developer add-ons and *Inkscape*.

Q What do other people love but you can't get on with?

A I only stuck with Unity for months, because of setting everything up again. (If I could save my settings easily, I would have changed distro within two weeks.) I'm trying to improve my spelling, writing and my of reading speed, and Unity does not help. I love a good menu.

LINUXVOICE

This is what we've done in the last 24 issues. Subscribe to the next 12 from just £38.

Every subscription includes access to every PDF, ePub and audio edition we've ever published.

shop.linuxvoice.com


LV