WIN £4,250 OF HACKER SWAG: DETAILS INSIDE

LINUXVOICE

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

XMAS BONANZA!

Our festive gift guide for the discerning geek

HACK A BLUETOOTH MONITOR p76

Try more Linux distros

Get a prettier desktop

WHEN DEVELOPERS FALL OUT! p28

USE LINUX SMARTER

Keep tabs on your system

Serve your own website

Keep on top of updates

Streamline the way you work

MODEL THE GLOBAL ECONOMY p84

32 PAGES OF TUTORIALS

GIANUGO RABELLINO

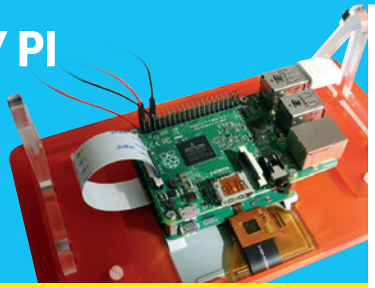
MR MICROSOFT

The senior director of open source communities at MS on the importance of being open

QUIZMASTER

RASPBERRY PI

Build your own Python-powered touchscreen quiz machine for glorious learning



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

MYSQL › FEDORA › UBUNTU › PYTHON & MORE!

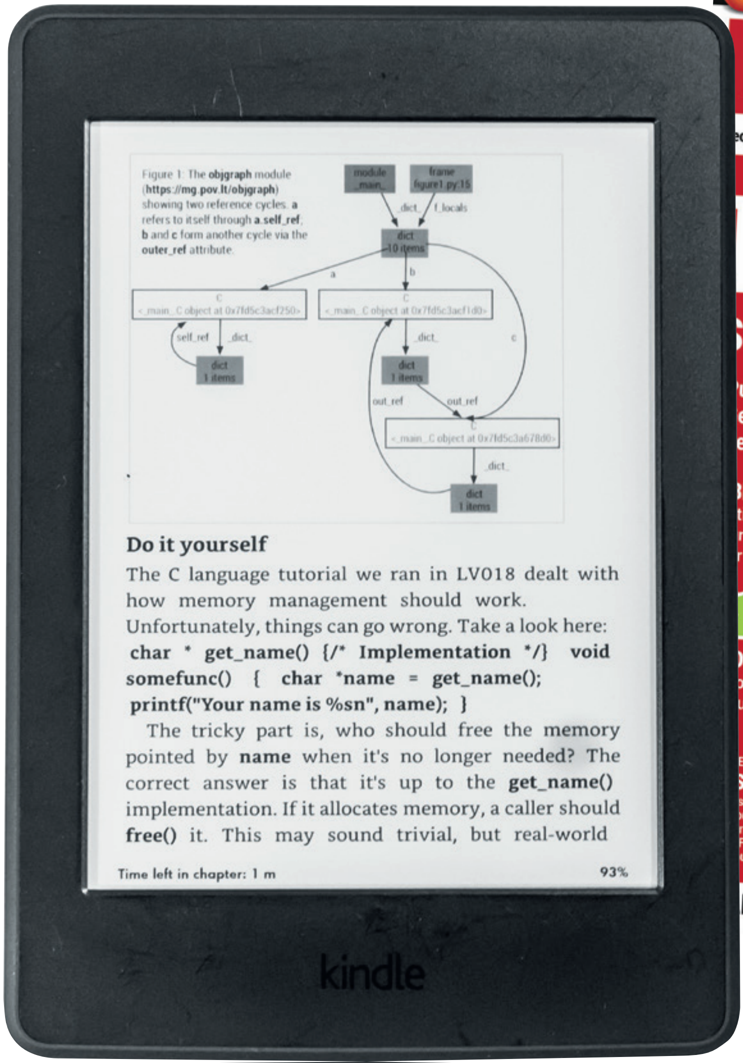


Figure 1: The **objgraph** module (https://mg.pov.lt/objgraph) showing two reference cycles. **a** refers to itself through **a.self_ref**; **b** and **c** form another cycle via the **outer_ref** attribute.

Do it yourself

The C language tutorial we ran in LV018 dealt with how memory management should work. Unfortunately, things can go wrong. Take a look here:
char * get_name() {/* Implementation */} void somefunc() { char *name = get_name(); printf("Your name is %sn", name); }

The tricky part is, who should free the memory pointed by **name** when it's no longer needed? The correct answer is that it's up to the **get_name()** implementation. If it allocates memory, a caller should **free()** it. This may sound trivial, but real-world

Time left in chapter: 1 m 93%

FIREFOX: DISCOVER ITS HIDDEN FEATURES
LINUXVOICE
ecember 2015
FREE SOFTWARE | FREE SPEECH
www.linuxvoice.com
The best kids' software
Build a DIY Oculus Rift
FREEDOM!
Switch to 100% Free Software today
urge your PC of proprietary software
et complete privacy and security
e free as Richard Stallman intended!
BREBOOT X200
tified by the Free Software ndation – could this be r next laptop?
DROID ANATOMY
over the incredible power ur pocket [ooh behave!]
32 PAGES OF TUTORIALS
SON RANDAL
sident of the urce Initiative rl, linguistics, Free Software/ urce schism
DESKTOP
GNOME 3.18
You're going to love the shiny, slick desktop environment that beats Apple's OS X at its own shiny, slick game
BASES › GITLAB › GARBAGE › ADA & MORE!
ISSN 2054-3778

VOICE OF THE MASSES

The January 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.

On our podcast (which is seven years old in February!), there's a section called 'Voice of the Masses'. This is where we ask our listeners a question. This question often starts off innocuous but the answers always surprise us with their insight and positivity. One of the best examples of this happened recently, when we asked, "Who is your Linux or Free Software hero?"

What surprised us most was that out of the 60 people proposed in the replies, there was only one mention of Linus Torvalds, and only three or four of Richard Stallman. The majority were for the unsung heroes behind much of the software we all use every day: Fabrice Bellard,for his work on *Qemu* and *FFmpeg*; Martin Gräßlin for speaking calmly in a KDE storm; and even Mark Shuttleworth for bringing Linux to the masses. But to even highlight these few is to miss the point – the best thing about Linux? It's built by all of us.

Graham Morrison
Editor, Linux Voice

THE LINUX VOICE TEAM
Editor Graham Morrison
graham@linuxvoice.com
Deputy editor Andrew Gregory
andrew@linuxvoice.com
Technical editor Ben Everard
ben@linuxvoice.com
Editor at large Mike Saunders
mike@linuxvoice.com
Games editor Michel Loubet-Jambert
michel@linuxvoice.com
Creative director Stacey Black
stacey@linuxvoice.com
Malign puppetmaster Nick Veitch
nick@linuxvoice.com
Editorial contributors:
Mark Crutch, Andrew Conway, Juliet Kemp, John Lane, Vincent Mealing, Simon Phipps, Les Pounder, Mayank Sharma, Valentine Sinitsyn.

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

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

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

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

What's hot in LV#022

ANDREW GREGORY

We haven't put a number on it, but our collection of tips is numerous and diverse, with people like Matthew Garrett and Matthias Kirschner contributing their all-time favourites.
p14

BEN EVERARD

Valentine's in-depth look at how Linux works is becoming compulsive reading. This month, he deconstructs and reconstructs the humble executable, which is something we use every day.
p94

MIKE SAUNDERS

We've not run a competition before, but we can't help being excited about the pirate booty we've got from Pimoroni. All you've got to do is find 10 penguins!
p26

SUBSCRIBE ON PAGE 56

Contents

Welcome to Linux Voice, the magazine that gives back to the community

SUBSCRIBE ON PAGE 56

WIN STUFF WORTH £4,250!

TURN TO PAGE 26!

NEWSANALYSIS

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

Opinion

The flood of foundations

Some companies like impartial supervisory bodies so much, they're creating their own!

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.

Of late, there seem to have been a tidal wave of new "open source foundations" appearing in the technology industry – there's the Node.js Foundation, Cloud Foundry Foundation, Cloud Native Computing Foundation and the OpenStack Foundation to name but a few. What is going on?

The first thing to observe is that there are two different kinds of entities that call themselves "open source Foundations". Some – like the Apache Software Foundation or the Document Foundation – are public benefit organisations, with a mission and bylaws that drive them to act in the interests of the public at large. The other kind – like all the examples I gave above, as well as better-known bodies like the Linux Foundation and the Eclipse Foundation – are actually trade associations, with a mission and bylaws that expect them to act in the interests of their members rather than of the general public. Almost all of the proliferation is in this latter category.

What is the value of a Foundation? In both the public benefit and trade association cases, there are clear benefits when a project has a large, diverse community. All the most important freedoms – to use the software for any purpose, study and improve it and share with anyone – are secured by using an OSI-approved open source licence. Any project that doesn't clearly point to the source code and identify how it's licensed is definitely a problem. But with that taken as read, a Foundation offers:

- An "Asset Lock", guaranteeing that community assets can only be used in ways the community approves (including trademarks and copyrights).
- A "bank", handling donations, paying staff and fulfilling tax-reporting obligations.
- An impartiality guarantor, anchoring the representation of its community and ensuring decisions are made in the way that the community wants independently of any one participant.
- An infrastructure provider, hosting code, mailing lists, forums and bug trackers and also hosting events.

So why do companies prefer trade associations as the vehicle for this, rather than public benefit charities? I liken it to the way technology companies responded to open standards in the 80s. Originally introduced as a way to stem the control of monopolistic mainframe companies over their markets, standards organisations increasingly became the domain of corporate politicians. Even nominally national standards bodies like BSI or international ones like ISO are actually occupied by career technology politicians employed by the largest corporations. The result has been regulatory capture – the mechanism invented to regulate the power of corporations has transformed into a medium for them to express their competitive goals and especially to chill new entrants to their markets.

A well designed foundation keeps a strong separation between the fiduciary responsibilities and the technical work

Corporate interests

Trade associations in open source are evolving in a similar way. While open source projects were originally grass-roots collaborations between individual experts, their disruptive force has led the corporate targets of that disruption to invest not just in technology but in the politics surrounding it. The new giant open source "Foundations" are high-stakes political venues with big entry tickets. Individuals still have a role in the technical work, but the overall strategy is a thing of smoke-filled rooms. Open source trade associations provide the ideal vector for the equivalent of regulatory capture in open source.

That's not to say they are all bad. A well-designed one (the Eclipse Foundation for example) keeps a strong separation between the fiduciary responsibilities and the technical work, and only allows the members to buy in to the former, as well as expecting those requesting higher status to commit to investing developers in the technical work. That's not to say all public benefit open source charities are perfect. Even the best designed one – the Apache Software Foundation – has been conspicuously gamed by corporate forces on multiple occasions.

So take care to disambiguate the term "Foundation", and encourage your employer not to start another one if that's what they are thinking. Join an existing Foundation – there are several of both flavours that accept new projects – or if they really must make a new one, seek specialist advice and focus first on software freedom. Remember, "Simon says ... no new Foundations!"

LibreOffice • Kubuntu • Red Hat + Ansible • ZFS • Pis in Space • KDevelop

CATCHUP

Summarised: the biggest news stories from the last month

1 New LibreOffice respin wins government support

The flagship open source office suite keeps going from strength to strength. Collabora Ltd has created a version called *GovOffice* with extra migration tools, deployment features and long term support, and the UK government has said it will "provide public sector organisations with savings on open source office software". If ODF file formats become the norm in the government, we'd be happy pandas. **www.collaboraoffice.com/collabora-govoffice.php**

2 Kubuntu head honcho Jon Riddell stands down

Jonathan Riddell founded Kubuntu back in 2005, and has grafted away over the last decade to establish it as one of the forefront desktop Linux distros. But in recent years he has expressed dissatisfaction with Ubuntu parent company Canonical for its handling of IP policies – to the point that the Ubuntu Community Council wanted to boot him out. With the release of Kubuntu 15.10, Riddell has left the project, deriding Ubuntu as a project that "won't obey its own rules".

3 Red Hat buys Ansible

Enterprise Linux giant Red Hat has snapped up Ansible, makers of the eponymously named IT automation software. Red Hat's goal with Ansible is to create "frictionless IT": **http://tinyurl.com/qjrr8dh**

4 ZFS to be included in Ubuntu as standard

Originally developed by Sun for its Solaris operating system, the ZFS filesystem and logical volume manager has since seen widespread usage in other Unix flavours, most notably FreeBSD. It features support for huge volume and file sizes, data corruption protection, snaphots and other snazzy features. Until now it hasn't been included as standard in many distros, but Canonical boss Mark Shuttleworth has said it will be available for all to try in upcoming Ubuntu releases.

5 DRM coming to JPEGs?

Officially "Digital Rights Management", but more commonly referred to in the FOSS world as "Digital Restrictions Management", DRM aims to stop people sharing information. Now the JPEG Privacy and Security group is investigating ways to add DRM to JPEG images – stopping you from copying or saving images you see on the web. We think DRM is simply ineffective and a waste of time, and the Electronic Frontier Foundation has already started campaigning against it: **http://tinyurl.com/op4lzdw**

6 First beta release of KDevelop 5 available

It's been over a year in the making, and *KDevelop 5.0* is inching ever closer to release with the first beta. The code base has been ported to *Qt 5* and KDE Frameworks 5, while the old C++ parser has been replaced by a much more powerful one from the *LLVM/Clang* project. Semantic language support for QML and JavaScript has been rolled in, and we may even see *KDevelop* releases on Windows and Mac OS X at some point. **www.kdevelop.org**

7 Raspberry Pis heading to the Space Station

As if the Pi wasn't already popular enough here on Earth, two units are now jumping onto a rocket for a stay on board the International Space Station. The Raspberry Pi Foundation recently ran Astro Pi, a competition to give UK school students the opportunity to develop experiments to run on Pis aboard the ISS. Seven experiments have been selected and will be carried up to the ISS with ESA astronaut Tim Peake on a shiny Soyuz 45S. **http://tinyurl.com/pisinspace**

8 Element14 to build custom Raspberry Pis

In other Pi news, distributor Element14 has created a new Customisation Service for the dinky single-board computer. If you're willing to order at least 3,000 units of your custom design, you can reconfigure the board layout, add components such as Wi-Fi or extra I/O pins, and even add onboard flash memory. Element14 will guide you through the process and show you what's doable (and what's not). See here for more details: **www.element14.com/custompi**

DISTROHOPPER

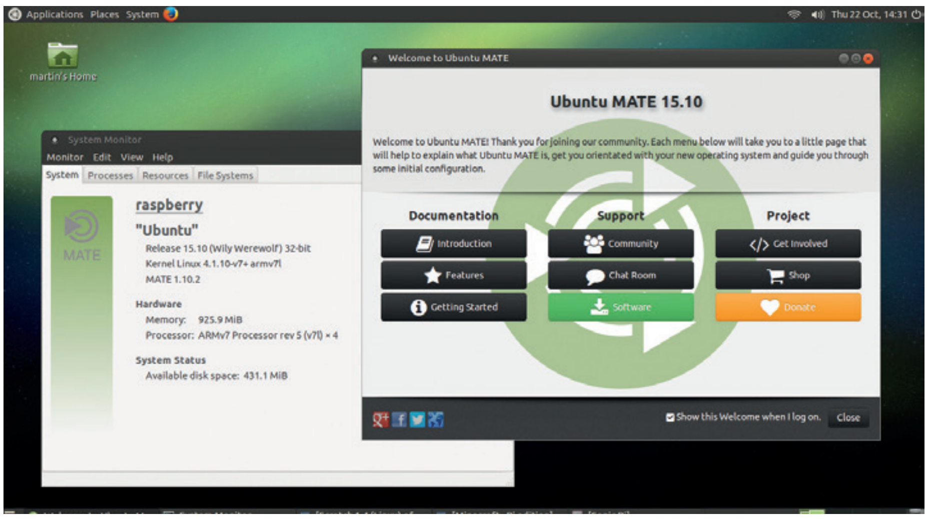
What's hot and happening in the world of Linux distros (and BSD!).

(K/X/L)ubuntu 15.10

News from the spin-offs.

Yes, it's that time of year again: a new Ubuntu release is here. Ben's review over on page 43 focuses on the main distribution, but here we'll explore the spin-offs. Kubuntu 15.10 features a snazzy KDE Plasma 5.4 desktop along with KDE Applications 15.08, while *LibreOffice 5.0* and *Firefox 41* make up the major non-KDE software components. It's available as a 1.3GB DVD ISO download and system requirements are a minimum of 1GB RAM and 10GB hard drive space (although we'd say double the RAM is much more sensible).

Meanwhile, Xubuntu 15.10 arrived at the same time and has also cranked up its memory requirements (1GB recommended) by dropping the lightweight *AbiWord* and *Gnumeric* applications in favour of *LibreOffice*. Xubuntu has always presented itself as a RAM-friendlier alternative to (K) Ubuntu, but *LibreOffice* is getting snappier and lighter. Xubuntu 15.10 also includes Xfce Panel Switch, making it easier to back up and restore panel layouts.



Ubuntu MATE 15.10

Welcome to Ubuntu MATE! Thank you for joining our community. Each menu below will take you to a little page that will help to explain what Ubuntu MATE is, get you orientated with your new operating system and guide you through some initial configuration.

raspberry
"Ubuntu"
Release 15.10 (Wily Werewolf) 32-bit
Kernel Linux 4.1.10-v7+ armv7l
MATE 1.10.2

Hardware
Memory: 925.9 MiB
Processor: ARMv7 Processor rev 5 (v7l) × 4

System Status
Available disk space: 431.1 MiB

☑ Show this Welcome when I log on.

Here's Ubuntu Mate 15.10 on a Raspberry Pi 2. Don't expect stellar performance, but it is usable.

Over in Lubuntu land, the new distro release is an evolutionary affair as the team prepares to move to the LXQt desktop in 16.04. Some LXDE components have been updated and artwork has been improved, but otherwise there's not a lot to write home about. Oh, and let's not forget Ubuntu Mate 15.10, the spin-off that uses the Mate desktop, a continuation of the Gnome 2.x codebase. This release includes a version optimised for the Raspberry Pi 2. The team has put in a lot of work to make the distro run smoothly on the Pi, so you can even use the dinky device as a general-purpose desktop.

CentOS goes 32-bit

No, this isn't a step backwards – it actually makes a lot of sense.

CentOS, the community-supported respin of Red Hat Enterprise Linux, went 64-bit only with version 7. This made sense for most use cases, as 64-bit CPUs from AMD and Intel have been the norm for many years now. But there have been calls from some CentOS users for a 32-bit version that's more suitable for older machines. But it's not just about decade-old hardware. 32-bit processors are still doing the rounds, such as the Intel Quark system-on-a-chip. We can expect to see more of these CPUs in everyday life thanks to the much-hyped "Internet of Things", so it's useful to have modern, mature and stable distros such as CentOS to run on them. 32-bit CPUs are more than capable enough for most tasks, especially if you don't need access to more than 4GB of RAM.

And then there's more: supporting a wider range of CPUs can often make it easier to finds bugs and security holes, as the OpenBSD project has found. The 32-bit CentOS 7 release is the work of the project's AltArch Special Interest Group, and you can find out more information, including potential bugs, over at **https://wiki.centos.org/SpecialInterestGroup/AltArch/i386**.

CentOS 7's 32-bit port is a boon for users of SoC development boards, and could help to identify tricky bugs too.

News from the *BSD camps

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

OpenBSD 5.8 arrived in the middle of October, sporting an impressive range of updates all over the system. Along with the usual set of new hardware drivers and performance tweaks, there have also been many improvements to the miniature *httpd* web server that replaced *Nginx* in the base system. It now supports pattern matching and redirections via Lua, along with HTTPS HTTP Strict Transport Security. A new **doas** utility replaces **sudo** and provides enhanced security by being much simpler, while OpenSSH is included in this release and has a new default cipher.

Over in the FreeBSD camp, the team has produced its latest quarterly status report – and the longest one ever written, reflecting the overall good health of the project. *Bhyve*, the FreeBSD hypervisor, has seen a lot of work including support for external firmware, which allows it to run Illumos (a fork of OpenSolaris) and Windows in headless mode. The *LLVM/Clang* toolchain has been updated to version 3.7.0, while a handful of developers have grafted away on support for the Acer C720 Chromebook. Almost everything works in FreeBSD now, making the machine an ideal little laptop for hacking on the go. See **http://tinyurl.com/pdo35u5** for the full report. Finally, we should give a mention to NetBSD 7.0. Progress towards this final release has been slow, with RC1 arriving back in June, but 7.0 brings a stack of new goodies including ARM multiprocessor support, accelerated graphics on x86 boxes using Intel and Radeon chips, and a new port for Psion EPOC PDAs. (Yes, NetBSD will run on almost anything containing a CPU, or someone is working on porting it.) Most notably for us Linux users, NetBSD 7.0 now runs on the Raspberry Pi, providing an alternative Unix flavour to our familiar Raspbian.

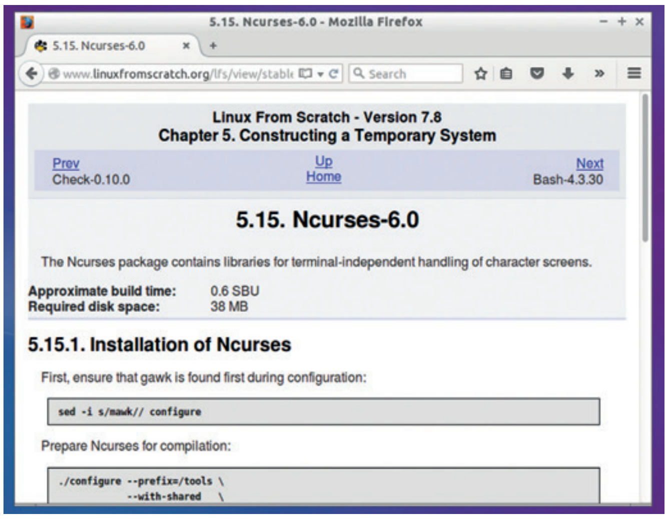
Got an old Psion in the loft doing nothing? Get NetBSD 7.0 running on it!
(Image credit: http://tinyurl.com/nojmnpy)

Linux From Scratch 7.8

If you really want to understand how Linux works – and more specifically, how a distribution is put together – you should spend some time with Linux From Scratch. As its name suggests, it's all about creating a Linux installation from the bare components, with no fancy graphical installers or setup scripts to help you on your way. Linux From Scratch (LFS) isn't a piece of software but rather a book that explains the process step-by-step.

And it's a fascinating process to follow. You start off by using an existing Linux installation to create a new LFS partition, into which you download some low-level toolchain components (such as a C compiler) and build them. From here you add system tools and libraries to the point of having a workable – albeit very rudimentary – Linux installation. It teaches you an enormous amount about how a Linux system boots, what all of the low-level components do, and of course it provides you with ample opportunity to tweak settings and customise the result. Expect to spend many hours working through the various steps, but it's worth it.

Linux From Scratch 7.8 was released in early October and includes updates to 30 packages including *GCC*, *Glibc* and *Binutils*. In addition, there's a spin-off of the book using *Systemd* as the init system, although the main book still currently focuses on *Sysvinit*. You can read the book online at **www.linuxfromscratch.org** or download a bzipped archive for offline reading. Once you've built an LFS system, try some of the other books on the website, such as BLFS (Beyond Linux From Scratch).


5.15. Ncurses-6.0 - Mozilla Firefox
5.15. Ncurses-6.0
www.linuxfromscratch.org/lfs/view/stable
Search

Linux From Scratch - Version 7.8
Chapter 5. Constructing a Temporary System

Prev Check-0.10.0 | Up Home | Next Bash-4.3.30

5.15. Ncurses-6.0

The Ncurses package contains libraries for terminal-independent handling of character screens.

Approximate build time: 0.6 SBU
Required disk space: 38 MB

5.15.1. Installation of Ncurses

First, ensure that gawk is found first during configuration:

```
sed -i s/mawk// configure
```

Prepare Ncurses for compilation:

```
./configure --prefix=/tools \
            --with-shared   \
```

LFS explains exactly what each component does, how much time it takes to compile, and how to build it.

YOUR LETTERS

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

STAR LETTER

ANTI-SEX LEAGUE

The simplest test of a search engine's openness is a search for 'sex'. I don't say it as a prurient interest, just as a search engines willingness to allow you to find what you're looking for unfiltered, unchaperoned if you will.

Much like the differences between Linux and other operating systems in its willingness to allow you to change aspects of the OS to suit your needs or interests. The more adaptable an OS is, the better suited it may be to one's specific needs and interests.

Here in America, you go into the library and ask the reference librarian a question, you get an answer. You do not get, "Why do you want to know?" or "I'm sorry we don't provide that information." or "Could you be more specific?". Maybe search engines are good for the lion's share of searches. When it comes to making moral (or market) choices about what constitutes acceptable, there's a lot to be desired. And if they are making moral (or market) choices, their accuracy as well as usability is in question.

Granted, the morals of the Middle East, Korea, China and elsewhere around the world (including the US) are as different as night and day. A search engine that attempts to please ALL in return for market share ultimately winds up not really pleasing anyone.

I'm an adult. I'm not telling you how to live your life. I don't want a search engine that tries to tell me how to live mine. If I'm not breaking any laws, I should be able to use a search engine to the fullest extent possible – unrestricted. Much the same as the Linux philosophy. Roll your own.

Mike Moore, USA

Oh, and those dastardly Europeans are interfering with our sovereignty again by outlawing default internet content filtering. Perfidious Brussels!

Andrew says: Well, quite. I'm open to the idea that consenting adults can put pictures of themselves on the internet in various states of undress, but I don't want to see it when I haven't had my breakfast, so some degree of insulation is probably OK – Google's safe search is a good idea for most people, most of time. But who's to say what's Google's idea of not safe? Or David Cameron? Or the Chinese government?

I bought a book a few years ago called *How to Build an Atomic Bomb and Other Weapons of Mass Destruction*. It's not beyond the realms of possibility that some people, in certain states, would have a search for this term filtered, or monitored, or brusquely investigated by men in uniform 5 in the morning. Thankfully I paid cash, so there's no way the UK security services know I have it in my possession. It's very good, but I have yet to use it to build anything.

IT'S THAT BIRD AGAIN

This is the second issue of LV where the subject of the penguin logo has come up. I would like to try and put this to rest and so I respond to Maurice George (LV019).

I have been hacking Linux since 1992 when Linux was only a green cursor blob on the computer screen. Now I am only going by memory but I believe that Linus Torvalds chose the penguin because they are the only creatures that do not have a leader but rely on each other and the penguin community for survival. They are also unafraid of humans. Does not the mighty penguin resemble what Linux is all about? Happy, content, relying on each other for support... there's no boss or leader but a very close knit community helping each other. Finally, should not Linus Torvald be the one to decide if the logo should be changed?

Eugene Wong

Graham says: Right, that's enough of this penguin stuff. Please let's just move on as a society.

DOCUMENTATION

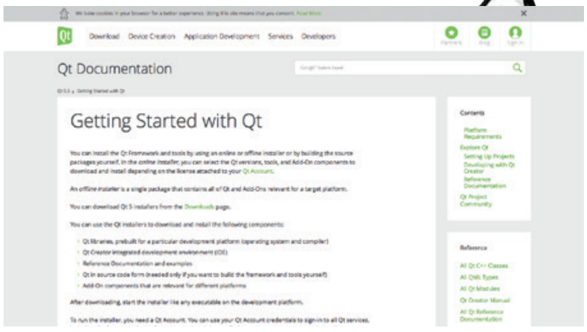
I was a little disappointed when I came to the end of your article on *Syncthing* as it did not include the idiots' guide to setting up on a headless computer/server, however the *Syncthing* documentation and forum answers are really good and, I think, worthy of note.

I wondered if a documentation rating would be a useful addition to the magazine. Would it prompt some others to improve their documentation for the benefit of as aspirant nerds? Also, I would be grateful for the occasional sidebar of explanation with some of the more advanced articles and; and what about a small glossary to your interviews with really clever buggers?

Don't want much then?

Thanks for the intro to *Syncthing*, should have it running this weekend once I have sorted Bug #720.

Paul, Farnham



We keep cookies in your browser for a better experience. Using this site means that you consent. Read More

Download Device Creation Application Development Services Developers

Qt Documentation

Qt 5.5 › Getting Started with Qt

Getting Started with Qt

You can install the Qt Framework and tools by using an online or offline installer or by building the source packages yourself. In the online installer, you can select the Qt versions, tools, and Add-On components to download and install depending on the license attached to your Qt Account.

An offline installer is a single package that contains all of Qt and Add-Ons relevant for a target platform.

You can download Qt 5 installers from the Downloads page.

You can use the Qt installers to download and install the following components:

- Qt libraries, prebuilt for a particular development platform (operating system and compiler)
- Qt Creator integrated development environment (IDE)
- Reference Documentation and examples
- Qt in source code form (needed only if you want to build the framework and tools yourself)
- Add-On components that are relevant for different platforms

After downloading, start the installer like any executable on the development platform.

To run the installer, you need a Qt Account. You can use your Qt Account credentials to sign-in to all Qt services.

Contents

Platform Requirements
Explore Qt
Setting Up Projects
Developing with Qt Creator
Reference Documentation
Qt Project Community

Reference

All Qt C++ Classes
All QML Types
All Qt Modules
Qt Creator Manual
All Qt Reference Documentation

Qt is the gold standard, the Rolls Royce, the Duisenberg of Linux documentation.

Andrew says: I've always considered it part of our job to compensate for the often rubbish state of documentation in Free Software by providing documentation ourselves. It's a gap in the market, if you like, and it's one reason we've never done a tutorial on developing with *Qt Creator*, for example: the documentation is already excellent, and, crucially, it's easy to find a definitive guide. Likewise *Syncthing*.

Some sort of comparison of documentation would be a worthwhile feature, if we did it right – most Free Software developers provide their work free of charge, and it's wrong to stamp our tiny feet when the docs aren't very good. We need to remember that, and be constructive. Regarding glossaries in interviews: I like it. Thanks!

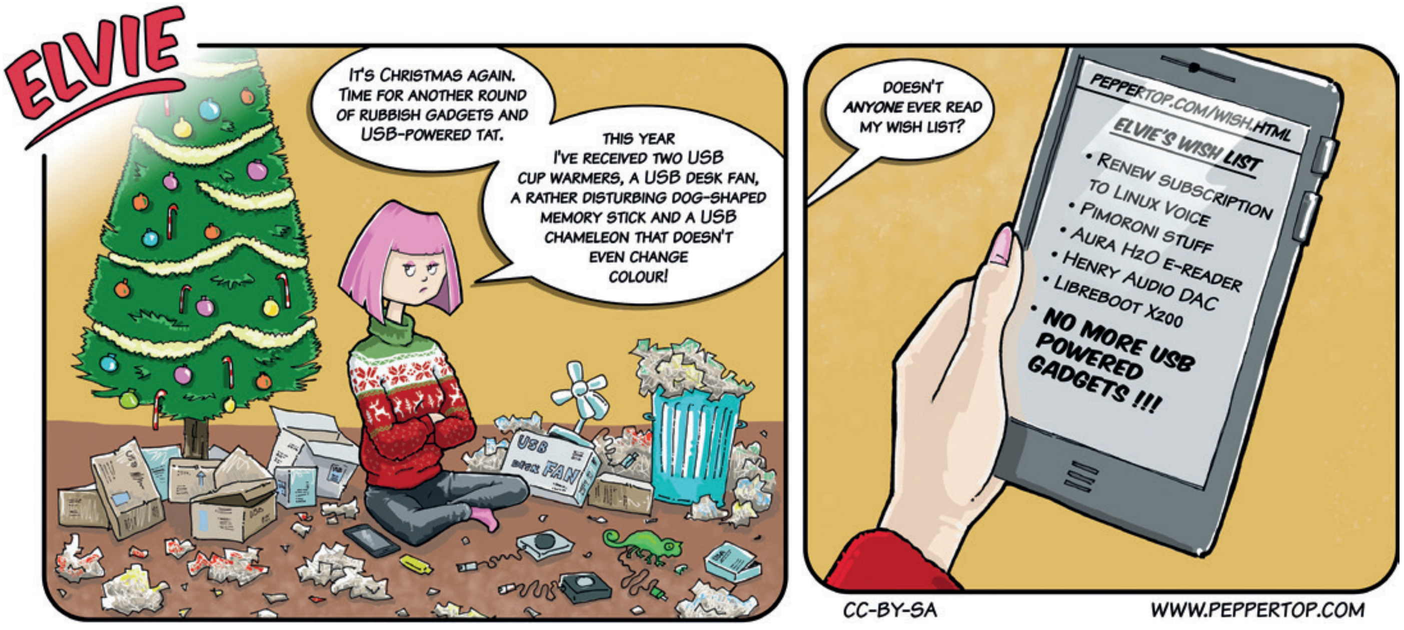
WELCOME, COMRADE

I was a bright-eyed Linux newbie when picking up your magazine from #1. Since then I'm now confident enough to say that "I kinda know what I'm doing on a GNU/Linux box", so thanks. I read the Distrohopper section each month with interest and wonder how I can possibly figure out the best distribution for my application – because there are so many out there!

My stack is Python 2, RabbitMQ, MySQL and SOLR, and I pay a cloud solution for the pleasure of this. This is on a GNU/Linux server and I have grabbed the most accessible distro out there for a newbie (Ubuntu) but have never really evaluated this decision. Is there a more appropriate distribution for me? How would I go about deciding what is the most suitable distribution for my application?

noisyboiler

Andrew says: If you're happy with it, then Ubuntu is the right platform for you. It's not for everyone, but it's used as a server platform by loads of massive companies, so you're in good company. Keep the faith (but you might as well try Mint, and Mageia, and Fedora, and Arch…)


ELVIE
IT'S CHRISTMAS AGAIN. TIME FOR ANOTHER ROUND OF RUBBISH GADGETS AND USB-POWERED TAT.
THIS YEAR I'VE RECEIVED TWO USB CUP WARMERS, A USB DESK FAN, A RATHER DISTURBING DOG-SHAPED MEMORY STICK AND A USB CHAMELEON THAT DOESN'T EVEN CHANGE COLOUR!
USB DESK FAN
DOESN'T ANYONE EVER READ MY WISH LIST?
PEPPERTOP.COM/WISH.HTML
ELVIE'S WISH LIST
• RENEW SUBSCRIPTION TO LINUX VOICE
• PIMORONI STUFF
• AURA H2O E-READER
• HENRY AUDIO DAC
• LIBREBOOT X200
• NO MORE USB POWERED GADGETS !!!
CC-BY-SA
WWW.PEPPERTOP.COM

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

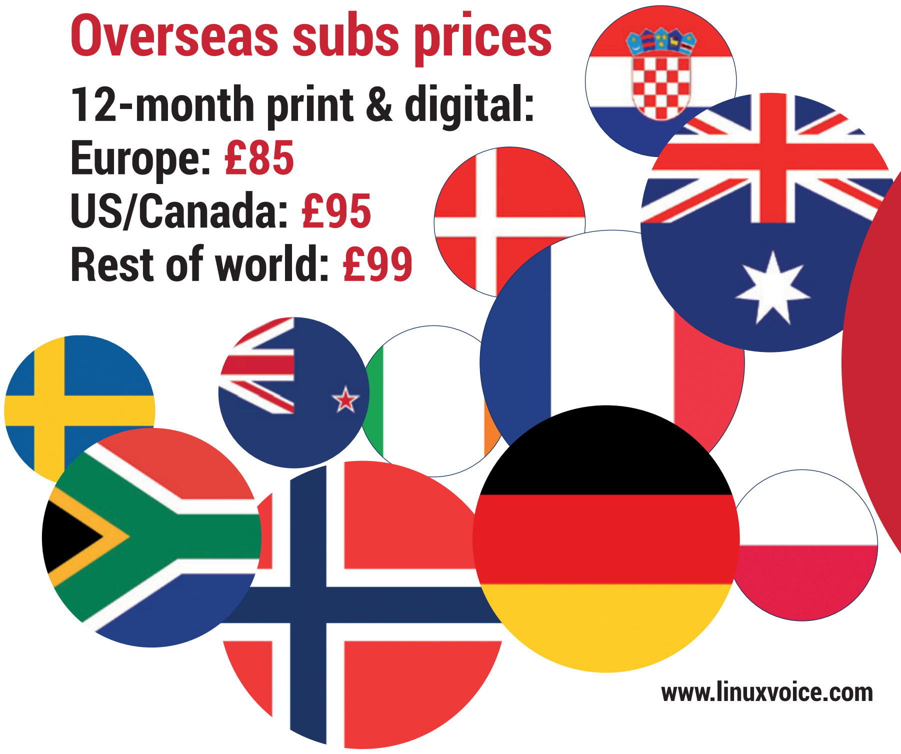
Save money on the shop price and get each issue delivered to your door

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

USE LINUX SMARTER

Try more Linux distros

Get a prettier desktop

Keep tabs on your system

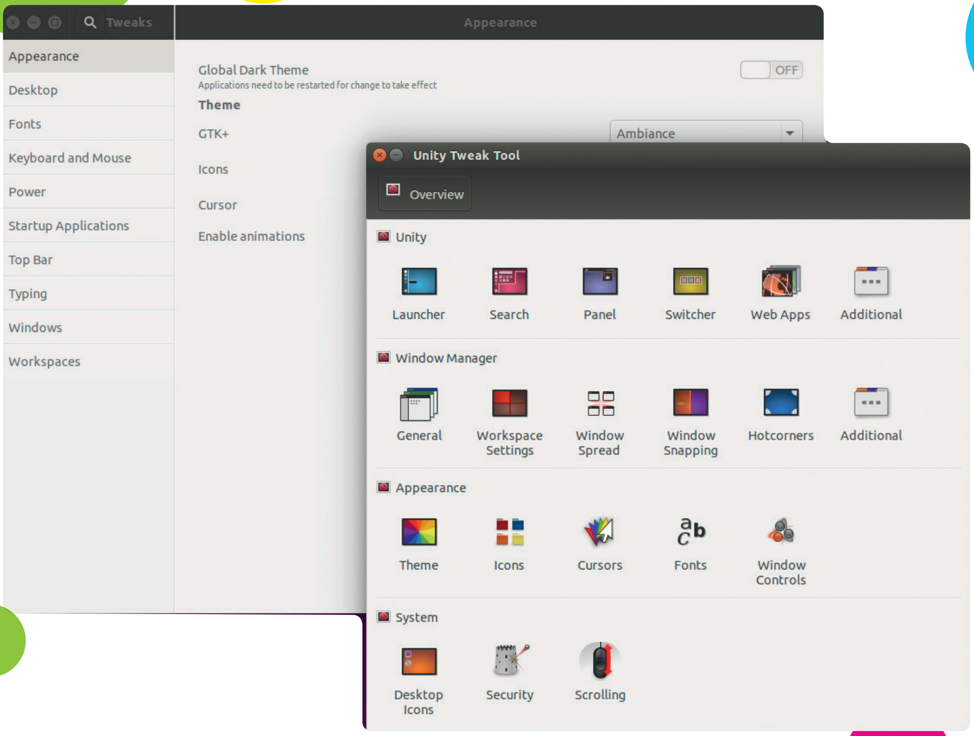
Serve your own website

Keep on top of updates

Streamline the way you work

Whether you're a Linux beginner or a certified expert, we've put together a collection of the finest tips known to Gnumankind.

Learning Linux is a lot like learning how to solve the Rubik's Cube. In the beginning, It can look inaccessible and complicated. Even knowing where to start is a challenge. But after installing your first distribution and moving on to your second, it gets easier. It's the equivalent to the classic beginner's strategy to the Rubik's Cube – solving one side of the cube followed by another. But the solution, as with achieving Linux enlightenment, is to build on layers. When you've nailed your first solution, go back to refine your reflexes, strengthen your finger muscles and commit new algorithms to memory. Which is exactly what we're going to do here – sending nuggets of speed, efficiency, wisdom and knowledge back to our former selves, enabling anyone to level-up their Linux skills.

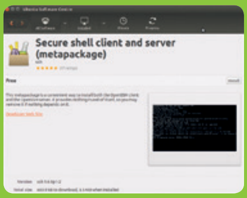


Global Dark Theme: OFF
Applications need to be restarted for change to take effect
Theme
GTK+: Ambiance
Icons
Cursor
Enable animations

Put back missing features in your desktop

Both Ubuntu and Gnome are well known for removing options, but you can get back a lot more control by installing their associated Tweak Tools. For example, Ubuntu's will enable you to disable Amazon searches, switch the window control buttons to the right or adjust the size and transparency of the launcher.

Install/enable SSH



The one essential daemon that needs to be running on any machine, whether it's brewing beer, a Raspberry Pi or a remote server, is **SSH server**. The package will often need to be installed separately, as with Ubuntu.

With the SSH daemon/server enabled, you can fix almost any problem remotely without re-installing, or connecting screens and keyboard, or even fixing PCs when the video or display is messed up.

Use keyboard shortcuts

There's a good reason Ubuntu displays common shortcuts when you first get to the desktop – using them will transform your experience. Learning just a few shortcuts for your desktop, your browser and the command line will make you faster and more efficient. More importantly, you'll look awesome. All desktops enable you to change the defaults, so it's also worth making your shortcuts map to the same keys across all applications.

Our favourite shortcuts

```
[graham@mbp-arch ~]$ cat unicorn.txt
```

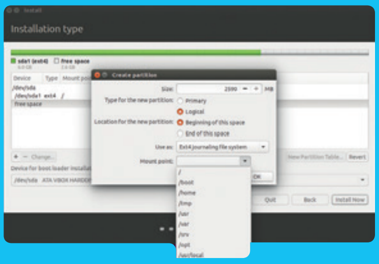
After a quick office poll, here are four of our most commonly used shortcuts:

- **Ctrl+C/X/V** Everyone knows these – cut, copy and paste.
- **Alt+Left Click** With Alt held down, drag any window without clicking on the title bar.
- **Ctrl+wheel** Zoom in/Zoom out. Works almost everywhere, from icon sizes in the file manager to *LibreOffice* and web pages.
- Ctrl+W/Q. Close windows and tabs with W; quit apps with Q.

Give Home a permanent home

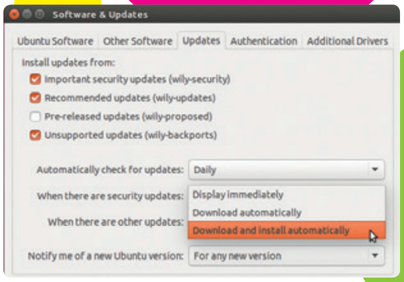
The one thing all of us like to do is install more than one Linux distribution, whether that's by running two instances of Linux at once, or replacing one with another release. To make this as hassle-free as possible, we'd recommend creating a separate **Home** partition when you install your first distribution. **Home** is where all your personal data lives, as well as your various configuration files, caches and libraries, and putting it on its own partition means you can share this data across multiple installed distributions, and keep your data safe when you install a new distribution or upgrade an old one.

The process for creating and selecting a **Home** partition is different for each distribution, but in all of the, you'll need to select a custom partitioning scheme from the installer. In Ubuntu, for instance, select 'Something Else'. Add at least a root partition with a **/** mount point, followed by a **/home** partition/mount point, and a swap partition. We'd recommend making swap the same as your RAM, root between 10–100GB (depending on your usage) and more for home.



Size: 2500 MB
Type for the new partition: Primary [✓ Logical]
Location for the new partition: [✓ Beginning of this space] End of this space
Use as: Ext4 journaling file system
Mount point: / /boot /home /tmp /usr /var /srv /opt /usr/local

Don't forget: disable the 'format' option when selecting an old Home partition.



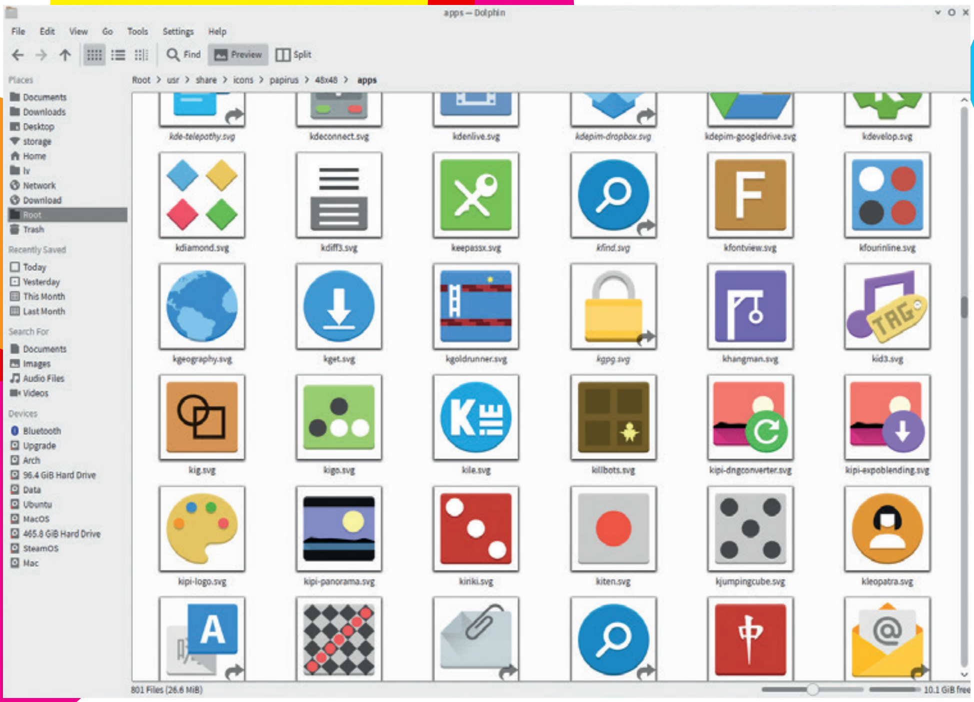
Install updates from:
- ✓ Important security updates (wily-security)
- ✓ Recommended updates (wily-updates)
- ☐ Pre-released updates (wily-proposed)
- ✓ Unsupported updates (wily-backports)

Automatically check for updates: Daily
When there are security updates: Display immediately / Download automatically / [✓ Download and install automatically]
When there are other updates:
Notify me of a new Ubuntu version: For any new version

To enable automatic update in Ubuntu, open the Software & Updates panel.

Update everything automatically

You only stop being vulnerable to a security flaw after you've updated your system. Unless you want to keep abreast of every threat, it's easier to turn on automatic updates. Ubuntu can download and install them automatically, for instance, and unlike Windows 8, you'll always be notified before an upgrade.



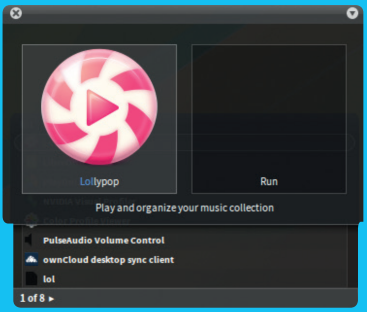
Install a different theme/font/icon/colour scheme

We know that appearances are only skin deep, but giving your desktop a new look is like putting a fresh lick of paint on your shed: it gives you a new perspective, and makes you feel like getting out/in there and making the most of it. In particular, we love the new Google-inspired 'Paper' theme for GTK/Gnome/Unity and 'Papirus' for the KDE desktop.

Gnome Do everything

Gnome Do is one of those little-known utilities you'll wonder how you ever lived without. It's a shortcut to launching applications, searching your desktop and the web, playing music, updating social networks, sending email, and doing almost anything else as long as there's a plugin for it. It does all this from a super simple keyboard shortcut, which is Super+Space by default (the Super key is usually the one with the Windows symbol on it).

Perhaps the reason *Gnome Do* isn't more widely used is that it's not obvious how it works. After launch, *Gnome Do* appears as two large squares. The first will hold the result of what you start searching for, while the second holds the action. Search for a file, for example, and the action will default to 'open'. You can page through the list of results by cursoring down, and you can change actions by tabbing across to the other square and using the cursor keys again. It's quick, powerful and can replace your launcher and speed up the way you work.


Lollypop
Run
Play and organize your music collection
PulseAudio Volume Control
ownCloud desktop sync client
lol
1 of 8

Gnome Do works brilliantly with the *Conky* desktop application launcher.

Matthew Garrett

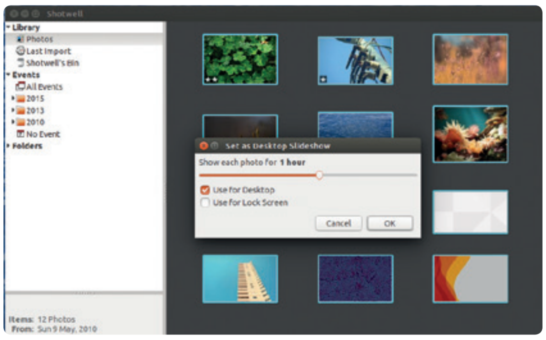
Freedom advocate and security engineer.

Forgot to type **sudo** before a command? **sudo !!** will re-run the previous command under **sudo**. But not just **sudo**: it can be used to precede the previous command with whatever comes before the **!!**.

7 Incredible KRunner Shortcuts

Like *Gnome Do*, *KRunner* is a command-driven launcher, opened from KDE by pressing Alt+F2. Here are some of our favourite features:

- **Web shortcuts** Type **wp:** to search Wikipedia and open the results in your default web browser.
- **Calculator** Precede a calculation with = to see the answer, eg **=3*sin(90)**.
- **Pervasive search** Type the beginning of an application or file to open it.
- **File manager** Typing **file:Downloads** opens a file manager for the Downloads folder.
- **Messaging** Type the name of a contact to initiate an IM conversation.
- **Amarok remote** Control playback with Play, Pause and Next.
- **Desktop control** Type **desktop 1** to switch desktops, or try **logout** and **shutdown**.



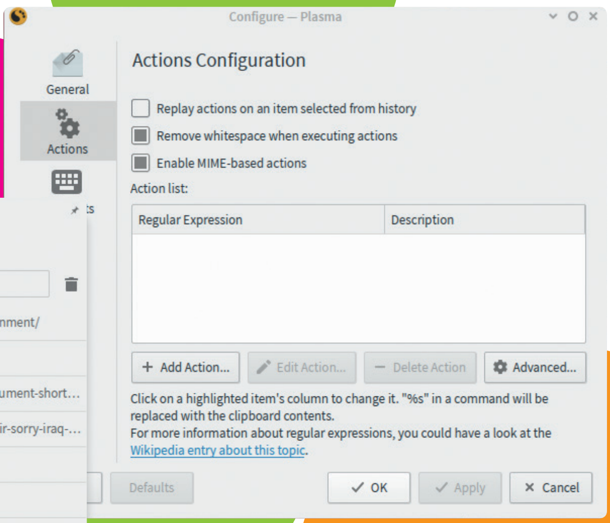
Dynamic backgrounds

In Unity, select images in Shotwell and select 'Desktop Slideshow' from the File menu. The desktop will cycle through the slideshow, and the panel and window colours will also change.

Use a clipboard manager

Copy and paste is fundamental to the way we use computers. Linux is already ahead of the game in the way you can select text with your mouse and paste it with a simple click of the middle mouse button, but you can do so much more.

Install a clipboard manager like *Glipper* or KDE's *Clipboard* and you can access any of your previous cuttings, reselect them, and paste as usual. But you can also do clever things like perform an action when you copy something specific, or use a regular expression to modify the data for pasting.


Configure — Plasma

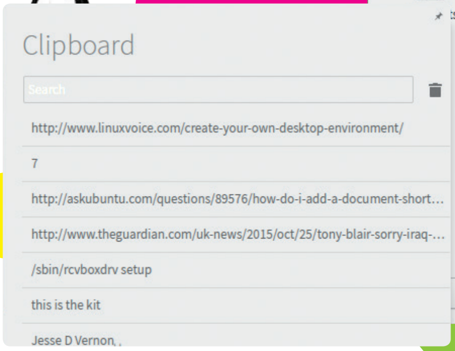
Actions Configuration

- [ ] Replay actions on an item selected from history
- [ ] Remove whitespace when executing actions
- [ ] Enable MIME-based actions

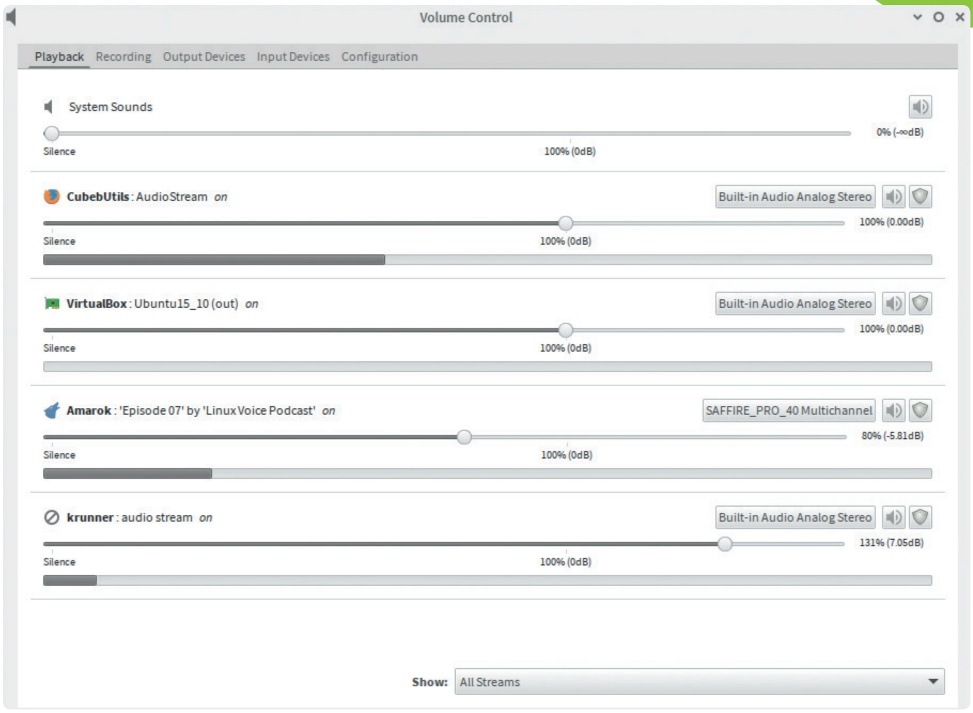
Action list:

Regular Expression	Description

Click on a highlighted item's column to change it. "%s" in a command will be replaced with the clipboard contents.
For more information about regular expressions, you could have a look at the Wikipedia entry about this topic.


Clipboard

```
http://www.linuxvoice.com/create-your-own-desktop-environment/
7
http://askubuntu.com/questions/89576/how-do-i-add-a-document-short…
http://www.theguardian.com/uk-news/2015/oct/25/tony-blair-sorry-iraq-…
/sbin/rcvboxdrv setup
this is the kit
Jesse D Vernon, ,
```


Volume Control

```
System Sounds                                                        0% (-∞dB)
Silence                              100% (0dB)

CubebUtils : AudioStream  on                   Built-in Audio Analog Stereo
                                                                 100% (0.00dB)
Silence                              100% (0dB)

VirtualBox : Ubuntu15_10 (out)  on             Built-in Audio Analog Stereo
                                                                 100% (0.00dB)
Silence                              100% (0dB)

Amarok : 'Episode 07' by 'Linux Voice Podcast'  on  SAFFIRE_PRO_40 Multichannel
                                                                  80% (-5.81dB)
Silence                              100% (0dB)

krunner : audio stream  on                     Built-in Audio Analog Stereo
                                                                 131% (7.05dB)
Silence                              100% (0dB)

Show: All Streams
```

Vimerise Firefox shortcuts

There are many addons for *Firefox*, but the first we install is *VimFx*. This will default all *Firefox* keyboard shortcuts to keys familiar to any *Vim* user, such as O for opening, X to close, GG/Shift+G for top/bottom and '/' for searching. Press F and every link is overlayed with a new shortcut for easy navigation – mouse free!

Manage your audio levels with Pavucontrol

Nearly every Linux distribution now defaults to PulseAudio for audio duties, but they provide little control over how volumes and devices are configured. If you need to see what's going on, and give yourself more control, install the sparse but powerful Pavucontrol tool. It lists every application generating sound and enables you to change the audio device or output used by each one, as well as visualising the levels and giving you control over the volume. You can also set application-specific default devices.

Ben Nuttall

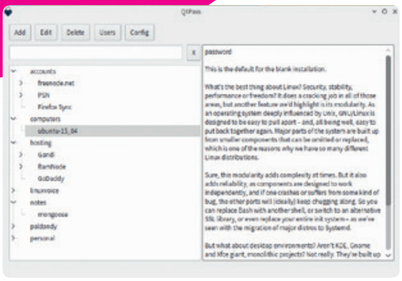
Education advocate at Raspberry Pi.

On the command line, Alt + . (Alt and full stop) places the last argument of the previous command into your current position. For example, if you've just entered **mkdir bob**, type **cd** and hit Alt + . and your command will now be **cd bob**. It's got history too, so keep hitting "." to get the one before.

Pro Tip: Graham Morrison

Magazine editor and amateur musician

If you've ever been in the dire situation of losing your partition table, perhaps after a wayward **dd** command, the **testdisk** utility is the best way I've found to recover your data. Run it from a live CD/USB drive and choose the Analyse option. With a bit of luck it will find and restore your precious data.



password

This is the default for the blank installation.

What's the best thing about Linux? Security, stability, performance or freedom? It does a cracking job in all of those areas, but another feature we'd highlight is its modularity. As an operating system deeply influenced by Unix, GNU/Linux is designed to be easy to pull apart – and, all being well, easy to put back together again. Major parts of the system are built up from smaller components that can be omitted or replaced, which is one of the reasons why we have so many different Linux distributions.

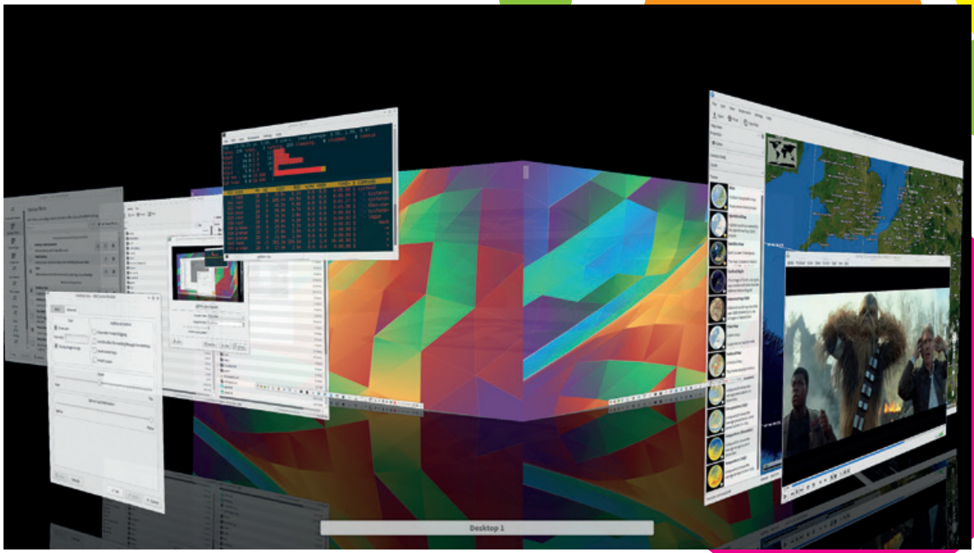
Sure, this modularity adds complexity at times. But it also adds reliability, as components are designed to work independently, and if one crashes or suffers from some kind of bug, the other parts will (ideally) keep chugging along. So you can replace Bash with another shell, or switch to an alternative SSL library, or even replace your entire init system – as we've seen with the migration of major distros to Systemd.

But what about desktop environments? Aren't KDE, Gnome and Xfce giant, monolithic projects? Not really. They're built up

Get a password manager

Passwords have never been more important, which is why they should all be random and unique. But that obviously makes them impossible to remember. This is what a password manager is for – it's a single repository locked by a strong memorable password (or two-factor authentication), which then gives you access to your other passwords.

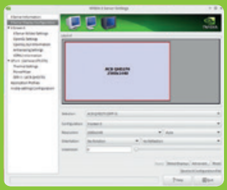
All the main desktops integrate their own, but your passwords are non-transferable, not available on other computers, and not on your phone. The solution is to use a portable password manager or a remote password store. *KeePass* is our favourite, because you can keep it on a USB drive and there are open source Linux and Android clients. You only need a copy of the database to access your passwords. But we also really like the **pass** tool, as it simply uses GnuPG encryption and the Linux filesystem to do a similar job.



Use your desktop's best feature: Virtual Desktops

Every operating system seems to have virtual desktops now, but that's because they're awesome, and Linux still has the most powerful implementations. You can, for example, configure your setup so that emails always launch on desktop 3, or use a tiling window manager to organise each desktop by task – say, accounts in one desktop, personal in another desktop, and real work in another. Save yourself from a having to buy a second screen and work from anywhere.

Dual and triple displays



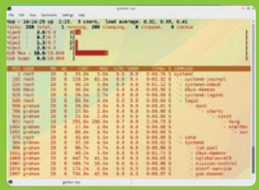
Linux now mostly works with second and third displays thanks to the X extension for multiple displays configuring itself correctly. However, we've found nothing to beat the flexibility and performance of Nvidia's proprietary drivers and their 'TwinView' implementation. Use the **nvidia-settings** tool that comes with these drivers to enable TwinView and edit the settings for both connected monitors and the dual display without restarting the desktop session.

Learn Vim, finally

Linux users do a lot of text editing, and while GUI editors are great, nothing beats *Vim* for being able to edit files direct from the command line, whether that's on a server, your desktop or a Raspberry Pi.

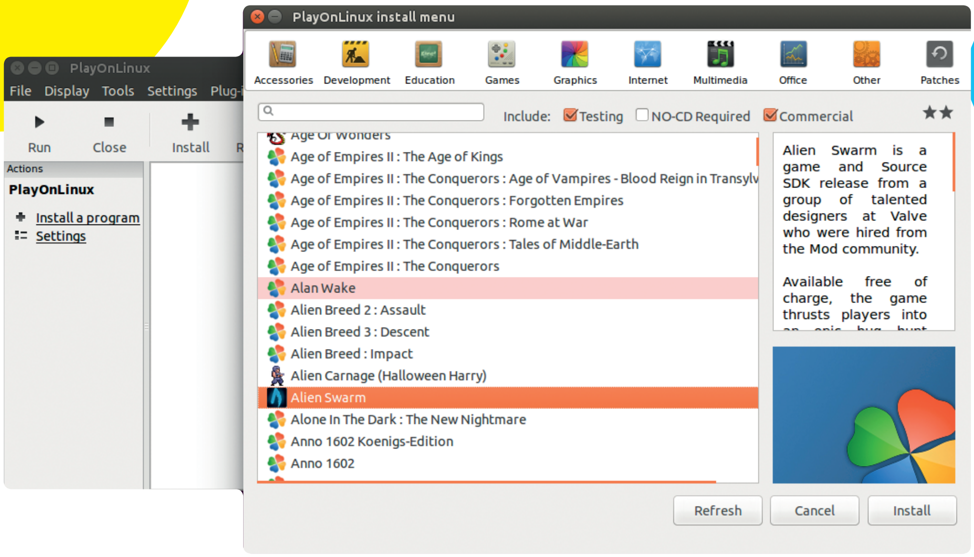
Vim is one of the most powerful editors ever created, but it uses lots of keyboard shortcuts. To make learning easier and fun, we highly recommend a website called Vim Adventures. It makes you play a game to learn *Vim*'s various intricacies!

Essential Bash shortcuts



Here our three essential tips to getting the most out of the command line:

- **Ctrl+R. <command>.** Search command history and auto-expand the same command.
- **Ctrl+A, E** Respectively, these go to the beginning and end of a command. Meta B and F will also move between words.
- **Ctrl+U, P** The first will cut from text before the cursor (great for mistyped passwords); the latter will paste this buffer before the cursor.



Age Of Wonders
Age of Empires II : The Age of Kings
Age of Empires II : The Conquerors : Age of Vampires - Blood Reign in Transylv
Age of Empires II : The Conquerors : Forgotten Empires
Age of Empires II : The Conquerors : Rome at War
Age of Empires II : The Conquerors : Tales of Middle-Earth
Age of Empires II : The Conquerors
Alan Wake
Alien Breed 2 : Assault
Alien Breed 3 : Descent
Alien Breed : Impact
Alien Carnage (Halloween Harry)
Alien Swarm
Alone In The Dark : The New Nightmare
Anno 1602 Koenigs-Edition
Anno 1602

Alien Swarm is a game and Source SDK release from a group of talented designers at Valve who were hired from the Mod community.

Available free of charge, the game thrusts players into

Run Windows software on Linux

There's a great range of software available on Linux, but every once in a while we find ourselves needing something that will only run on Windows. For this scenario there's *Wine*, a compatibility layer that enables Windows executable files to run on Linux, but it can be difficult to set up. Fortunately, there's also *Play on Linux*. This is a wrapper for *Wine* that has pre-set configuration files for loads of common programs that make them really easy to install and run. It was originally designed for games (hence the name), but now it includes a range of programs, including office software and development tools. Thanks to an intuitive interface, getting this software on Linux takes just a few clicks.

Relax your mind

Humble Bundles are pay-what-you-want collections of DRM-free indie games for Linux, Windows and Mac OS X. They're a great value source of entertainment (www.humblebundle.com).

Grab extra software

Most distros come with plenty of software in their repositories, but if you find yourself needing something that's not already available (or a newer version of a piece of software), most distros have alternative community-maintained repositories.

Arch has the Arch User Repository (AUR), which has a frankly mindblowing array of software. Red Hat and Centos have the Extra Packages for Enterprise Linux (EPEL) repositories, which contain mostly server software that isn't supported by Red Hat. Ubuntu has Personal Package Archives (PPAs) hosted on Launchpad, which are basically mini repositories that you can add for each piece of software.

Save time with aliases

You may find that there are certain commands you end up running very frequently. You can use aliases to save shorted versions of these commands to make them easier to access. The format for this is:

```
alias <newcmd>="<cmd to run>"
```

So, if you constantly find yourself wanting to view the full details of all the files in a directory (**ls -la**), you can use:

```
alias la="ls -la"
```

Now, whenever you type **la**, *Bash* will run **ls -la**. If you want this to stay every time you restart *Bash*, you need to add the alias command to the end of your **~/.bashrc** file.

Pro Tip: Alan Pope

Ace community manager at Canonical

Often I've wanted to look at the source code for something installed on my Ubuntu system, but don't want to have to go looking for it online in Launchpad, Bitbucket, GitHub or Sourceforge. With Debian-based systems you can get the source package that was used to build the binary of whatever is on your system, including patches, like this: **apt-get source firefox**. Reading and understanding the source is of course another matter...

REISUB

Shhh, don't tell the Windows and Mac users, but sometime Linux crashes, and sometimes it crashes hard. Not very often, but once in a while the screen will lock up, and nothing you can do with keyboard or mouse seems to do anything.

The last-ditch option is the magic SysRq key combination. If you hold down Alt and Print Screen (also known as SysRq) then press R,E,I,S,U then B, your system will restart a little safer than just powering down (which can corrupt data). These key presses correspond to (in order):

- Switch keyboard to raw mode.
- Ask all processes except init to finish.
- Kill all unfinished processes except init.
- Sync all mounted filesystems.
- Remount all filesystems as read-only.
- Reboot.

If you're having trouble remembering this, some people find the mnemonic 'Reboot Even If System Utterly Broken'.

If you need a more whimsical way to remember the REISUB sequence, try: 'Raising Elephants Is So Utterly Boring".

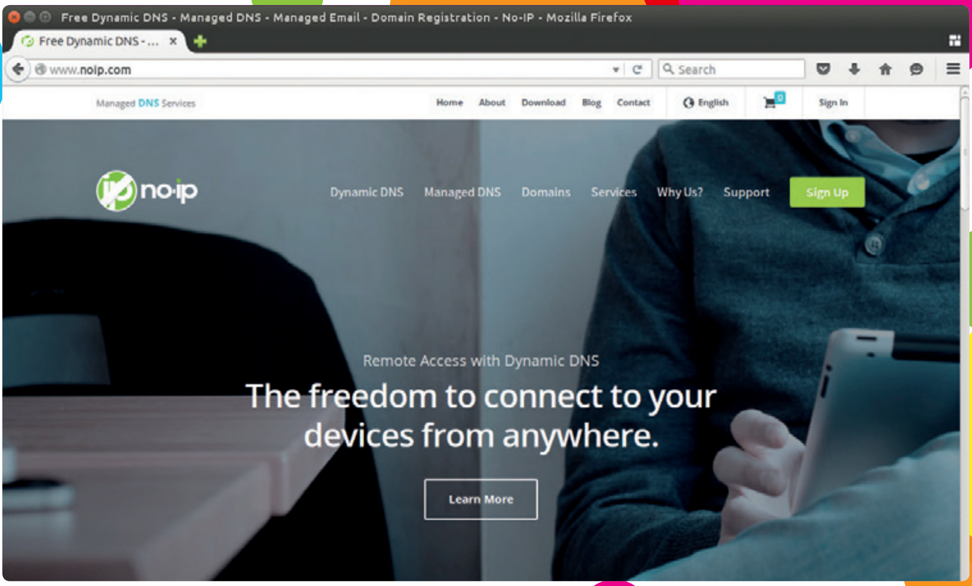
Pro Tip: Mike Saunders

Creator of the famous MikeOS

You can turn any machine with SSH access into an instant web proxy server with this command:

ssh -N -D 0.0.0.0:8888 user@hostname

In your browser, go to connection settings (eg Advanced > Network > Settings in *Firefox*) and use 127.0.0.1 (your local IP) as the SOCKS host, and 8888 for the port. Browsing will now be via the SSH machine.


Free Dynamic DNS - Managed DNS - Managed Email - Domain Registration - No-IP - Mozilla Firefox
www.noip.com
Managed DNS Services
Home About Download Blog Contact English Sign In
no-ip
Dynamic DNS Managed DNS Domains Services Why Us? Support Sign Up
Remote Access with Dynamic DNS
The freedom to connect to your devices from anywhere.
Learn More

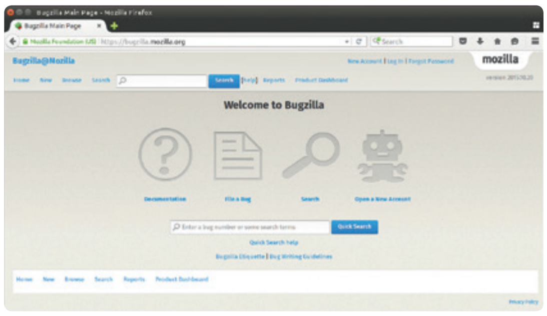
Join the web

If you've got a spare Linux machine, you can use it to host a website. Most distros include a web server (such as Apache) that can easily host pages, and with a dynamic DNS system you can get a domain name to point to your home internet connection. Using these, you can become your own web master and share your passion with the world, make your fortune by starting the next Google, or make some of your data available when you're away from home.

Monitor your system

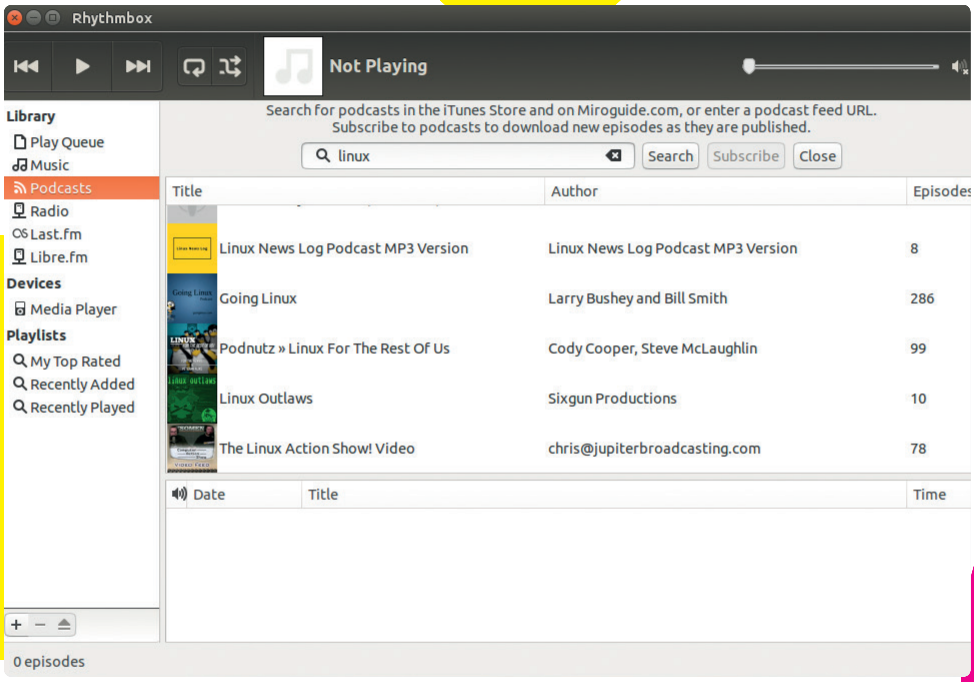
The **top** command is well known for providing a real-time overview of which software is using the CPU and memory. As well as providing a broad overview, it gives detailed statistics for each process. This style has inspired other commands to monitor performance . Our four favourite are:

- **iftop** displays detailed information about how much data is going through your network port.
- **ptop** and **mtop** help you monitor your *PostgreSQL* and *MySQL* databases.
- **virt-top** supplies all the latest information on how your virtual machines are performing.
- **apachetop** provides detailed information about how your *Apache* web server is performing.
- **iotop** keeps an eye on your disk performance.



Report bugs

When you find a bug in open source software, don't ignore it. Report it to the developers and help them make the software even better.



Rhythmbox

Not Playing

Search for podcasts in the iTunes Store and on Miroguide.com, or enter a podcast feed URL.
Subscribe to podcasts to download new episodes as they are published.

linux | Search | Subscribe | Close

Title	Author	Episodes
Linux News Log Podcast MP3 Version	Linux News Log Podcast MP3 Version	8
Going Linux	Larry Bushey and Bill Smith	286
Podnutz » Linux For The Rest Of Us	Cody Cooper, Steve McLaughlin	99
Linux Outlaws	Sixgun Productions	10
The Linux Action Show! Video	chris@jupiterbroadcasting.com	78

0 episodes

Listen to podcasts

Podcasts are a great way of learning more about Linux, providing as they do the facility to pipe information and entertainment directly into your ears as you commute, or do your weekly shop. There are loads of options, from the confrontational Bad Voltage, to the melodic Ubuntu UK Podcast, to the joyfully stuck-in-the-mud Linux Luddites – and don't forget the fortnightly Linux Voice podcast, which the team somehow find time to record when they're not making this magazine.

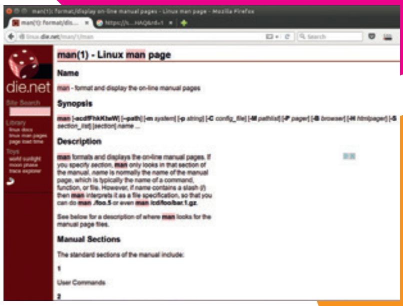
Virtualise new distros

There are hundreds of different distros out there, so how do you know which one is right for you? The best option is to try a few out, this can be time consuming.

Fortunately there's a quicker way: virtualisation. Using a tool like *VirtualBox* or *KVM* you can create virtual machines that enable you to boot a distro from within your currently running machine. The downside of this is that the system you're testing won't be as snappy as it would if it were running natively, but sidestepping the need to push the file onto a USB stick and reboot makes it much faster to try new distros.

Pro Tip: Nick Veitch
Publishing legend

Virtual machines are your friends! The overheads to using them are practically non-existent these days and mean you can easily separate your home desktop with all the things which you critically need to work from dangerous nonsense – strange network services, experimental kernel options, different distros, silly software which requires at least half a dozen unstable versions of libraries to run and anything that Mike had a hand in.



man(1): format/display on-line manual pages - Linux man page - Mozilla Firefox

linux.die.net/man/1/man

man(1) - Linux man page

Name

man - format and display the on-line manual pages

Synopsis

man [-acdfFhkKtwW] [--path] [-m *system*] [-p *string*] [-C *config_file*] [-M *pathlist*] [-P *pager*] [-B *browser*] [-H *htmlpager*] [-S *section_list*] [section] *name* ...

Description

man formats and displays the on-line manual pages. If you specify *section*, **man** only looks in that section of the manual. *name* is normally the name of the manual page, which is typically the name of a command, function, or file. However, if *name* contains a slash (/) then **man** interprets it as a file specification, so that you can do **man ./foo.5** or even **man /cd/foo/bar.1.gz**.

See below for a description of where **man** looks for the manual page files.

Manual Sections

The standard sections of the manual include:

1

User Commands

2

Find the missing manual

Almost every piece of command line software comes with a manual that's full of information (know as the man page). You can get the man pages with the command:

```
man <command>
```

There's a particular style and format for man pages, which can take a little getting used to, so it's a good idea to become familiar with the manual before you need it. One of the best features of the manual is that it doesn't rely on the network, so you can always get the help you need – great for then you're stuck with a knotty problem and can't get access to the internet.

How do you learn how to use **man**? Why with the man page of course! Get started in your journey to manual mastery with:

```
man man
```

Edit images with commands

If you have a lot of photographs, making edits to them can be time consuming: resizing them to save space, correcting for a problem with the camera or rotating them. It all takes time. Rather than go through each photo individually in graphical editing software such as *Gimp*, you can so everything from the command line using the *ImageMagick* tools. The most useful of these is the **convert** program, which takes a file, edits it and creates a new image.

There are a huge array of options (far too many for us to cover here), but a simple example of creating a numbered thumbnail from every PNG file in a directory is:

```
convert '*.png' -resize 120x120
thumbnail%03d.png
```

Dmesg reporting

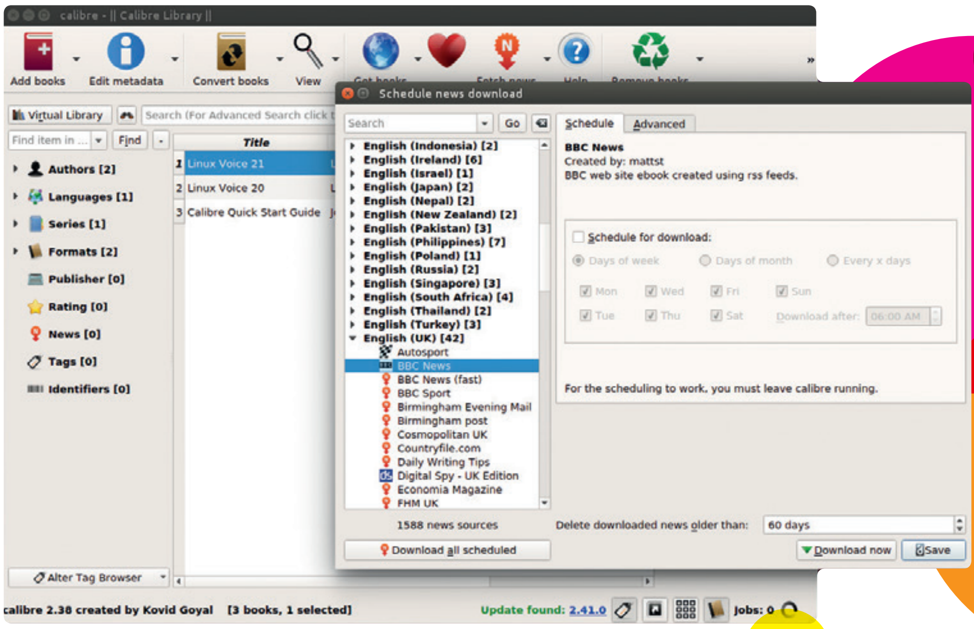
If you're having trouble with hardware, the most useful source of information is the kernel message buffer, which is displayed with the command **dmesg** (you can pipe it into **less** to enable you to scroll through, or **grep** to search for a particular word). The kernel message buffer includes all the messages from kernel drivers that are controlling the hardware, so if there's anything awry in this area, you should see evidence of it in the output from **dmesg**.

Dmesg is also useful for debugging the boot sequence. All those messages that appear on the screen as your distro starts (when you press Esc to hide the splash screen) are safely stored here, so you can find out what happened when you started your machine.

Use a BSD

This might sound like an odd tip to help you become a better Linux user, but it can be a useful point of comparison. If you've only used Windows, OS X and Linux, then you've missed a whole genre of free software OSes to help you understand computing. BSDs, as the other major open source Unix-alike, provide a really useful counterfoil to Linux. By using both Linux and a BSD, you can get a better feel for the decisions Linux distros make, and decide for yourself if you think they're good or bad.

PC-BSD and GhostBSD are great options for your first try, as they're both built with desktop users in mind. FreeBSD is also worth considering if you're planning on using your BSD machine as a server.



1 Linux Voice 21
2 Linux Voice 20
3 Calibre Quick Start Guide

English (Indonesia) [2]
English (Ireland) [6]
English (Israel) [1]
English (Japan) [2]
English (Nepal) [2]
English (New Zealand) [2]
English (Pakistan) [3]
English (Philippines) [7]
English (Poland) [1]
English (Russia) [2]
English (Singapore) [3]
English (South Africa) [4]
English (Thailand) [2]
English (Turkey) [3]
English (UK) [42]
Autosport
BBC News
BBC News (fast)
BBC Sport
Birmingham Evening Mail
Birmingham post
Cosmopolitan UK
Countryfile.com
Daily Writing Tips
Digital Spy - UK Edition
Economia Magazine
FHM UK

1588 news sources

BBC News
Created by: mattst
BBC web site ebook created using rss feeds.

For the scheduling to work, you must leave calibre running.

Delete downloaded news older than: 60 days

calibre 2.38 created by Kovid Goyal [3 books, 1 selected] Update found: 2.41.0

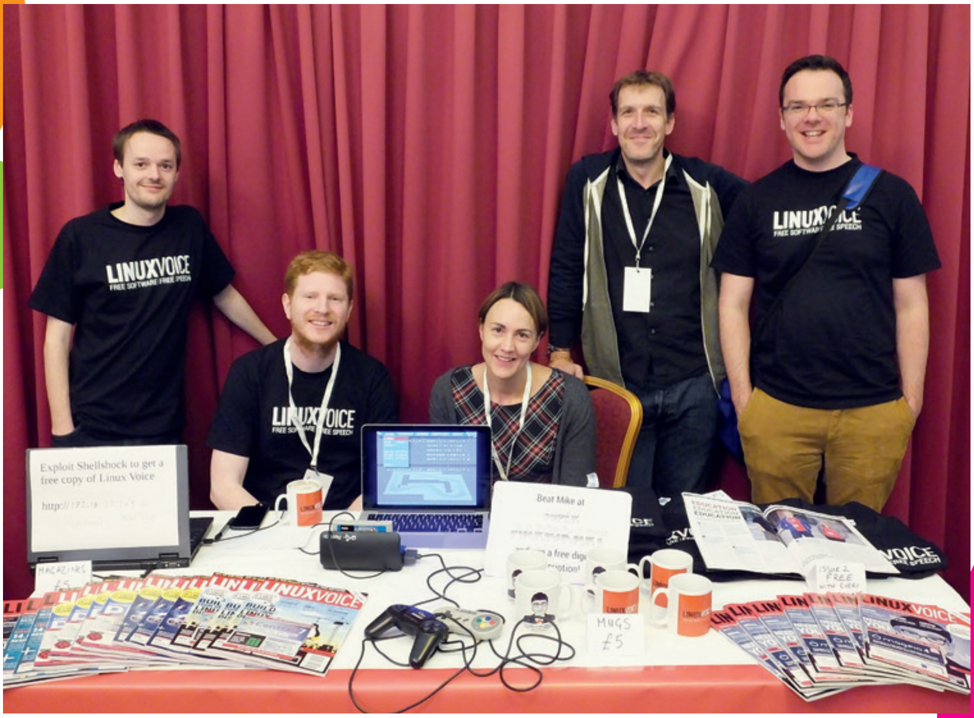
Pro Tip: Ben Everard

Author of the best Raspberry Pi book.

When picking your Linux setup, remember that cutting-edge distros force you to update a lot, while stable distros can have older software. Decide what balance between the two aspects is right for your needs and find a distro that makes the same compromise. Despite what some zealots might say, there is no perfect choice.

Calibre newsfeeds

You can create ebooks automatically from web pages or RSS feeds. Just point the Calibre application at the sites you're interested in and it will grab content for you to enjoy offline on your eReader or smartphone.



Meet fellow Linux users

We don't know of a better way to experience the Free Software community than by meeting fellow geeks. There are Linux User Groups (LUGs), events and meetups around the world where people come together to chat Linux and maybe have a beer or two – a couple of our favourites are OGGCamp in the UK and FOSDEM in Belgium.

Recover deleted files

We've all had that horrible sensation of a slipped finger on the mouse, or an accidental command, and all of a sudden you've deleted an important file. All is not lost, however, and with a little luck you may be able to get the deleted files back.

First, make sure you don't write anything else to disk. Shut down the machine and reboot with a live distro if possible, then try either **extundelete** or **photorec** to recover the lost files.

The clipboard in the CLI

Copy and paste are two invaluable commands in graphical applications, and they can be useful at the command line as well. The command **xsel** can be used to pipe data in and out of the clipboard. As a quick example, you can copy a list of the files in the current directory with:

```
ls | xsel --clipboard --input
```

You can Ctrl+V the list into a graphical application, or send data the other way by using the **--output** flag.

Commandlinefu

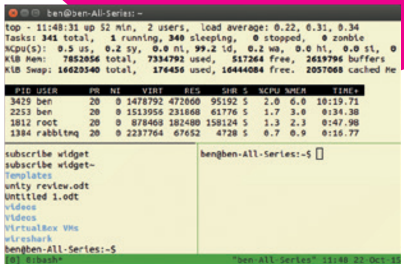
Want more command line tips? The website **www.commandlinefu.com** has a list of user-submitted tricks for your perusal and enjoyment.

There are lots of useful tricks, as well as some frankly odd options. Ever wanted to watch *Star Wars* in the terminal? Or how about querying Wikipedia via DNS? If you have, then you're in luck. Whether you're a terminal newbie or a greybeard, you'll find some new and interesting tricks to improve your skills.

Pro Tip: Matthias Kirschner

President, FSF Europe.

I write as much text as possible in my preferred editor. I use the same editor – in my case "the editor of the beast" (*Vim*) – for emails, personal notes, press releases, FSFE briefings/statements, blog entries, etc; There was a time when this was impossible to do with with text fields in the web browser, like editing wiki pages, writing online comments, or some blog systems. The *Firefox*/*Iceweasel* extension *ItsAllText* solved this problem: it adds a button on text fields in your browser. When you click on it, it opens your preferred text editor, and you can edit the text with your beloved shortcutsand no need to copy and paste.



```
ben@ben-All-Series: ~
top - 11:48:31 up 52 min,  2 users,  load average: 0.22, 0.31, 0.34
Tasks: 341 total,   1 running, 340 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.5 us,  0.2 sy,  0.0 ni, 99.2 id,  0.2 wa,  0.0 hi,  0.0 si,  0
KiB Mem:   7852056 total,  7334792 used,   517264 free,  2619796 buffers
KiB Swap: 16620540 total,   176456 used, 16444084 free.  2057068 cached Me

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+
 3429 ben       20   0 1478792 472060  95192 S   2.0  6.0  10:19.71
 2253 ben       20   0 1513956 231868  61776 S   1.7  3.0   0:34.38
 1812 root      20   0  878468 182480 158124 S   1.3  2.3   0:47.98
 1384 rabbitmq  20   0 2237764  67652   4728 S   0.7  0.9   0:16.77

subscribe widget                  ben@ben-All-Series:~$
subscribe widget~
Templates
unity review.odt
Untitled 1.odt
videos
Videos
VirtualBox VMs
wireshark
ben@ben-All-Series:~$
[0] 0:bash*                        "ben-All-Series" 11:48 22-Oct-15
```

Remote access

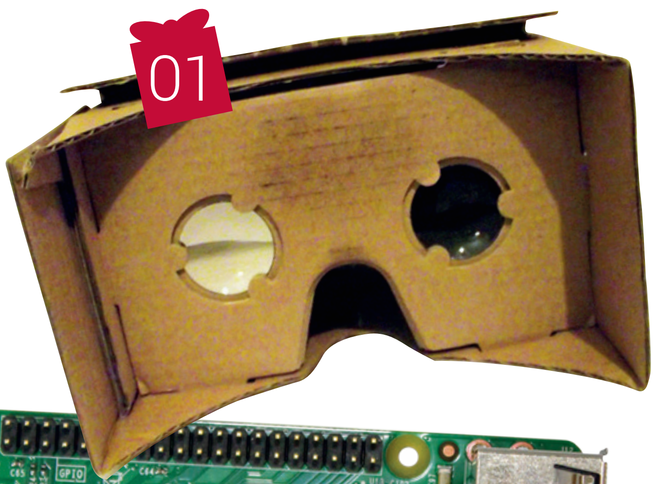
As Linux users, we spend quite a lot of our time remotely accessing other Linux machines. Here are our top five tips for remote access.

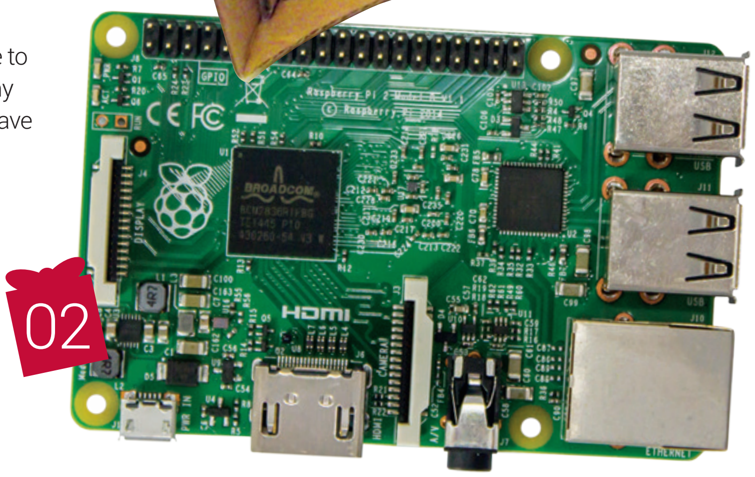
- Transfer files using **rsync** with the **--partial** flag and you can resume failed transfers.
- The MObile SHell (*Mosh*) is a wrapper for SSH that's designed for unstable internet connections such as via a cell phone.
- The terminal multiplexer (*Tmux*) has many tricks up its sleeve for maintaining terminal sessions even when you end a connection and re-establish it.
- Passwords are insecure, and it's far safer to use certificate-based SSH logins.

GEEK GADGET GIFT GUIDE

You can never have enough gadgets. Nor can your friends – and we're here to help you all decide what to get for each other.

There are certain times of the year (maybe one is close?), when a convenient list of lovely things for any Linux or open source geek comes in handy. And as we've looked at rather a lot of this stuff over the last 22 issues, now is the perfect excuse to revisit and revise some of those items, as well as look at a few new things, for anyone looking for inspiration. If you're not looking for inspiration, but would like to inspire someone else, why not circle a couple and leave these two pages open somewhere prominent?





01 Google Cardboard (from £4)

If you've already got an Android phone, this low-cost virtual reality headset is an absolute blast. It relies on your phone for everything – from the gyroscopes, which are used for head tracking, to the screen, which is split into two and focused onto your retinas through a couple of cheap plastic lenses.

For something so hackneyed, the experience is fantastic. Just install any 'Cardboard' apps via Google Play and you'll find yourself fully immersed in the Tuscan countryside, standing on mountaintops, shooting along a roller coaster or exploring an Egyptian tomb. Google's original origami cardboard hurts your nose and doesn't hold the phone adequately, so we'd recommending spending a few meagre pounds more on a proper comfortable plastic enclosure.

02 Raspberry Pi 2 Model B (£30)

We're sure you've already heard about this serious upgrade to the all-conquering Raspberry Pi, but we're even more enamoured by its capabilities after a few months of using it in our own projects. In particular, it's a brilliant media player, especially with *XBMC/Kodi*. For lots of content, you still need the hardware MPEG-2 decoding unlocked (this costs a couple of pounds via **raspberrypi.org**), but this revision can easily stream HD material without hitting the CPU, making it much cooler and more stable. As always, you'll also need a quality power supply, USB hub and probably a case. And for a chance to win one, turn the page.
https://www.raspberrypi.org

03 Ubuntu Phone (from £125)

Despite Canonical's touch-based portable operating system not quite hitting the mark, and facing an uphill adoption struggle, we can't help but admire the company's young pretender to the smartphone throne. The OS itself is doing great things, with the long-awaited convergence mode making an appearance, and it's more open than Android. We'd suggest an Ubuntu Phone is ideal of you want an open source device for tinkering, and we'd recommend a more powerful device for this reason.
www.ubuntu.com/phone

04 BitScope (from £94)

Oscilloscopes are very cool. They visualise the intricacies of varying voltages passing through a circuit, turning what's theoretical into something you can see. They're also incredibly useful for synthesizer technicians and musicians, as those audio voltages will also reveal the constituent waveforms within an audio path. BitScope's headless designs feature Linux support, using your desktop or even your Raspberry Pi as the screen. They're also excellent logic analysers – visualising the binary signals sent from integrated circuits. For oscilloscopes with this kind of power, they're excellent value for money.
www.bitscope.com

05 Kobo Aura H20 (£140)

We love electronic readers. We read dozens of books via their silky backlit e-ink screens every year and we put a lot of love into our own ePub editions specifically for that purpose. And while it's a shame we know of no open source reader, the Kobo Aura H20 supports all DRM-free formats, works with Linux and has a screen to rival the latest Kindle – plus, you can read it in the bath. Dare we also mention it's the perfect accompaniment to a new Linux Voice subscription?
http://kobo.com

06 Steam Controller (£40)

We've not had a chance to play with one of Valve's official boxes yet, although we've been running our own Steam Box since the launch of SteamOS. We still can't quite believe it lets us play AAA games, natively, on Linux, and the situation is only going to get rosier now you can buy officially endorsed Steam PCs. A central part of this strategy is Valve's new controller, which uses two circular touchpads to emulate the responsiveness of mouse control. This works brilliantly with Valve's own games but takes some getting used to for others. Either way, it's the cheapest way to join the Linux games revolution.
http://store.steampowered.com/hardware

07 LibreBoot X200 (from £290)

This is the most expensive item we're looking at here and, as such, won't be an impulse purchase. But if you care about Free Software, it's still great value.

The X200 is a reconditioned Thinkpad overhauled with a software stack endorsed by the Free Software Foundation. This includes LibreBoot as a BIOS/UEFI firmware replacement, and Trisquel GNU/Linux as the operating system. Technically, the machine is more than adequate with perhaps just the screen falling below modern HiDPI standards (it's a 12.1" 1280×800 TFT LCD display). Even more impressively, profits from sales will fund the LibreBoot project.
http://minifree.org

08 Henry Audio DAC (approx. £160)

If you love music, and listen predominately from CDs or Flacs, this high-quality digital-audio-converter (DAC) is a great upgrade over your computer's inbuilt audio output. It plugs into a spare USB port, requires no external power and no special drivers. Since our review in issue 9, the price has been lowere, making it even better value against its competition, and it's the only quality DAC we've found that's completely open source. Admittedly, you'll need some serious DSP programming skills to make this relevant, but there's already a huge community built around the SDR-Widget, which is exactly what the USB DAC 128 Mk II is built around.
www.henryaudio.com

Win!

BOOTY WORTH £4,250!

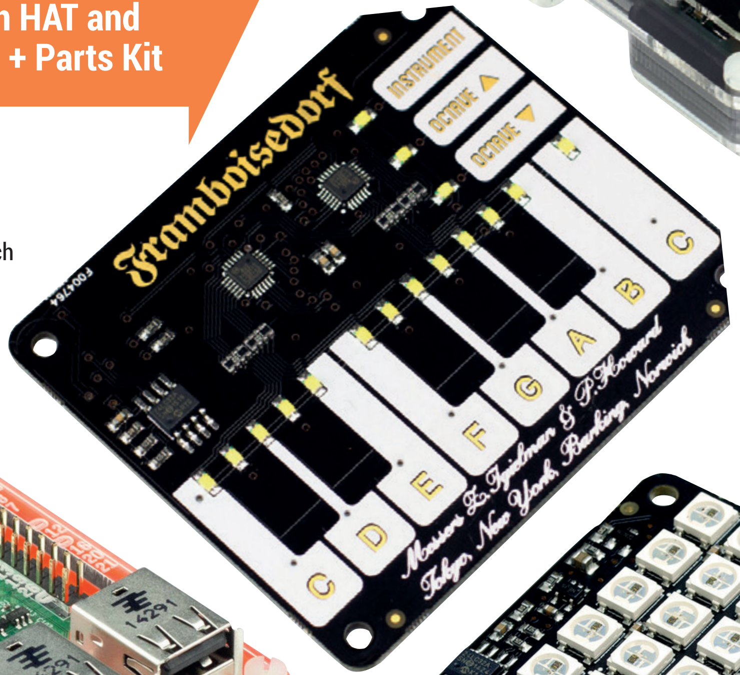
We've got together with the great folks at Pimoroni to give away lots of lovely stuff.

We're good friends with Pimoroni (it stands for Pirate, Monkey, Robot, Ninja). We spoke to them for one of our first interviews in 2014, and since then they've moved to new premisies and continued to be hugely successful. Pimironi's success has mirrored that of the Raspberry Pi – the company has sold over 150,000 of its Pibow Raspberry Pi cases, and now sells more than 1,000 different products from its home in Sheffield. Its Picade, the brilliant Pi-based table top arcade machine, was the UK's first Kickstarter project, and the company now makes lots of other ingenious 'HATs' for Raspberry Pi, augmenting Pis with everything from piano keyboards to migraine-inducing flashing lights. Which is why we've partnered with them to give away dozens of their best gadgets!

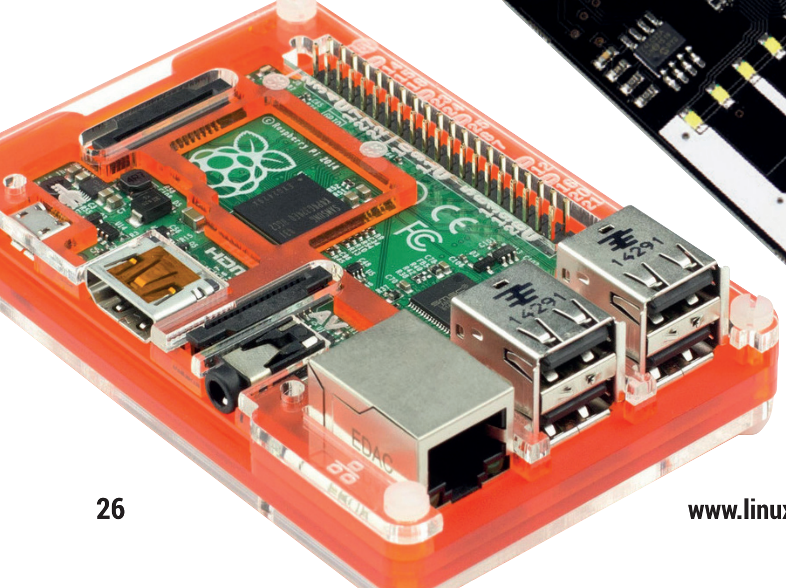
- 5 x Raspberry Pi 2 + Pibow Coupe, Picade (including 8-inch screen)
- 5 x Raspberry Pi 2 + Pibow Coupe, Picade Console
- 5 x Raspberry Pi 2 Starter Kit, Piano HAT, Display-O-Tron HAT, Unicorn HAT and Explorer HAT Pro + Parts Kit

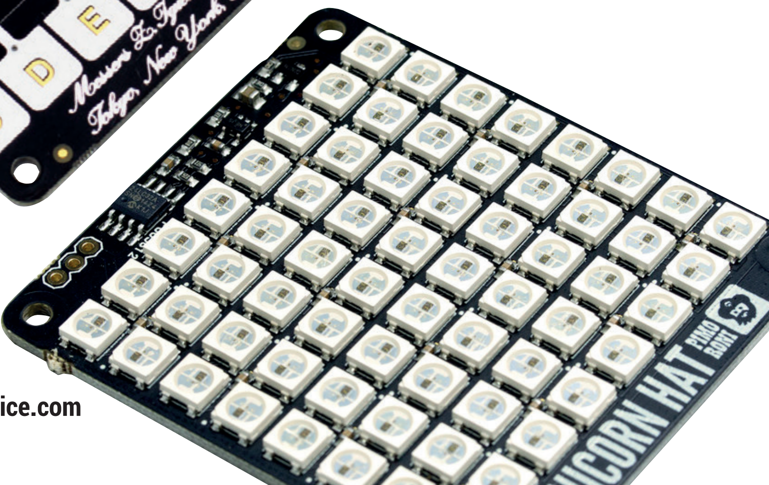
WIN ONE OF THESE EXCELLENT PI BUNDLES!

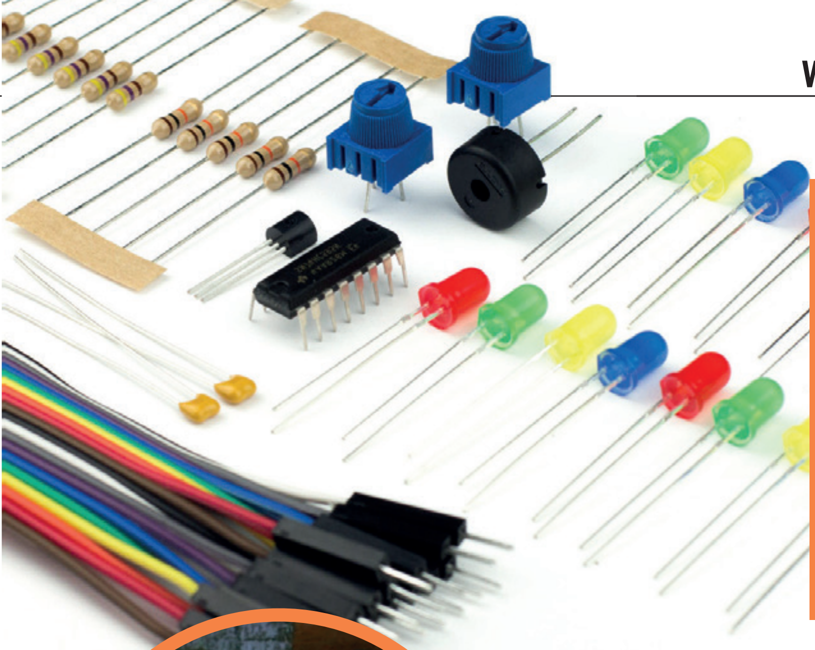
Use the Display-O-Tron to parse updates from **linuxvoice.com**.


Framboisedorf
INSTRUMENT
OCTAVE
OCTAVE
C
D
E
F
G
A
B
C
Messrs. Le Fjeldman & P. Howard
Tokyo, New York, Barking, Norwich

Pimoroni's lovely cases accomdate their HATs, such as the Piano HAT and the Unicorn HAT (both right)







HOW TO ENTER

We've hidden ten penguins throughout this issue of the magazine. After you've found them, email their page numbers, with your name and postal address to: picomp@linuxvoice.com (only one entry per person please)

We're giving away five complete Picade kits, five console kits and five starter kits, including everything you need for Pi-based fun.

MORE SWAG!

We've got more Pimoroni swag to give away throughout November and December. Follow us on Twitter @linuxvoice and listen to our podcast for more details.

TERMS & CONDITIONS

Competition runs from 9 November 2015 until 11.59pm 31 December 2015. Only one entry per person. No purchase or payment necessary. Winners wil be chosen at random from entries with the correct page numbers. No cash alternatives. We'll announce the winners on http://linuxvoice.com and email the winners directly. Prizes unclaimed after 31 January 2016 will be re-assigned. We reserve the right to modify this promotion and replace items with others of equivalent value. The prize value is current as of 29 October 2015. We respect your privacy and will only retain your details for the purpose of running this competition.

(SHE)BANG OUT OF ORDER

Flamewars and insults on mailing lists are driving developers apart. **Mike Saunders** asks: can Codes of Conduct/Conflict save the day?

Imagine you've spent weeks working on a project in your free time – some code for a popular Free Software application. You've devoted hours to your work, refined your code, tested it for bugs and written some documentation. Full of confidence and optimism, you submit your patch to the program's mailing list, hoping that it will be accepted into the next release – or at least you'll get some constructive feedback. But no. Your efforts are instantly dismissed with these words: "Your code is crap, you suck, and you should never have been born."

Sounds extreme, doesn't it? Fortunately, such behaviour is rare in the Free Software world. We've observed interaction in all manner of FOSS projects over the last couple of decades, via mailing lists, IRC channels and real-life meetups, and most people are friendly and patient. But as Linux and FOSS grows, the amount of hostility, abuse and threats on mailing lists and forums is expanding at an alarming rate too. Most recently, Sarah Sharp, one of the relatively few female kernel hackers, said she no longer wants to contribute due to the "toxic" environment around the kernel community. She had previously criticised Linus Torvalds for his acerbic rants and flowery language on the mailing list.

Meanwhile, *Systemd* head-honcho Lennart Poettering has described the FOSS world as "quite a sick place to be" after receiving an onslaught of abuse and even death threats. Because of all this, many projects are now putting into place Codes of Conduct (or Conflict): documents listing rules to which the community should adhere, and guidelines for dealing with disagreements. But will they work? Is it sad that we need these guidelines in the first place? And why is the internet so angry?

The answers are complex as we'll see, and reflect a history, culture and mindset that goes far beyond lines of source code. We spend 90% of our time thinking about technology at Linux Voice, but the people behind it – with their own sets of views, problems and quirks – are fascinating as well.

It might seem easy to pinpoint the crux of the problem: Free Software developers are 99% antisocial male nerds who live in their parents' basements and don't understand anything about human interaction, right? Well, this argument might have held some water back in the early 1990s, but even then, it wasn't all about reclusive shut-ins. Even when GNU, Linux and FOSS was largely the domain of hobbyists, many contributors were university students, retired Unix admins with families, and other "normal" developers.

Does a fish rot from the head down?

Fast forward to today: a large chunk of development work – even the majority in some projects like the Linux kernel – comes from full-time developers working 9–5 jobs in offices around the globe. Even for those hackers who work from home, the vast number of conferences, meetups and hackathons mean that developers meet up in person very regularly. Writing FOSS code is a respectable, social job, so we can't simply ascribe negative behaviour to the FOSS world being a load of socially inept übergeeks who never see the light of day.

Lennart Poettering has remarked that on the Linux kernel mailing list, "the fish rots from the head down". In other words: Linus Torvalds has set the standard for communication, and it only gets worse from there. Long-time kernel developers are used to Linus's epic rants, in which he thoroughly lambasts other hackers for their failures, mincing no words when he wants to get his point made.

One argument in favour of the Torvalds-style response is that it saves time in the long run. Take these two scenarios:

Dev: Hi Linus. Here's a patch that adds feature X to the kernel. What do you think?

Torvalds: Hi there Dev. Thanks for sending the patch. Well, kudos for giving it a try, but I'm not really sure it's the right approach. Maybe you could try it slightly differently?

(Two weeks later) Dev: Hi Linus. I've reworked the patch and made it slightly different...

Torvalds: Hi Dev. Thanks again, but it still doesn't fit into the way we do things in the kernel. etc. etc.

This back-and-forth exchange of emails and patches could drag on for months. Contrast it with this:

Dev: Hi Linus. Here's a patch that adds feature X to the kernel. What do you think?

Torvalds: This is completely broken and entirely unsuitable for the kernel. Throw it away.

This response is more brash – and arguably impolite – but it gets the message across much more quickly. The developer in question may feel hurt that Torvalds doesn't like his/her code. but at least he/she won't spend weeks or months trying to 'fix' something that will never be accepted any way.

Lennart Poettering is no stranger to online abuse, but says he looks beyond it and focuses on code.

> "Linus Torvalds is a role model .We may accept the odd rant, but what happens when others try to emulate his ways?"

Who's your daddy?

However, the problem runs deeper. Many developers have said they don't object to this level of directness, but the insults go too far. In one of Torvalds's famous tirades, he said that developers who write code in a certain way "should just be retroactively aborted". Some would argue that such statements are so clearly over the top that they're not meant to be taken literally – Torvalds doesn't actually want to kill people. And others have noted that Torvalds's quips always focus on a person's abilities as a coder, and not personal aspects.

The big issue here is: Torvalds is a role model for many younger and less experienced hackers. We may accept the odd hyperbolic rant when he's deeply disappointed in another (senior) developer, but what happens when others try to emulate his ways? Some greenhorn developers may assume that it's perfectly normal to post abuse to the mailing list – and the more abuse, the more they'll be seen like their hero. They don't understand how Torvalds thinks, how such outbursts are extremely rare, and how they're about code. No, they think it's cool and trendy to hurl around obscenities and abuse.

CODES OF CONFLICT

The solution may lie in carefully crafted guidelines...

In early March 2015, 60 kernel developers signed off a patch that could remedy the situation. The 'Code of Conflict' – a short 223-word text file – was created by long-time kernel hacker Greg Kroah-Hartman and accepted into the mainline source tree by Linus Torvalds. For such an important project like the Linux kernel, you might expect this document to be very specific in its demands and expectations of the community, but it's actually rather vague. For starters, it alludes to the fact that you need a thick skin as a kernel developer:

"Your code and ideas behind it will be carefully reviewed, often resulting in critique and criticism. [...] This development process has been proven to create the most robust operating system kernel ever, and we do not want to do anything to cause the quality of submission and eventual result to ever decrease."

So in other words: you won't be handled with kid gloves, we will be harsh if your code is bad, and the system has worked so far. But this doesn't sound like much progress, does it? If we can carry on as before, what's the point of having a Code of Conflict in the first place? Well, the second section deals with that:

"If however, anyone feels personally abused, threatened, or otherwise uncomfortable due to this process, that is not acceptable. If so, please contact the Linux Foundation's Technical Advisory Board."

This isn't a silver bullet solution, but it provides something that never existed before: an official way to report and register bad behaviour. If you're on the receiving end of personal abuse, no longer do you have to suck it up or simply quit kernel development, but you actually have an avenue to (hopefully) get the situation resolved. The Code of Conflict also signs off with this positive thought:

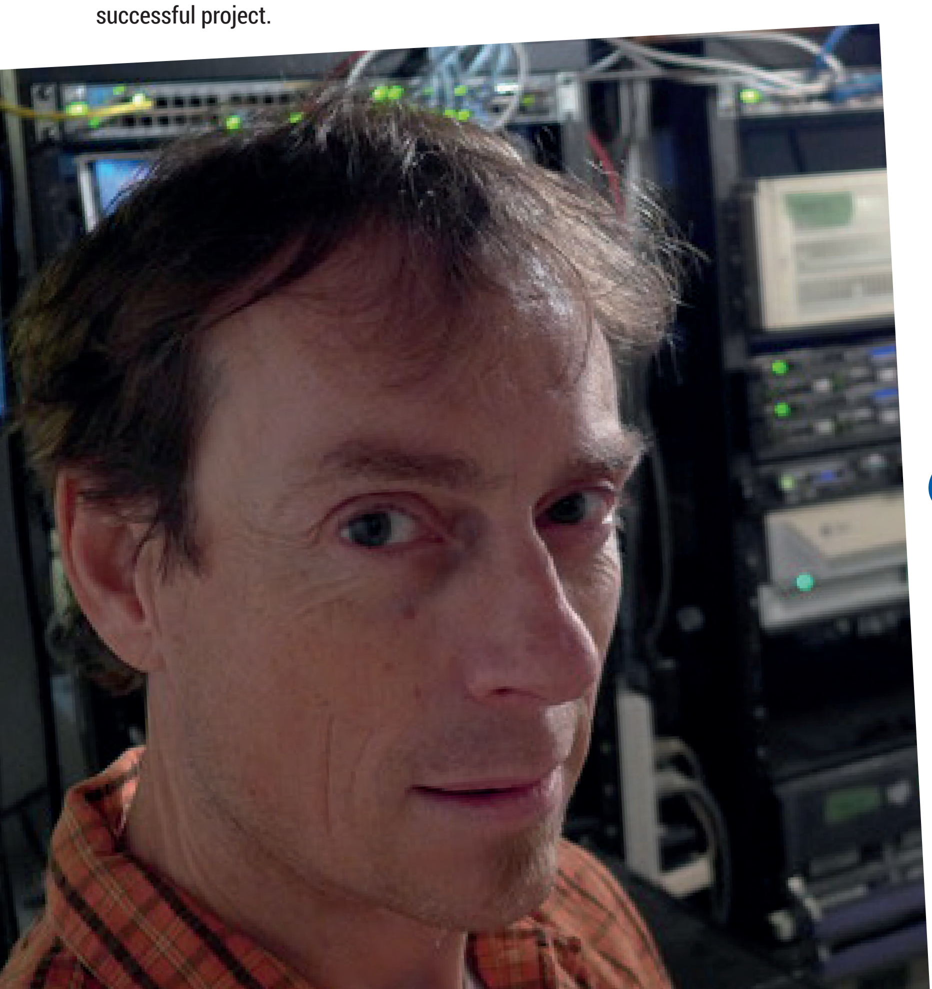
"We are all humans, and frustrations can be high on both sides of the process... keep in mind the immortal words of Bill and Ted, 'Be excellent to each other.'"

So there are very few specifics in the document that actually define what abuse is or what the acceptable level of communication should be. Some may find the Code of Conflict deeply lacking in that respect, but we think it's a good start. By and large, the Linux kernel has been an enormous success, so let's try to fix the problem with a general solution, rather than requiring everyone to read a 5,000-word document and sign it off before contributing – like some kind of tiresome End User Licence Agreement.

The LLVM approach

Since the Linux kernel got its own Code of Conflict, some other notable FOSS projects have adopted one as well. *LLVM*, the development toolchain providing some healthy competition to *GCC*, took a different approach to the kernel's document with a whopping 1,361-word file (**http://tinyurl.com/llvmcoc**) that goes into many more specifics. It describes in depth how mailing list posters should be patient, welcoming and respectful, and gives concrete examples of behaviour that should be avoided, including: violent threats; discriminatory jokes; personal insults; unwanted sexual attention; and personal information ("dox").

Theo de Raadt founded OpenBSD after being expelled from NetBSD for abusive behaviour, and has since gone on to run a successful project.



"LLVM's Code of Conflict gives concrete examples of behaviour that should be avoided, including violent threats and unwanted sexual attention"

LLVM has adopted a detailed Code of Conflict, whereas the Linux kernel's essentially says "just try to be nice".

Similarly, the *LLVM* guide provides a much more detailed set of steps for reporting bad behaviour and how it will be resolved. The LLVM Advisory Committee will get together and review the incident, suggesting possible resolutions: the misbehaving developer could be given a private reprimand if the incident is minor, or asked to make a public apology. If it's a more serious case of threats or personal abuse, that developer could be asked to take a week off to cool down, or be permanently expelled from the project (with a chance to appeal the decision).

So we have two types of Code of Conflict: the Linux kernel's short-and-vague approach, and *LLVM*'s very detailed document. Which one will be more effective in the long run remains to be seen, but we prefer the former. Trusting developers to use their intuition and know what's right or wrong seems more fitting to the open and diverse community around FOSS – but maybe some people will simply still not get it, and a more precise set of guidelines will be necessary in the future.

Saving face

Those of us who've grown up in the western world get used to a certain amount of 'banter': jokes, jibes and even the odd spate of personal insults here and there. We don't want to hurt anyone's feelings, but we also don't get deeply offended by such natter at the pub or between friends. Consequently, we carry a lot of this over to our communications online – often with liberal use of emoticons to make it clear that we don't mean anything truly offensive.

But for many cultures – especially in the Far East – the concept of 'face' plays a much bigger role. People carry a sense of dignity based on their position in a social group, and if that sense is scuffed or damaged by abusive criticism, the effects can run deep. Imagine a Chinese or Japanese kernel hacker being told on the mailing list that they are utterly rubbish, their work is worthless, and they should turn off the computer forever.

Many of us would find such remarks unpleasant, but not care what that person thinks and carry on with our work. We'll then joke about that person at the pub. Whereas the Chinese or Japanese developer may feel deep shame in being humiliated in public, and lose face among colleagues or friends who also happen to be on the list. Of course, some of this is a stereotype and there are developers in every country who respond differently. But if we want to bring talent from around the world to Free Software development, we need to be aware of cultural differences, 'saving face', and craft our criticism correspondingly.

It's not all bad...

One of the most famous forks in Free Software history is OpenBSD, the operating system that forked from NetBSD back in 1995. A year earlier, the NetBSD Core Team had expressed concerns that one of its most prominent developers, Theo de Raadt, was being abusive to other hackers on the mailing list and dissuading others from contributing to the project. The problem was described to de Raadt in private:

"Your abusive actions have seriously impaired the success of the NetBSD project in several ways. Your actions have driven away developers or potential developers, and have alienated many users. They have also squandered much of the good will that various people have directed at the project."

Meanwhile, on the public netbsd-users mailing list, the Core Team expressed their decision with regret:

"On December 20, Theo de Raadt was asked to resign from the NetBSD Project by the remaining members of 'core'. This was a very difficult decision to make, and resulted from Theo's long history of rudeness towards and abuse of users and developers of NetBSD. We believe that there is no place for that type of behaviour from representatives of the NetBSD Project, and that, overall, it has been damaging to the project. This decision was difficult to make because Theo has a long history of positive contributions."

What followed was an epic series of mailing list posts and private mails, all of which de Raadt has archived at **www.theos.com/deraadt/coremail.html**. The file contains over 52,000 words – so not some light bedtime reading – but we spent a few days going through it all. In summary: de Raadt had behaved extremely inappropriately, using personal abuse and sexual references to other developers.

In the end, de Raadt left NetBSD, forked it into OpenBSD, and now has a more popular project today (with a different focus: full-on security rather than portability). Despite his abrasive personality, de Raadt took many NetBSD developers with him and runs a successful project today – arguably for the same reasons that makes the Linux kernel a success under Torvalds. (And de Raadt himself has said that he's "not as angry" as he was 20 years ago.)

Of course, the OpenBSD community is tiny in comparison to Linux, so there aren't as many eyeballs watching how developers interact. Maybe one day OpenBSD will expand enough that the developer base is diverse enough to require a Code of Conflict. And who knows – maybe someone will fork OpenBSD into another project, and the cycle will continue forever...

FAQ

Software Defined Networking

Redefine your infrastructure on the fly with the latest network technologies.

BEN EVERARD

Q Networking's all about hardware. Cables, routers, interfaces, that sort of thing. Where does the software come into it?

A Things like routers and firewalls have networking hardware, but they also all have software that controls the hardware. This software does things such as decide where packets should be sent (if indeed they should be sent at all).

Q Ah, I think I've seen that. At home, I've got a Wi-Fi router with a HTML control panel that enables me to block ports, forward data and that sort of thing.

A That's exactly the sort of thing we're dealing with. In Software Defined Networking (SDN), we talk about the control layer (which is the software that manages the hardware), and the data layer (which is the actual networking hardware itself).

On most current networking equipment both of these things run on the same device, so in the case of your home Wi-Fi router there's one box that has both the networking hardware and the software that provides you with the configuration options. In an SDN setup, these two aspects are separated, so the control layer runs on a separate machine to the data layer.

Q What, so instead of one router, I'd have to have a hardware router and a server to run the control software? What's the point in adding that complexity?

A Well, if you're just running a single router, there's not much point in SDN. However, if you run more than one router or switch then a single machine can be the control layer for all of them. This means that rather than managing a single piece of hardware, the control software can handle all of the hardware at the same time. As well as being able to manage multiple pieces of hardware, the control software will also be able to see the whole network, so will be able to make more intelligent decisions about how the network should be configured.

The end goal of SDN is a network that can quickly and easily adapt as the uses of the network change. This goes hand-in-hand with things like virtualisation technology, which enables you to quickly and easily change the software stack running on hardware. SDN isn't a silver bullet to solve all a businesses IT problems, but a more flexible IT setup should enable a business to be more flexible in its operations.

Q That makes sense. I guess this means that you need a way for the control layer to communicate with the data layer. Is there a standard for this, or does each hardware manufacturer do it differently?

A The most popular way of controlling data layer hardware is with the OpenFlow protocol. This works in exactly the way we've just described. There's an OpenFlow controller that handles the control layer, and hardware that handles the data layer. Using this, you can combine hardware from any manufacturer and any controller provided they all support OpenFlow. This is known as the southbound protocol.

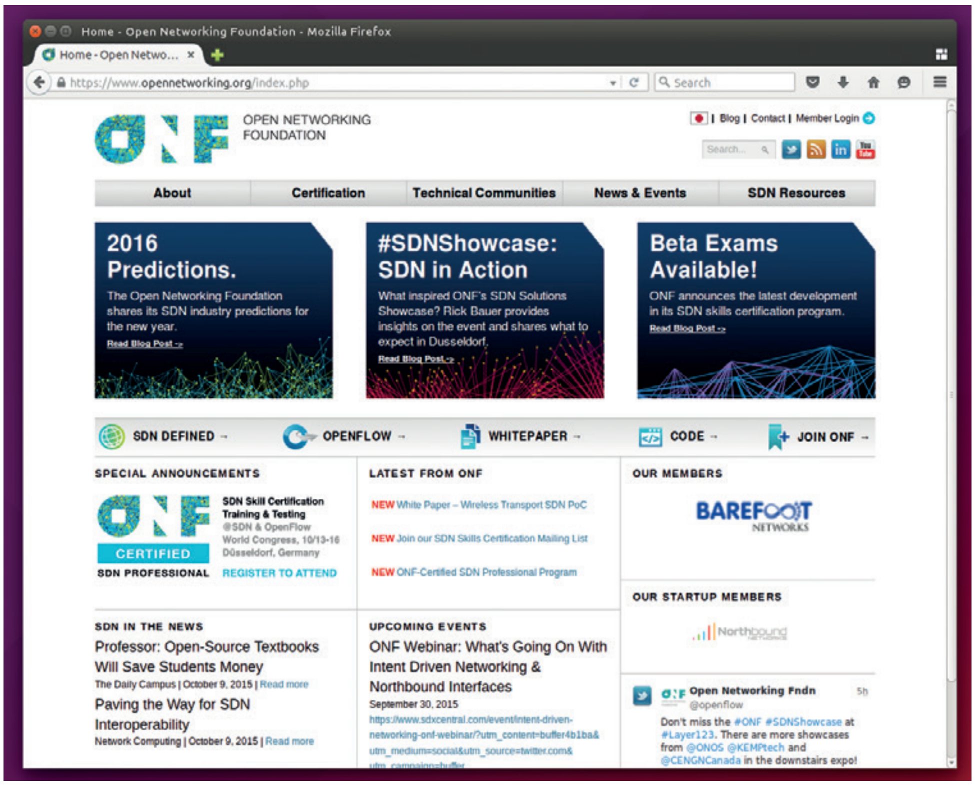
The SDN controller should also enable software to run on it from above. This software is know as the applications layer, and the idea is that the setup will enable standard software in the control layer to run software in the application layer to run on any physical hardware in the data layer.

The software running in the application layer can then take advantage of the controller's power to configure the network in different ways. This could include, for example, a web app to handle levels of Quality of Service (QoS) across all nodes on the network or an algorithm for automatically balancing the load on the network. The connection between this higher-level software and the controller is known as the northbound protocol. There isn't yet a standard for northbound protocols, and different SDN controllers allow different software to run on them.

The end goal of Software Defined Networking is a network that can quickly and easily adapt as the uses of the network change

Q Northbound? Southbound? What's any of this got to do with a compass?

A Some of the terminology can be a bit confusing. Typically,



2016 Predictions.
The Open Networking Foundation shares its SDN industry predictions for the new year.
Read Blog Post ->

#SDNShowcase: SDN in Action
What inspired ONF's SDN Solutions Showcase? Rick Bauer provides insights on the event and shares what to expect in Dusseldorf.
Read Blog Post ->

Beta Exams Available!
ONF announces the latest development in its SDN skills certification program.
Read Blog Post ->

SPECIAL ANNOUNCEMENTS
SDN Skill Certification Training & Testing @SDN & OpenFlow World Congress, 10/13-16 Düsseldorf, Germany

LATEST FROM ONF
NEW White Paper – Wireless Transport SDN PoC
NEW Join our SDN Skills Certification Mailing List
NEW ONF-Certified SDN Professional Program

OUR MEMBERS

OUR STARTUP MEMBERS

SDN IN THE NEWS
Professor: Open-Source Textbooks Will Save Students Money
The Daily Campus | October 9, 2015 | Read more
Paving the Way for SDN Interoperability
Network Computing | October 9, 2015 | Read more

UPCOMING EVENTS
ONF Webinar: What's Going On With Intent Driven Networking & Northbound Interfaces
September 30, 2015
https://www.sdxcentral.com/event/intent-driven-networking-onf-webinar/?utm_content=buffer4b1ba&utm_medium=social&utm_source=twitter.com&utm_campaign=buffer

Open Networking Fndn @openflow 5h
Don't miss the #ONF #SDNShowcase at #Layer123. There are more showcases from @ONOS @KEMPtech and @CENGNCanada in the downstairs expo!

The Open Networking Foundation sets the standards for SDN, including OpenFlow, and its members include Google, Facebook, HP, Intel and almost anyone else who's anyone in the tech world.

whenever someone draws a diagram, SDN infrastructure has the data layer at the bottom of the page, the control layer in the middle and the applications layer at the top. The protocols are all in relation to the control layer, so the protocols going downwards from the control layer are called southbound, and the protocols going upwards are called northbound.

Q This SDN thing sounds useful. Are most corporate networks run on SDN now?

A The concept of SDN has been around since the late 90s, but it's only been a practical solution to enterprise tech needs since around 2013. Since it requires new hardware, SDN is only slowly catching on. However, many are suggesting that it will become a serious player in the technology scene in 2016 and 2017. Industry analysts IDC, for example, expect the worldwide SDN industry to be worth $8 billion by 2018. If you're an early adopter, or like to get ahead of the tech curve, you need to start investigating SDN now.

Q What's all this SDN stuff got to do with Linux anyway?

A There's nothing inherently Linuxy about SDN; however, given the prevalence of Linux in the data centres where many of these SDNs are running, and the flexibility of the Linux stack, it should come as no surprise to learn that a lot of the SDN hardware runs Linux. This includes things like OpenvSwitch and Microsoft's Azure Cloud Switch (ACS).

Q Hang on just one second. Did you just say that Microsoft runs its SDN on Linux?

A Yep! We were equally surprised. Microsoft's ACS is used in its Azure cloud data centre to control the hosted environment. If that's not a ringing endorsement for Linux as the base platform for SDN, then we really don't know what is.

Q If all these SDN systems are built on Linux, does that mean I can build my own SDN setup on my Linux box?

A First off, we'll just repeat what we said at the start, slowly and clearly: there's no point in using SDN if you only have one or two pieces of network hardware (eg routers or switches). The chances of you having a home network that would actually benefit from SDN is pretty small. Of course, just because there's no technical benefit in something, that doesn't mean you shouldn't dive in and do it anyway for the geeky fun of learning something new! SDN is both an interesting area and a rapidly growing aspect of system administration.

Assuming you don't want to go out and buy expensive enterprise-level networking hardware, you've got basically two options if you want to experiment with SDN. The simplest option is to use virtual machines. Using visualisation software such as *KVM* or *VirtualBox*, you can start machines with virtual network interfaces.

These interfaces connect to virtual networks. Usually, the visualisation tools you're using will connect them together in a fairly straightforward manner, but you don't have to use the technology that comes out-of-the-box. Instead, you can use more powerful virtual networking software such as OpenvSwitch (**http://openvswitch.org**), which understands the OpenFlow protocol.

Q Isn't there a less heavy-duty way of giving it a go?

A An alternative approach is to build some networking hardware using a Linux-based machine. You'll need a computer to build it on, which could be more or less any machine that can run Linux (including a small ARM device such as a Raspberry Pi or an Odroid). You'll need to add more than one network interface, which is easily and cheaply done using USB Ethernet interfaces, and then you'll need some form of SDN driver running on it, such as LINC (**https://github.com/FlowForwarding/LINC-Switch**).

Either of these methods (and indeed, both simultaneously) can be used to create the data layer of an SDN. On top of this, you'll need to create a control layer. There are a few open options here including Pox (**www.noxrepo.org**), Floodlight (**www.projectfloodlight.org/floodlight**), or Maestro (**http://zhengcai.github.io/maestro-platform**).

The process of setting up an SDN isn't completely straightforward, but if you go through it, you'll end up learning a lot about this emerging technology.

GIANUGO RABELLINO
MICROSOFT'S GNU WHISPERER

Graham Morrison simply walks into Mordor to discuss open source strategy with Microsoft's senior director of open source communities. But is all as it seems?

Saved from a career in law by a career at Microsoft, Gianugo Rabellino has played a huge role in Microsoft's burgeoning interest in open source. Microsoft is now a significant contributor to many projects, including the Linux kernel and a partner in five initiatives with the Linux Foundation. Thanks to customer demand and the prevalence of Linux in everything from phones to the cloud, along with the wiser stewardship of its new CEO, Satya Nadella, Microsoft has released a version of *Office* running on Android, and has even been working on its own internal Linux distribution – both things that were completely unimaginable 10 years ago. But how real is this change? We find out what's going on at the heart of Microsoft.

LV Why did you take on the job at Microsoft?
Gianugo Rabellino: This October is my fifth anniversary at Microsoft. I joined in 2010, coming all the way from Europe. I had an open source services company, and before that, and after that of course at Microsoft, I spent all my life in open source.

I'm a member of the Apache Software Foundation, contributed to various projects, founded the first official Linux organisation in Italy in 1994, so I go back to Linux 0.99 and a big pile of floppies. I had a C64 when I turned 14. My parents had a company and they had one of the first PCs.

LV So from a C64 to a PC, so not through the Amiga then?
GR: No, I missed the Amiga stuff and that's one of my regrets, but what can you do?

LV PCs are probably a much wiser choice anyway.
GR: In a number of ways, yes. So I found myself thinking that I almost got this degree in law but I don't want to be a lawyer, what am I going to do? I didn't have the time or the money to pursue another degree and then I found myself back in computing, stumbled into open source, fell in love with it. Luckily that was when the internet had started.

LV What was it that you liked about open source? You could have had Windows on your PC?
GR: I could have except that I was utterly broke. And I got into the BBS world. And at the centre point, I wanted to run my own BBS but I could only have one computer. And that was quite a luxury. I saved and bought a 386 because I heard that those things could run more than a program at a time, so I could run my BBS and use the computer at the same time, and lo and behold Windows 3.1 didn't do that. So I got into [IBM's] OS/2. I loved everything about it.

LV Yeah, it had multitasking.
GR: Yes, exactly. And then they did OS/2 3.0 and it was like OK, no that was a mistake. Friends brought a pile of floppies with Linux and it was like, oh wow, look at that – that's interesting! And then I thought, you know what,

Like a lot of the high-rollers we speak to, Gianugo got his first taste of Linux using a stack of Slackware floppy disks.

“The only way to achieve interoperability is by building open protocols, open formats, open standards – and that gets coupled more and more with open source”

Gianugo talks a good game, but the proof of the pudding is in the eating (and he was part of the team that open sourced .NET, so that's a tasty pudding).

instead of running a BBS, I really want to spread the word about this Linux stuff, so I'm just going to open up access. Dial up a number, get a login prompt, no password, you're in and you can play around.

And I would be on another terminal doing stuff while other people were absolutely tearing my machine to shreds. And one day, just by accident, I get a talk request. Do you remember?

LV Ah, the days of FidoNet.
GR: Exactly. And it turned out to be a professor at a university who was looking for help to set up a lab, and he said it looks like you know Unix why don't you come and help me out. I still had to prepare for my final exam, but a few months later I was pretty much a teacher's assistant of Computer Science managing a lab in my small town. And that's what got me into it, and it was a combination of "hey I got this operating system for free, wow that's awesome", and then I was lucky enough that I could actually dial into the university and I had the internet, which is something you couldn't afford back then. There was no web, those were the days of Gopher [a protocol that pre-dated HTTP]. And I remember sending my first traceroute and being amazed that my first traceroute went to Vienna, and I was like I'm in a small town in Italy and now my stuff is getting to Vienna. And then this guy told me, yeah but this is actually Vienna USA, it's not Vienna Austria. My mind was blown, and I realised I could just use IRC, back in those days where you could just use IRC.

LV We still put our magazine together through IRC.
GR: So then I came on IRC and I found people from HP, people from IBM, people from Sun and I understood that I had another shot at a career. I could actually learn because there's so many resources that I could use. And I started hacking and I never looked back.

LV That's really good.
GR: Yeah. I owe everything to open source software. This is the beginning and the end of it.

Microsoft were doing their first forays into open source and back then they were actually involved in OpenXML, which you might recall was quite controversial at that time. And my impression as an open source person was that Microsoft was not getting credit for what they were trying to do. Everyone was second guessing that Microsoft was coming from the angle that these guys were not right.

LV You mean in a general IT sense, nothing to do with open source?
GR: It was the attitude towards Microsoft. I really thought that they deserved more credit, and were genuine in their efforts to do more open stuff.

LV When was this?
GR: We're talking 2008, which is when I started (Sourcesense). Then when I decided for other reasons to leave Sourcesense and take a sabbatical, I sent an email to my contacts at Microsoft saying I'm leaving the company, going somewhere else, it was fun working with you, and they said hey do you want to come have an interview. So that's how I joined.

LV So Redmond has realised it needed some help because its open source work was very isolated?
GR: They were doing some bits of open source here and there. They were dipping their toes. But then it came to a point that there was clearly an executive mandate to do more, to explore. So I joined in 2010 and had 23 interviews to get the job, and I asked for those because I really wanted to talk to as many people as I could before making a big decision such as moving a family of four, including a two-month-old, to the USA and also taking a career that was built on open source, on my open source reputation, and joining Microsoft. But at the time I signed up, I was positive that there was a genuine willingness to turn this company around and make it become more open. And it wasn't just... lip service, they wanted to change the company. So that was a challenge that was really exciting for me.

> I owe everything to open source software. That's the beginning and the end of it

LV Can you say who at Microsoft wanted that change to happen?
GR: I have to give a lot of credit to my hiring manager, Jean Paoli, whom I've worked with for the last five years. He clearly had that vision, but he was also backed by a lot of executives at

Microsoft, including for a number of years Satya Nadella, the current CEO. He was the one who approved of our first big plan to build open source for Azure. There was a lot of executive backing behind this effort.

LV Did everybody understand the advantages and why it was important internally? All we saw was the outside image presented by people like Steve Ballmer.

GR: Steve Ballmer was 2001 and floppy drives. So fast forward a few years and what we got was probably the tail end of that phase when Microsoft was very reluctant towards open source. The 'no way' phase. And then, pretty much when I joined or shortly thereafter, we started a phase of genuine, open and honest asking of ourselves why. I mean, I'm open to the view of using open source, but give me a good reason. And of course there's a spectrum. It went from people who were highly sceptical to people who were very open, but I never met anyone at Microsoft who was under the idea that we are not doing it just because it was open source. I never saw that in my five years at Microsoft.

Recently, I think we squarely entered the age of 'why not', so give me a good reason not to. We're going to default to openness. At the same time, what I think happened, and I think it's as important as this Microsoft changes, is that open source changed as well. The Microsoft of floppy drives and the open source of floppy drives had become the Microsoft and the open source of the internet era, where everything is connected, products are shipped to the cloud. The software that runs your phone is coupled with your device. Those are new things and so openness has become a more nuanced concept.

Today, I don't like just talking about open source. It's important – it's absolutely crucial – but if you talk about open source alone, you don't bring open standards into the picture. If you don't bring interoperability into the picture, I mean, we need to see the same webpage and access the same email. It looks like magic, but there's a lot of technology behind it.

LV Wouldn't you say that Microsoft wasn't too worried about interoperability and open standards up until 2005–2006?

GR: I think it's interesting to see how Microsoft got interested in interoperability and, subsequently, went into open source – into openness in general. Because the only way to achieve interoperability is by building open protocols, open formats, open standards – and that gets coupled more and more with open source. Right now there are so many projects out there where it's really hard to say whether they are a standard or an open source project. What is the new container initiative? It's based on Docker. They have these four components that I care about: open source, open standards, interoperability and community development, because that's the other part.

What we learn about over the years is that open source is stuff that you throw at the wall. I mean, it's nothing. You're building the 'whole of the garden' code. It doesn't matter. Code by itself rocks. But I'm an *Apache* guy, to me it's community over code always.

LV Have you built a community within Microsoft so that those open projects will remain open?

GR: Absolutely, yes. And again, that maps to something that changed in the market. Gone are the days of five years until the next version of Windows or the next version of *Office*. The turnaround needs to be faster: Azure updates every day. Your Windows devices update nearly every day or on a weekly basis. We need that fast turnaround. And we need to make sure our products and technologies can embrace that model.

LV When you say open source, is it more important for the source to be open and readable with more of an emphasis on permissive,

Linux is no longer a cancer that attaches itself to everything it touches (cf Ballmer, 2001). Rejoice, oh my brothers!

or is it that it becomes more of a community project?
GR: An open source license gives you all of that. So I can read it, I can modify it, I can redistribute my modification. But the real value comes when those modifications get merged back in. When I talk about open sourcing products or technologies at Microsoft, I always say if you don't intend to accept contributions there's little point in what you're doing.

LV But initially Microsoft had quite specific Microsoft open source licences. Thankfully, these seem to be fewer.
GR: Yes, absolutely, they're gone now. We haven't been using them for years.

LV Coming from the Apache side of things, did you have anything to do Microsoft's new licensing regime?
GR: Marginally, but by the time I joined there was already very very little use of MS-PL [the Microsoft licences]. If anything, I nudged towards *Apache 2.0*, being an Apache guy, and recently we've switched more and more to the MIT Licence. But we realised that there was very little advantage in building our own licences, but back when Microsoft did that, I wasn't at the company then. It kind of made sense. That was the time when everybody was doing their own licences. And then it became a problem and we understood at Microsoft that we need to go where the community is and the community wants to coalesce and consolidate on a handful of licences. We don't need so many, we just need a few. And that is why today we have five or six licences and everything else is really in the long tail. We just follow that trend.

LV What would you say has changed the most at Microsoft since you joined in its attitude to open source, and also how does the wider world of computing outside of Microsoft feel about the company?
GR: When I look back at these five years, my impression is that we did everything organically and that's probably one of the reasons why we were successful at changing the company so much. We didn't jerk, we took it little by little. Small steps with a bit of a bottom-up, grassroots approach with executive coverage from the top, and always focused on 'the code talks'. It's a useless conversation about the specific open source project or technology without showing code, that's the whole idea. We have been consistently pushing on the same strategy for five years and little by little we made this stuff change.

When I interview people who want to work at Microsoft, want to work in my team, I always tell them you cannot be successful in changing a big company if you take a speedboat approach. You're not on a lake in a speedboat doing zig zags. You're actually the tug boat that needs to steer the container ship. And the only way you can do it is by using a lot of torque. And little by little, slowly but surely – it looks like you're not making progress – but then you look back and you see that the ship is turning.

LV So you're able to look back and see the difference between people at Microsoft five years ago and people now?
GR: It's consistent. It just took this much time. It was a journey. We had to demonstrate business value. We had to validate. So when I joined there was a hypothesis that we need to change, we need to become more open because it's good for us, it's good for our customers, it's good for the company.

LV Was there any pressure from outside Microsoft?
GR: Oh there was tons of pressure. We always listen to customers, that's one of the main reasons why. And it was a changed landscape. Imagine how the world has changed since, as an example, the inception of XML. XML meant that all of a sudden you could have a heterogeneous data centre in a number of technologies and still have those machines communicate with one another. And that brought our customers to say, you know what, we love your technology but we also love this other technology now and they can talk together and we want to keep it that way, we want to use both. So that was the pressure that started it all off.

LV Was there a change in attitude at Microsoft?
GR: There was a big shift in the market. The market went from a single-vendor

The Microsoft Open Source Programs Office has a small team of four people – if you fancy working to help Microsoft use more ~~Free Software~~ open source, why not apply for a job?

"Right now, we are at a point from an executive trajectory point of view, we are where we want to be: the era of the 'why not?'. We need to default to being open."

market, where you are married to your customer for a decade because of lock-in, to a much more dynamic market where customers could mix and match. They told Microsoft and they told all the other companies, "hey this is the way we want to operate going forward. We're going to have a heterogeneous set of technologies that can interact with each other, and we're going to keep you accountable for that". So we have an interoperability executive council, which is part of what my team has been doing for the past few years and we have large companies and startups coming to us and telling us to fix this, you ought to fix this.

Being more open is good for our customers, and it's good for the company

How far have you got in your mission at Microsoft?

GR: From this standpoint in turning the ship around, I think we've done it, we're done. To me, there were two major milestones: open sourcing of .NET – open sourcing of one's crown jewels if you like – and the second major milestone for me was folding MS Open Tech back, which meant taking open source to the next level, realising that open source is across the company.

Were you involved in the open sourcing of .NET?

GR: I absolutely was. I actually sit on the board of the .NET foundation.

It's probably one of the biggest and most positive things Microsoft has done for open source in that it's genuinely useful.

GR: And a few months after we announced open sourcing .NET, we announced *Visual Studio Code*. It's about meeting developers where they are. It's the ultimate evolution of the mixed IP idea. We went from "I want to have a little bit of everything in my data centre" to "I will need to have a little bit of everything because I'm in the cloud and that's what I'm running today"; it may not be what I'm running tomorrow, it may not be what I'm running next month. It may not be with this particular provider, it may be with somebody else. And I also have my private cloud and I need to connect everything. So interoperability is supremely important.

Don't get me started with devices, with the idea that, hey I build apps or web sites. Take a random room of five or six people and you will find a mix of probably twelve different operating systems, devices, screen sizes etc. And we still have to deliver a consistent experience. So the fact that we can do it, thanks to the work the Microsoft and many others did at places like the W3C, just thoroughly amazes me.

So what's next?

GR: Right now the mission of the Open Source Programs Office is to enable, simplify and promote open source across the company. We managed to turn the ship around and now we need to sail it.

Where are you sailing to?

GR: We're sailing to where the market wants us to sail. We know that there's going to be a lot of openness down the road, so we're heading in that direction. Now it's about taking all those little stumbling blocks that are still there, making sure the process is smooth. Making sure that when you ingest an open source package, you do your own due diligence, you make sure it's an appropriate thing to do and we need to make sure that the process doesn't take much [effort].

Right now my major concern is making sure that we will need to create a process to manage resources internally and I want that process to be as frictionless as it can possibly be because we are doing these things to speed us up.

LISTEN TO THE PODCAST

LINUXVOICE

WWW.LINUXVOICE.COM

BUY LINUXVOICE MUGS AND T-SHIRTS!

LINUXVOICE
FREE SOFTWARE | FREE SPEECH
"IT'S EASIER TO ASK FORGIVENESS THAN IT IS TO GET PERMISSION"
Rear Admiral Dr Grace Murray Hopper

shop.linuxvoice.com

REVIEWS

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

Andrew Gregory
Is building a scale model of Harlech castle out of abandoned laptops.

By the time you read this magazine, the world will have forgotten about the TalkTalk data leak/hack, in which a telecoms company in the UK mislaid a load of its customers' data (including bank details). That's a shame, because there are lessons to be learned.

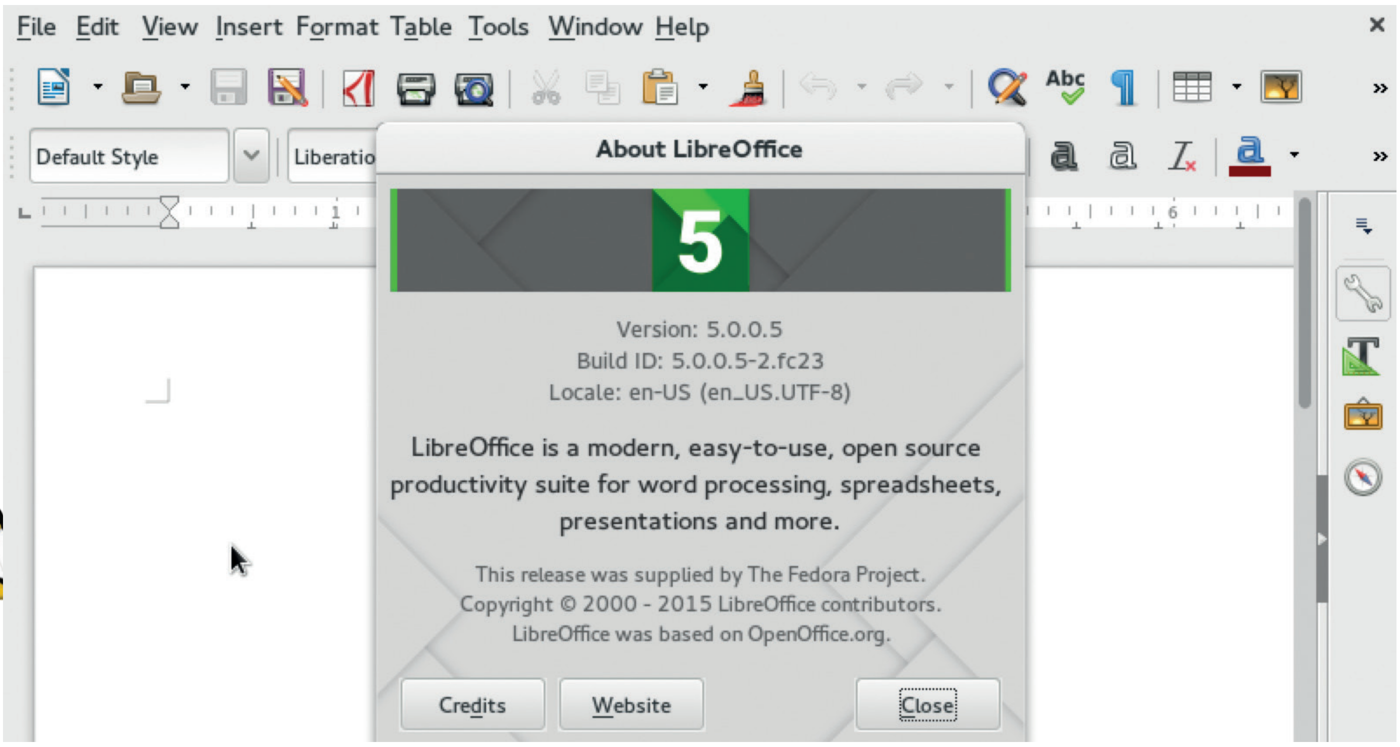
The most important of these is that you always need someone to blame. TalkTalk's share price fell from the day the hack was announced, a slide that was only arrested when police arrested a 15-year old from County Antrim in connection with the attack.

Always blame someone else

Are you keeping your customers' data unencrypted? Fine, carry on. Do you know of a breach, but doing nothing, hoping it'll go away? Also fine. But find someone to blame, pronto.

No doubt once this child has been prosecuted the CEO will leave for another gold-plated salary and they'll all pat themselves on the back for riding out the storm. But really, if a 15-year-old can hack your network, it's not the 15-year old that should be arrested.
andrew@linuxvoice.com

On test this issue . . .



Version: 5.0.0.5
Build ID: 5.0.0.5-2.fc23
Locale: en-US (en_US.UTF-8)

LibreOffice is a modern, easy-to-use, open source productivity suite for word processing, spreadsheets, presentations and more.

This release was supplied by The Fedora Project.
Copyright © 2000 - 2015 LibreOffice contributors.
LibreOffice was based on OpenOffice.org.

Fedora 23

Slightly later than expected, Fedora 23 is here to delight **Mike Saunders.**

Web www.fedoraproject.org
Developer Red Hat and Fedora Project
Platforms IA32, x86-64, PPC, ARM

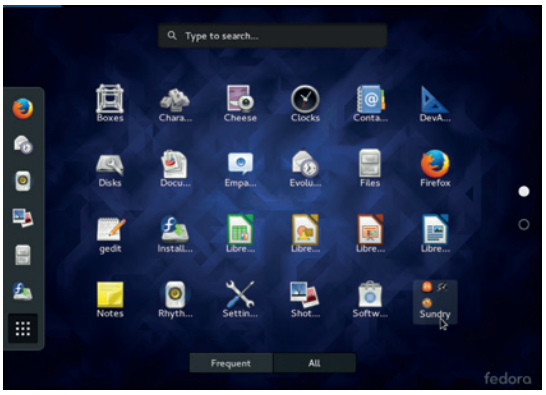
Fedora 23 was due to be released a few days before we went to press, but some last-minute spanners in the works meant that it was held back by one week. That's fair enough, we feel – time-based releases are a great way to ensure that software gets shipped at some point and doesn't languish in development hell, but it's worth being a bit flexible to ensure the final release is of decent quality.

As with the last few releases, Fedora 23 is available in multiple flavours: Workstation (for typical desktops and office environments), Server and Cloud. One major change that affects all versions is package hardening: where possible, binary executables are built as PIC (position independent code), which means they can be placed anywhere in the operating system's memory address space. Using ASLR (address space layout randomisation), the OS can make security holes less damaging, as crackers can't assume that certain bits of code are at specific points in RAM.

Waiting for Wayland

Desktop-wise, Fedora 23 ships with Gnome 3.18, and while many improvements have been made to Wayland, the X Window System still provides the default graphical layer. *LibreOffice 5.0* made it into the release just in time as well. In the Server flavour of the distro, there's a new role to set up Fedora as a cache server for web applications (using **memecached**), while the *Cockpit* administration interface now supports SSH key authentication and can work with Kubernetes to manage clusters of Linux containers.

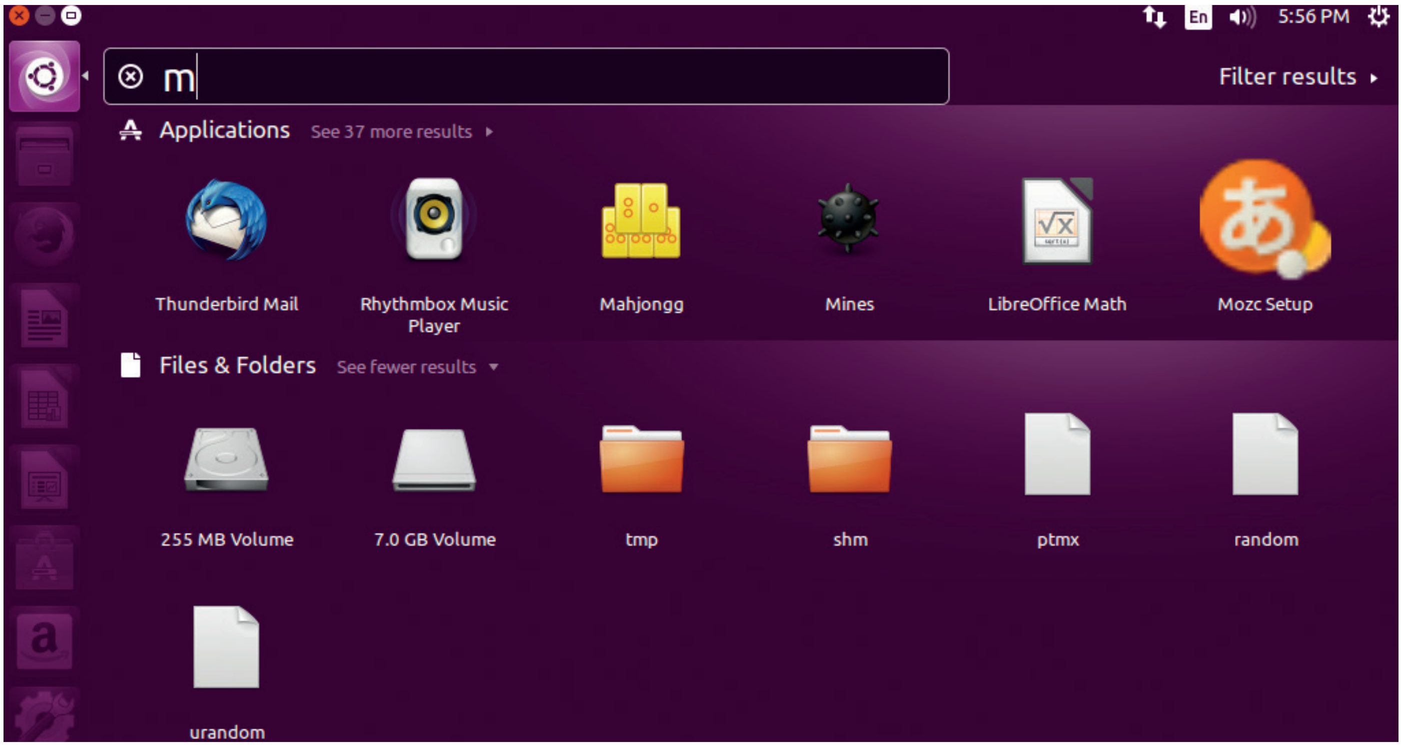
Other changes include *Docker 1.8* in the Cloud flavour, a migration from Mono 2.10 to 4.0, and Unicode 8.0 support. Along with the standard Gnome version of the Workstation release, there are "spins" featuring other desktops such as KDE, Xfce, LXDE, Mate and Cinnamon. On the whole it's a worthy upgrade: the switch to PIC binaries should have a positive impact on the distro's security in the long run, and easy access to the latest Gnome and *LibreOffice* releases make it a no brainer.



Gnome 3.18, the default desktop in Fedora 23, sports two new programs (*Calendar* and *Characters*) and Google Drive integration – see p43 of last issue for our review.

A shiny new desktop, updates to *LibreOffice* and more security – there's plenty to like here.

★★★★☆



Ubuntu 15.10 (aka Wiley Werewolf) Desktop and Server

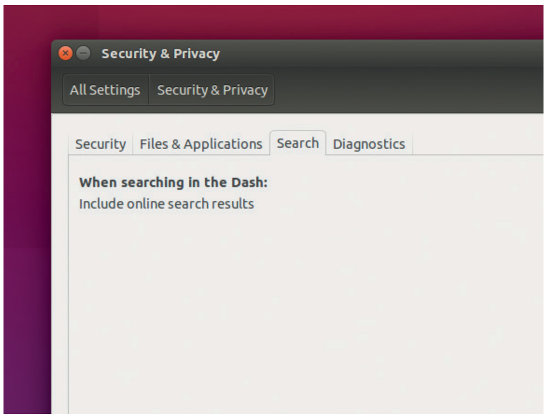
Ben Everard's purple werewolf costume left some people confused at Halloween.

Fire up Ubuntu 15.10 desktop and you could be forgiven for thinking you're running 15.04, or 14.10. Not much has changed in quite some time. There's a purple-ish geometric background, a set of blocky icons on the left-hand side and the same Unity experience that you'll either love or hate. As you'd expect, Werewolf comes with the latest upstream software, but otherwise, there's no reason to upgrade.

Let's now move swiftly on to the Server edition of Ubuntu 15.10, where there are some pretty big changes afoot. The biggest of which is the new OpenStack installer (*Autopilot*). It's a little bit of a shame that in 2015 an easy installer for software can be considered a feature. However, OpenStack isn't an easy system to set up, and having a simple path to running a private cloud will make Ubuntu a much more attractive option for people taking their first foray into this system.

LXD, Canonical's container management tool, is now shipped by default. This isn't a huge change, since *LXD* has already been available for some time, but by pushing it into every installation, Canonical is trying to get people into its own tool rather than alternatives such as *Docker*.

Users with heavy network loads may be interested to see the inclusion of the Data Plane Development Kit (DPDK) in the latest version of Ubuntu. This set of drivers and libraries enables users to handle network packets far more quickly than with traditional kernel drivers.



When searching in the Dash:
Include online search results

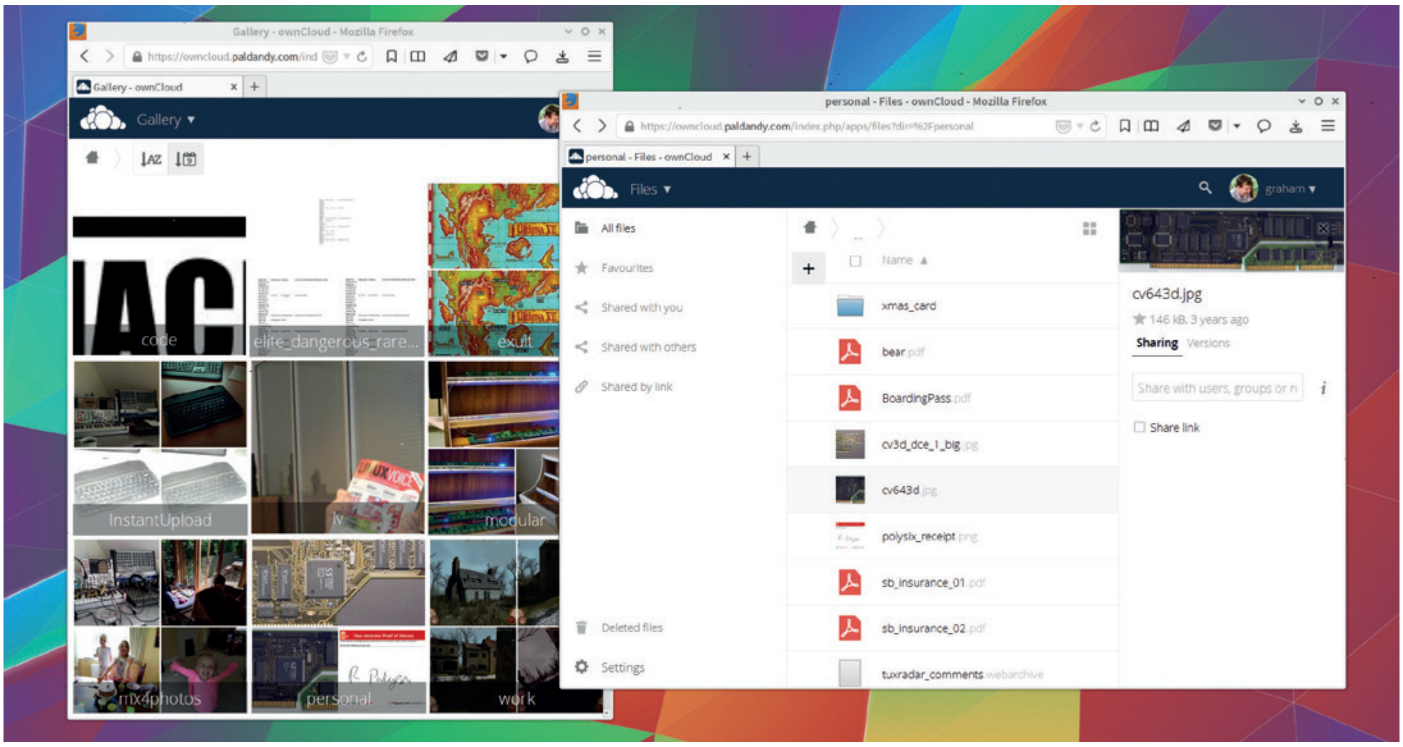
Ubuntu still includes online results in local searches, but it's easy to disable if you want to increase your privacy.

Like all regular versions of Ubuntu, 15.10 will only be supported for nine months, which isn't long enough for many organisations. However, these new technologies are a show of strength from Canonical six months before the release of the next LTS version (which will be supported for five years). If the new technologies prove to be stable, it will pave the way for the next release (16.04) to further cement Ubuntu's position as the leading OS for modern data centres.

Web www.ubuntu.com
Developer Canonical
Licence Various free software licences

Not much new on the desktop, but a strong sign of things to come in Ubuntu server.

★★★★☆



OwnCloud Server 8.2

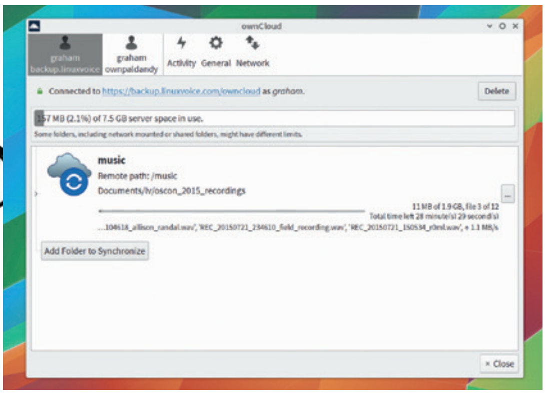
Thanks to this great update, **Graham Morrison** has [almost] managed to drop Google.

Web https://owncloud.org/
Developer OwnCloud Inc.
Licence AGPLv3

We use *OwnCloud* every day to put this magazine together. It's a fantastic piece of software that turns anything capable of being a server into the closest open source gets to Dropbox or some of Google's cloud-bases services. It's also rapidly evolving, and 8.2 is the second major update since 8.0 was released in February 2015. After manually updating our old 7.x servers, we've been able to use the automatic internal update feature for both 8.1 and 8.2, making the upgrades effortless.

Many changes for this release are visual. There's a small new menu adjacent to each file that provides download, delete and rename duties, for instance, while upload and file creation moves to a new '+' menu at the top. These replace the slightly clunky icons that appeared when you hovered over files and folders in previous versions. Clicking on the Details menu item also opens a new panel on the right where you can easily see the sharing status of your selection, along with a preview of its contents – and brilliantly, access to each version if the file has been modified. We love the way you no longer have to save a text file when you edit it, as all changes are saved automatically. But our favourite visual upgrade is to the Gallery view, replacing the old Pictures mode. You can sort your images by creation date, and the update time for us was much faster than with previous versions. You can also zoom and pan around your photos, making this the first time we've felt comfortable sharing a folder link with someone rather than using another online photo repository.

Administrators get a lot more control from the command line, including the ability to modify the number of versions and the amount of trash kept by the system, and the ability to encrypt and decrypt everything. And while it's not specifically part of the server package, we have to mention that the recent upgrades to the Linux desktop standalone client are brilliant. It no longer sucks CPU cycles and can connect to multiple servers at once. *OwnCloud* is a project that just keeps getting stronger.



Connected to https://backup.linuxvoice.com/owncloud as *graham*.

157 MB (2.1%) of 7.5 GB server space in use.
Some folders, including network mounted or shared folders, might have different limits.

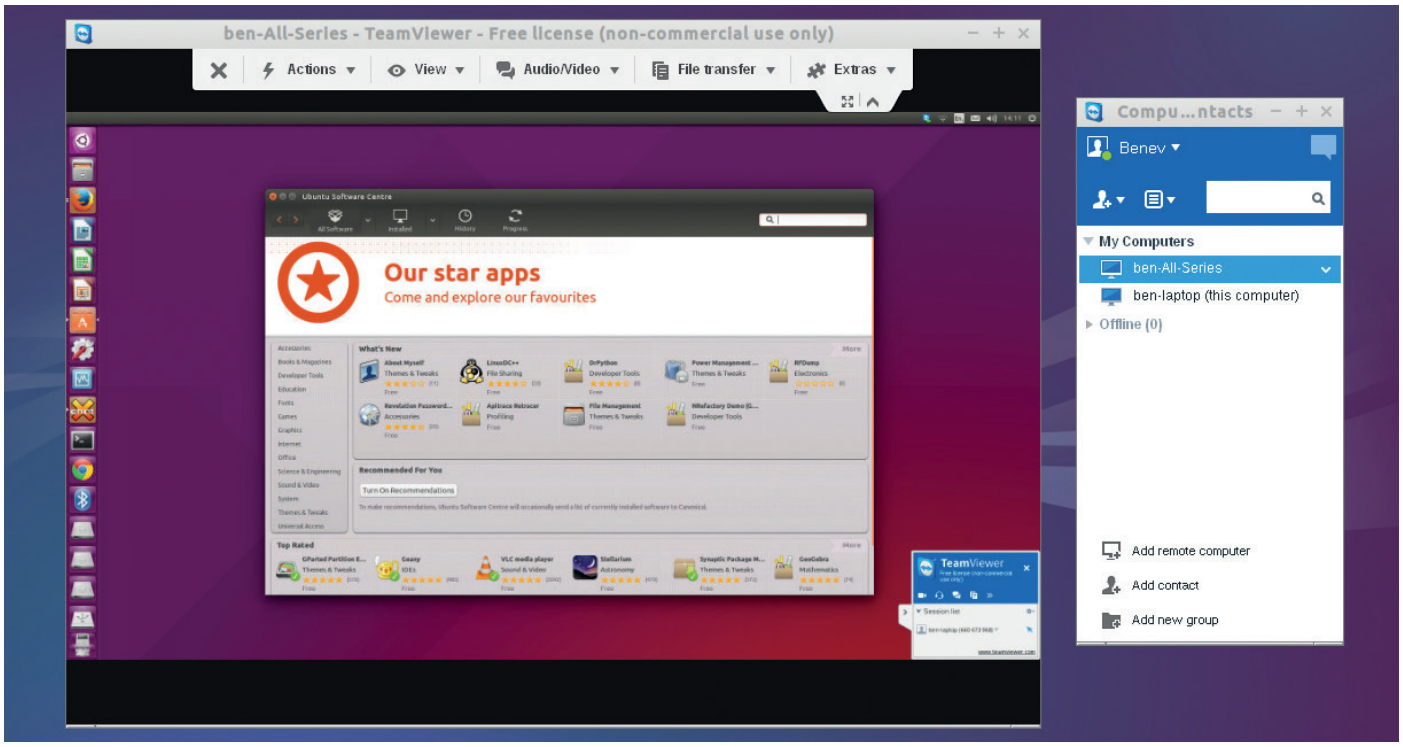
Remote path: /music
Documents/lv/oscon_2015_recordings

11 MB of 1.9 GB, file 3 of 12
Total time left 28 minute(s) 29 second(s)
...104618_allison_randal.wav', 'REC_20150721_234610_field_recording.wav', 'REC_20150721_150534_r0ml.wav', + 1.1 MB/s

The best thing about the new desktop client is that you can use it with more than one account and *OwnCloud* server at the same time.

Nothing touches *OwnCloud* for its feature set, or for its development speed.

★★★★★



TeamViewer 10

Ben Everard may be about to become the technical support team for his social circle.

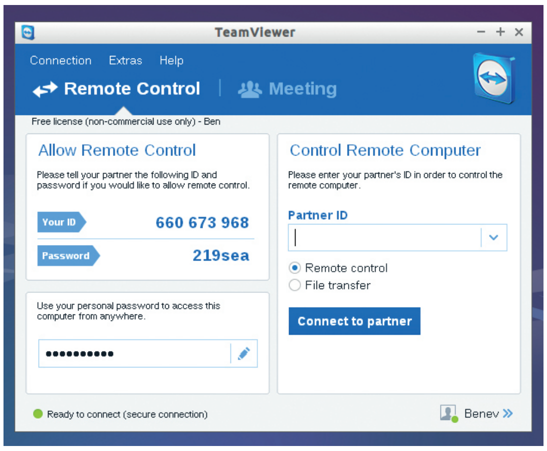
Linux has a surfeit of remote access tools, from the command line SSH to the graphical *VNC*. Many video chat tools also have screen-sharing capabilities. With all this, is there really a need for a proprietary option?

To answer that question, we have to look into the two things that *TeamViewer* does. First, as you may expect, it shares a computer's screen with a remote computer. The remote user can then take over the mouse and keyboard and use this as they wish. The most common usage for this is fixing problems. *TeamViewer* performs this task perfectly well, but so do other tools. *TeamViewer*'s real advantage comes in the second task it performs: managing these remote connections simply without needing an inbound TCP connection to the machine being controlled.

That all sounds a little technical, so let's go back for a minute. Imagine your friend calls you up and says they're having a problem with their computer. How do you connect to it? With SSH or *VNC*, you need an IP address to contact. This is fine for a server; it's probably even possible over a home internet connection, but it won't be easy to set up.

With *TeamViewer*, they'll just get a few numbers displayed on their screen which you need to enter into your machine, and you have access to their desktop. It's simple enough for you to be able to talk a technically inept person through the installation and setup in a few minutes; then you'll have access to their machine and can fix their problem for them. That alone is worth the full five stars from us. There are only two problems we can see: it's closed source; and if you make it too easy for people to come to you with their problems, they may never fix anything themselves again. We'll leave it up to you to decide how serious these problems are.

As well as Linux, *TeamViewer* works on Windows, Mac, Android, Windows Phone and Blackberry, so you can help people of any computer denomination.


TeamViewer
Connection Extras Help
Remote Control | Meeting
Free license (non-commercial use only) - Ben

Allow Remote Control
Please tell your partner the following ID and password if you would like to allow remote control.
Your ID 660 673 968
Password 219sea
Use your personal password to access this computer from anywhere.
••••••••••

Control Remote Computer
Please enter your partner's ID in order to control the remote computer.
Partner ID
Remote control
File transfer
Connect to partner

Ready to connect (secure connection)
Benev

You can try if you like, but these login details won't let you take over the Linux Voice network.

Web www.teamviewer.com
Developer TeamViewer
Licence Free for personal use or from £449

The easiest way of fixing other people's problems on just about any computer.

★★★★★

GAMING ON LINUX

The tastiest brain candy to relax those tired neurons

DEAR SAINT NICHOLAS...

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

The first Steam Controllers – on which much of the viability of Linux as a mainstream gaming platform hangs – have been shipped, and those lucky enough to have pre-ordered have given their thoughts online. At the same time, Valve has pushed back the official release closer to Christmas to iron out some of the kinks.

Responses have been positive, though the unusual double trackpad design has been reported to work better on some types of games than others, and it's expected that later versions will attempt to do more to make it a more viable option for FPS games. The controllers also had issues with running out of the box on some major Linux distros, though this has now been patched up.

Meanwhile, contracts have been secured in the UK, US and Australia to bring the Linux-powered consoles and their controllers to the biggest bricks-and-mortar game retailers in those countries, bringing PC gaming back to store shelves alongside consoles after a long absence. The sheer variety of hardware and pricing will be a challenge for vendors.

Having gotten a chance to try out SteamOS on a home made Steam machine, it's certainly impressive. As a gaming distribution, it takes plug-and-play to a whole new level, without having to wrestle with pesky graphics drivers or fine-tune things after installation. There is no doubt as to the ease of use to potential Linux newcomers, though its acceptance by the gaming community will remain to be seen.

Soma

Atmospheric sci-fi horror from the makers of Amnesia: The Dark Descent.

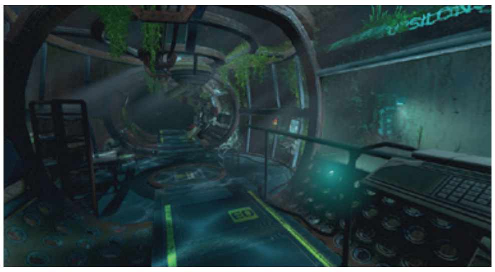
Penumbra and *Amnesia* are often considered to be among the scariest games of all time, but *Soma* focuses more on story than horror. It's far more mature than its predecessors, using philosophy as an theme to its sci-fi exploration-focused story.

The game's predecessors were considered terrifying due to the lack of self defence abilities, but also for their atmosphere and excellent sound design. *Soma* does these things well, though by this point the formula has been imitated to the extent where it has almost become a genre of its own.

As a sci-fi story-focused game, *Soma* is hugely successful, though the horror aspects can be half-baked. At times, it feels as if more could have been achieved if the game stopped doing what was expected of it and concentrated on the excellent plot. That said, the atmosphere achieved through the traditional means of horror really adds to the experience.

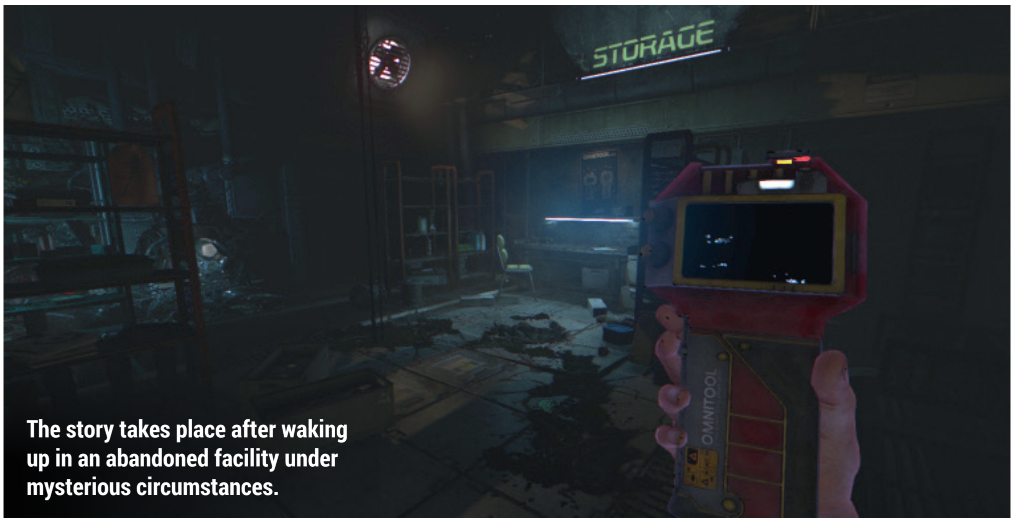
The story deals with some intriguing themes such as human consciousness and unfolds at an excellent pace as the mysteries of the underwater facility and its former occupants are steadily revealed. The loneliness of the experience and constant second-guessing as to the goings-on really pull the player into the game's world, and in effect also serve to bolster the horror aspects, which *Soma* does more effectively. It is these aspects that make the game undoubtedly better than its predecessors.

We recommend *Soma* – it's one of those games that create conversations between people who have played it, albeit in this case far more about the story and the issues it raises than the jumpscares.



Soma's mysterious underwater facility feels very much alive and is full of decay.

Website http://store.steampowered.com/app/282140 **Price** £29.99



The story takes place after waking up in an abandoned facility under mysterious circumstances.

The atmosphere achieved through the traditional means of horror really adds to the experience.

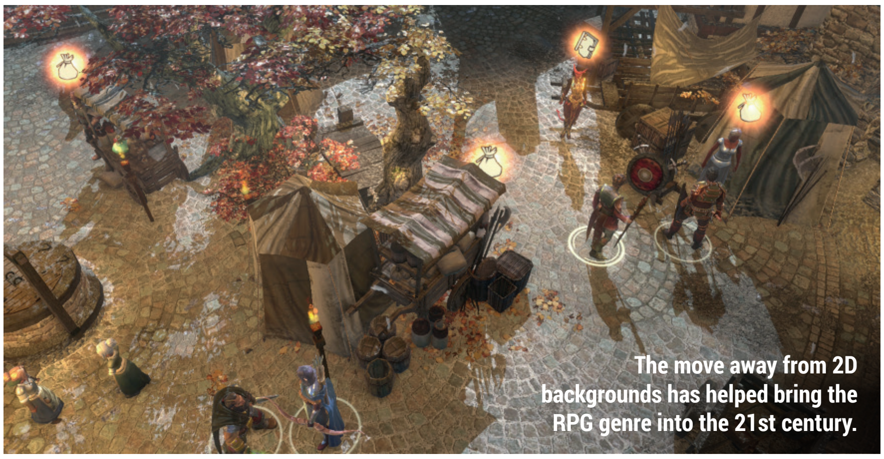
Sword Coast Legends

A well polished RPG set in the Forgotten Realms universe.

There have been many successful attempts at reinvigorating the RPG genre and *Sword Coast Legends* pulls it off exceptionally well. Rather than lazily relying on the success of *Icewind Dale* and *Baldur's Gate* and drowning in a sea of nostalgia, the game delivers 3D visuals that look great even when zoomed incredibly close to the characters.

There's still plenty of standard fantasy tropes of bustling towns and huge caves filled with monsters and bandits alike. Many of the Dungeons & Dragons crowd claim that the game's multiplayer mode is not true to the rulesets and have criticised it pretty extensively. Not knowing the least bit about D&D, I can only speak for the campaign, which has high production values, a solid story and excellent voice acting. That said, the combat and skill trees are somewhat superficial, which need not be a bad thing in order to attract a new generation of players, but seasoned RPG fans may be somewhat disappointed in this regard.

Website http://store.steampowered.com/app/325600 **Price** £32.99


The move away from 2D backgrounds has helped bring the RPG genre into the 21st century.

Grand Ages: Medieval

A pleasant trading simulator which could have been far more.

When *Grand Ages* was announced, it looked to be an exciting combination of the best aspects of 4X strategy games like *Civilization*, with the complex empire-building aspects of a grand strategy game. However, it turned out to be a somewhat superficial trade simulator with limited combat and building mechanics, and no political mechanics whatsoever.

Nonetheless, *Grand Ages* is by no means a bad game. It is fun and satisfying, while providing many hours of gameplay – even if it does feel like a missed opportunity at times. The campaign is well put together, with a story that takes place in the latter days of the Byzantine empire.

The gameplay often lacks balance, ranging from it being near impossible to turn a profit, to being incredibly overpowered and there being little challenge. However, a number of patches have already addressed some of these issues, and progression feels pretty solid most of the time. The game would suit fans of the *Patrician* series and other trade-oriented games the most, while grand strategy fans may also enjoy it.

Grand Ages clearly has an impressive engine and a good base for a solid franchise.

Website http://store.steampowered.com/app/310470 **Price** £29.99

ALSO RELEASED...

Wasteland 2: Director's Cut
Just over a year after the critically acclaimed *Wasteland 2* was released, it has received a complete graphical overhaul and a few very welcome additions. Among the additions to this *Fallout*-esque RPG are controller support, extra voice acting and more character customisability. Needless to say, the Director's Cut is a huge improvement on a game that was already extremely impressive.
http://store.steampowered.com/app/240760

The Beginner's Guide
From the makers of *The Stanley Parable* comes another equally fascinating experience, which diverges from the traditional mechanics of video games. The player explores the numerous creations of a game developer, taking them on an emotional journey that touches upon aspects of the human condition. The story is mostly driven by the narrator, who is also the creator of the levels, and it lasts a couple of hours at most.
http://store.steampowered.com/app/303210

Skyhill
This fun roguelike/adventure/survival game meshes together some seemingly different genres, with the player getting to enjoy the best of each. The challenge here is making your way down the 100 floors of a hotel by scavenging for food, crafting items and battling mutants to get through to the exit. There's also good narrative thrown in to add intrigue and remove repetition.
http://store.steampowered.com/app/382140

Exercises for Programmers

Ben Everard changes out of his Lycra – these are different exercises.

Author Brian P Hogan
Publisher Pragmatic Bookshelf
Price £15.99
ISBN 978-1680501223

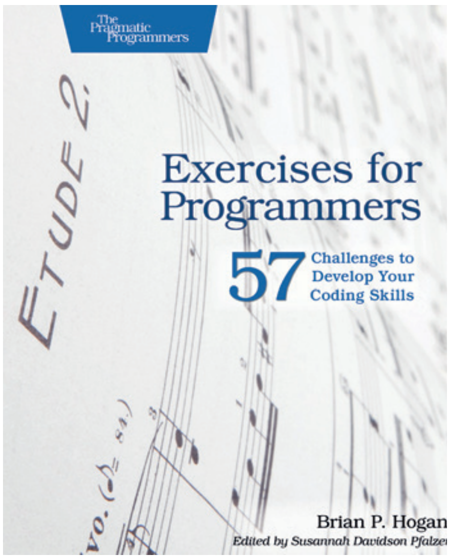
How long does it take to learn a new language? The answer to this question really depends on what you mean by learn a language. A decent programmer can probably pick up the syntax and basic usage of a new language in a weekend. After a month, they should be fairly comfortable. However, it can take a long time to really get you head around all the little bits that you need to know.

Exercises for Programmers is a set of challenges that are designed to cover every area of a language so that when you've covered them, you'll be able to program comfortably in that language. The tasks are there to test your knowledge of a language, not your skill as a programmer, so you won't need to come up with any novel algorithms or tricks to get through this book. The result is a slightly pedestrian set of exercises which didn't really excite us.

Exercises for Programmers takes you through everything from creating user interfaces to using web services for the purpose of getting data, all to force you to search through all the features, libraries and modules of your chosen language. These exercises should work with any language regardless of paradigm.

A useful, but slightly dull, workout to help you make sure you have a complete grasp of a new language.

★★★½☆



Musicians practise scales, so perhaps programmers should practise the routine parts of their craft as well.

You Code as a Crime Scene

Ben Everard's code always looks like a crime scene.

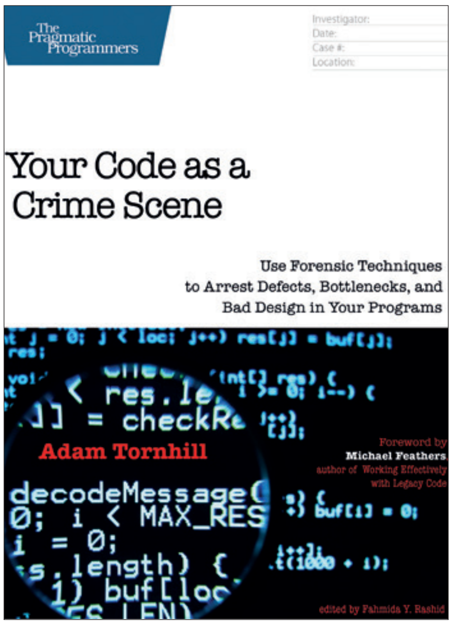
Author Adam Tornhill
Publisher Pragmatic Bookshelf
Price £23.99
ISBN 978-1680500387

It doesn't matter how good your code is, there are bugs in it, and one of the challenges of programmers is finding where they're hiding. In *Your Code as a Crime Scene*, Adam Tornhill introduces the idea of using forensic techniques to work out where they're most likely to be. The process is based on geographic offender profiling, which attempts to locate the likely location of a criminal based on the pattern of their crimes. Tornhill uses tools and techniques that attempt to locate likely places for bugs based on the location of complexity in and changes to the code base. This spatial mapping produces visualisations that highlight a series of 'hotspots' where bugs are likely to occur.

By identifying these hotspots, you can focus your bug squashing activities in the most fruitful places, and also use this knowledge to inform your development practices (should this code be re-factored?) and human resources (should more people know how this code works?). In small projects, developers can easily keep track of the whole codebase, and so identifying hotspots isn't very useful, but as projects get larger, it becomes more useful to know where problems may arise. Adam Tornhill takes the reader through a series of real open source projects to demonstrate the techniques. This isn't a failsafe approach guaranteed to leave your software spotless, but could be a useful weapon in the endless battle for software quality.

A novel approach to software analysis that could prove useful for managing large projects.

★★★★☆



The deerstalker and pipe aren't essential accessories to this book, but we highly recommend them.

Blockchain

Graham Morrison finds the first example of currency gentrification

Author Melanie Swan
Publisher O'Reilly
Price £15.54
ISBN 978-1491920497

This is a book that covers classic O'Reilly territory: it takes a technically challenging new area that could lead to a revolution, and explains why that technology is important. O'Reilly has done this in the past with both social networking and the smartphone economy, and Bitcoin is ripe for similarly skilled divination, especially as we've yet to see a practical analysis of the algorithms and the potential outside of digital currency. However, this isn't the book we're looking for. It's going to appeal to readers who want a more academic exploration of how blockchains (the indivisible list of all transactions behind crypto-currencies like Bitcoin) are likely to operate outside of the darknet.This isn't surprising when you look at the author's background: Melanie Swan is the Founder of the Institute for Blockchain Studies and is currently studying for a Contemporary Philosophy MA.

But everyone already knows about Bitcoin. The ongoing drama surrounding the bankruptcy of the Mt. Gox Bitcoin exchange is still making headlines, even in the general press, and the alleged theft of Bitcoins by a US secret service agent while investigating the Solk Road black market portal is even more newsworthy, as is the plight of Silk Road's creator, Ross Ulbricht, who was given a life sentence for his crimes.

With these kind of magnetic back stories, it's surprising that this book does very little to pull in casual readers who have discovered Bitcoin through this coverage. After a lengthy preface, the book wastes no time on technicality by diving into second guessing what the algorithms behind Bitcoin might lead to – the reinvention of financial services, for example, or self-signing contracts. It does a good job of putting all predictions into one place, even expanding into the future with Blockchain 3.0, and what different kinds of blockchains might lead to. This is where the book succeeds, and where potential readers will get the most.

Let's hope the future of Bitcoin doesn't include the NSA running its own mines.

Open University

But for us, it reads like a pre-peer reviewed research paper for potential investors, even finishing with scholarly sections on challenges, limitations and the classic 'Conclusion'. It's not what we were expecting, and we think says something about who the book is targeted at. Certainly not people interested in the specifics of the algorithm and how it could be subverted into different roles.

A description of the algorithms does appear in the first appendix, titled 'Cryptocurrency Basics'. It could be that Bitcoin makes sense without these technical foundations, but not for us. And we also found the complete lack of any diagrams disappointing, especially when blockchain transactions can be visual in a way that dramatically helps with comprehension. It left us mostly with a feeling of missed potential for anyone but a Bitcoin student.

An academically written and dry look at the potential of blockchains.

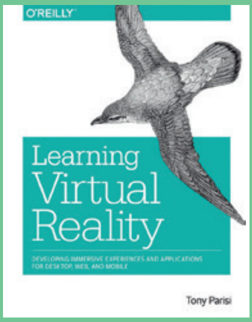
★★☆☆☆

Also released...

December 2015

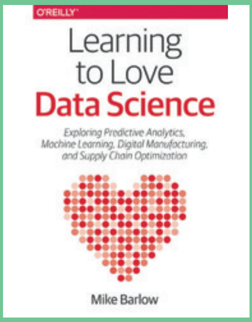
Learning Virtual Reality

We know the current craze for virtual reality is likely to go the same way as 3D movies, and be little more than a footnote in a couple of years, but we're genuinely excited by its revival (we loved playing the Virtuality arcade machines of the 1990s). This book is an introduction to developing for VR, featuring Oculus, WebVR and the ultra-cheap Google Cardboard.


We can't wait for Super Tuxcart in virtual reality.

Learning to Love Data Science

Data science is becoming mainstream, and is being used everywhere from manufacturing to journalism. This book is a sobering appraisal of its potential, complete with a few case studies. It's likely a good read for anyone in the position to try and do clever things with big data, which in the age of citizen journalism and open data, means almost anyone.


We thought all science was based on data.

Lego Power Functions

We love the idea of this book. It's full of small projects that showcase Lego's various Technic moving elements in modular re-usable forms. It's exactly what Lego is good at, and the reason why we all still enjoy messing around with the stuff. That the functions in the book have value in the real world of construction is an added bonus. We promise.

Build anything with Lego. Except political works.

GROUP TEST

Speed up your Linux box by switching away from a heavy desktop to a lightweight window manager. **Mike Saunders** weighs up your options.

On test

FVWM

URL www.fvwm.org
Licence GPL
Latest release 2.6.5
It's almost as old as the Linux kernel, and it's insanely configurable.

IceWM

URL www.icewm.org
Licence GPL + LGPL
Latest release 1.3.8
Like the traditional Windows 9x layout? This dinky WM will float your boat.

Window Maker

URL www.windowmaker.org
Licence GPL
Latest Release 0.95.7
This takes a more novel approach and apes the design of Next/OpenSTEP.

i3

URL www.i3wm.org
Licence BSD
Latest release 4.11
A tiling window manager designed to maximise screen real estate usage.

Fluxbox

URL www.fluxbox.org
Licence MIT
Latest release 1.3.7
Very small, very fast, and – with the right themes – very good-looking too.

Awesome

URL http://awesome.naquadah.org
Licence GPL
Latest release 3.5.6
For maximum productivity, you want to keep your hands on the keyboard.

Window managers

Changing just one component in your Linux installation can have a massive impact on your productivity. Just think about how much time you spend managing windows: moving them around, maximising and minimising them, and placing them side-by-side to work on two tasks at the same time. Perhaps you also use virtual desktops, keyboard shortcuts and other features for managing your activities. Dealing with these things may only seem like a small part of your daily work, but it all adds up over the months and years.

Even if you've gotten used to your regular desktop environment – be it Gnome, KDE, Xfce, Mate, Cinnamon or something else – you could be working much smarter and faster. We have nothing against those desktops, but they have their downsides. They're generally heavy on the RAM banks, they often tend to limit customisation, and they lack some power-users' features that can save a huge amount of time in the long run.

We're going to look at six of the most useful alternative lightweight window managers (WMs). Compared to the likes of Gnome and KDE, they provide relatively few features – just the ability to manage windows and start programs. You can add the other features of a desktop environment (such as a file manager) via your distro's package manager, with the end result being something that uses much less RAM, runs at a blistering pace, and offers customisation and features way beyond the big-name desktops.

The six WMs we're testing here all offer their own unique sets of features and are available in pretty much every distro, so you can try them today. We'll help you to explore them, point out their benefits, and show you some tricks to get the most out of them.

The window managers we're testing here all offer speed, features, and the potential for customisation.

The agony of choice

There are hundreds of window managers out there on the internet, most of which aren't in active development any more or were simply forks of other window managers with a few tweaks made. The Arch Linux wiki is a great resource for information on other WMs, so take a look at **https://wiki.archlinux.org/index.php/Window_manager** to explore further.

And if you don't find a WM to scratch your itch, and fancy getting started with a new project, why not write your own? There's a lot to learn, but it's not as difficult as you might think. Getting the basics done requires just a small amount of code, and Chuan Ji has written an excellent introduction with concepts and code at **http://tinyurl.com/njlkhkp**.

Layers upon layers

The modularity of the graphical stack on Linux/Unix can be confusing.

For new Linux users who've come over from Windows or Mac OS X, the different layers that make up the end-user environment in Linux can be bamboozling. But it's this modularity that makes our operating system so flexible. At the lowest level we have the X Window System, which talks to graphics hardware and renders pixels to the screen. It provides a rudimentary API (application programming interface) so that programs can say "draw a line" or "turn these pixels a different colour".

Very few programs interact directly with X, however. Most software uses a graphical toolkit that does the hard work of talking to the X software (the X server), such as *Qt* or *GTK*. These toolkits provide a layer of abstraction and make porting software to other platforms, such as Windows, much easier. The next layer is the window manager, which talks to the X server and provides mechanisms for moving windows around, resizing windows and closing them. It's possible to run programs without a window manager, but they'll be fixed in size and position, making it impossible to work with multiple apps at once.

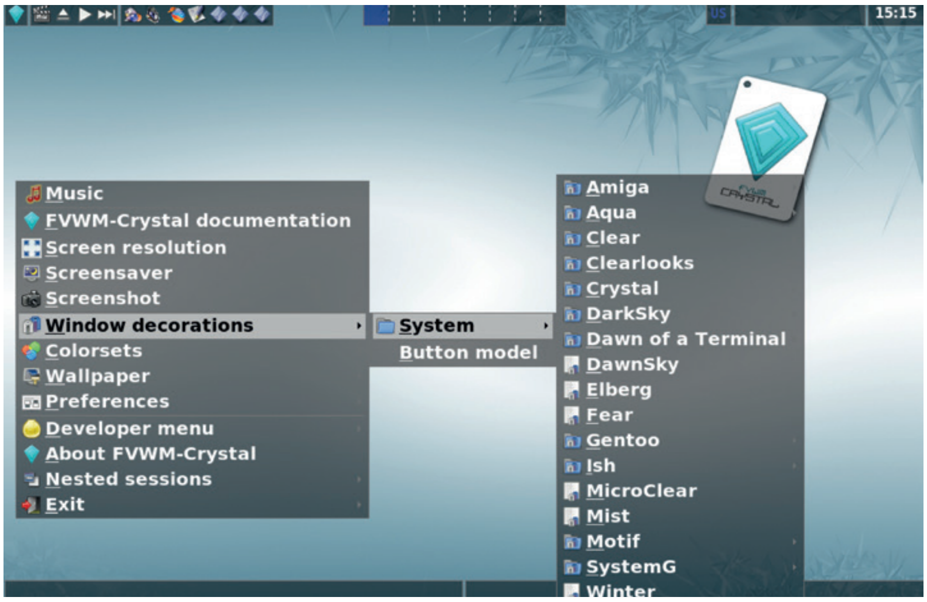
FVWM

As old as the hills.

Back in the early 90s, as the Linux kernel was paired with GNU software and we had a fully Free Software OS, FVWM was the window manager of choice. Actually, there wasn't a huge amount of choice back then, with most people running this or its predecessor, TWM. GNU/Linux was very much the domain of übergeeks back then, so FVWM didn't have much in the way of fancy wizards or step-by-step setup tools – no, you were expected to spend a few days working through a configuration file, exploring hundreds of options to craft your desktop to perfection.

To try it out, install it from your package manager, log out of your current session, and choose FVWM from the list of desktops or window managers when logging back in. (Unfortunately the location of this list varies with all the login screen setups out there, so you'll just have to click around until you find it. Time for some standards, we think!)

Once you're logged in you'll see a grand total of nothing. Left-click on the desktop, however, and a small menu will pop up; click on Setup Form to bring up a dialog box that can be used to create a more sane configuration. Select 'Create a starting **.fvwm2rc**' file and the buttons for FvwmWinList, FvwmButtons, FvwmPager, FvwmIconBox and FvwmTaskBar, then click the F2 button at the bottom followed by F3. Now you'll have a more usable setup with various components including a virtual desktop manager and window list at the bottom. Also, when you now left-click on a blank part of the desktop, you'll see a more detailed menu from which you can launch software.



For a prettier and more user-friendly introduction to FVWM, try its shiny FVWM-Crystal spin-off.

As mentioned, FVWM is insanely configurable – just look at the manual page (**man fvwm**). It's a mind-boggling 58,000 words long. If you look at your configuration file in **~/.fvwm/.fvwm2rc** you'll find plenty of options to play around with, so you can edit this file in a text editor, save and restart to view your changes. To get some inspiration for what's possible, try doing a web search for "fvwmrc" – many people have uploaded their configurations (with plenty of comments) so you can nab ideas and create a setup perfect for your liking.

TOO MUCH CHOICE!

It's quite tough to make FVWM look pretty, but one spin-off called FVWM-Crystal (**http://fvwm-crystal.sf.net**) does a decent job here, and is available in a separate package in many distros. This is then presented as another option in the login screen, and provides more sensible defaults than the vanilla FVWM package. It works in very much the same way – left-click on the desktop to bring up a menu – but with attractive program launchers, workspace switchers and taskbars out of the box.

FVWM is a fascinating project rich with history, but we can only recommend it if you're willing to put in the hours poring through the manual page and configuration file. If you've never been satisfied with any other window manager or desktop, maybe you can finally create your dream environment with FVWM. Plus, of course, it's a very mature project, so once you've built up your ideal **.fvwm2rc** file, you know it will work long into the future.

VERDICT
Has virtually every customisation option you can imagine, but requires a lot of patience to set up.
★★⯪☆☆

IceWM

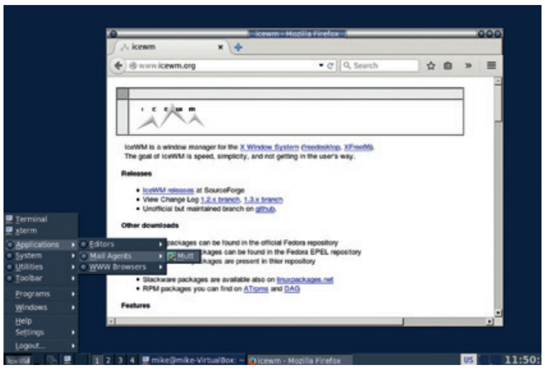
Harking back to the glory days of Windows 9x.

Well, when we say glory days we mean that in jest of course – Windows 95 was a big improvement over 3.11, but it was still a horribly unstable stack of software trying desperately to cover up its DOS roots. Nonetheless, for all its faults (and we were clinging on to our Amigas back when it came out), its interface had a certain amount of charm. The Start menu and taskbar combo worked well enough that it has been copied endlessly over the years.

IceWM is one of the older window managers here, and has been in development since the late 90s. Consequently it has a rather retro appearance, although a handful of fresher themes are included. Once you've installed it and logged in, you'll see a familiar taskbar and Start menu panel along the bottom. It also contains a clock and system tray on the right, along with workspace switcher buttons.

Click the IceWM button in the bottom-left to open a program menu – and note especially the Settings submenu where you can change the theme. But here you'll encounter a problem: IceWM isn't very clever at picking up what software is installed on the system. So in the program menus you'll see launchers for programs that you probably don't have installed.

To fix this, you'll need to run a separate utility and feed the results to IceWM's configuration file. One such option can be found at **https://github.com/gapan/xdgmenumaker**. It's annoying that this extra step is required, but on the upside IceWM's system requirements and memory usage are tiny in comparison to desktop environments. It's fast, and it's ridiculously stable – this author used it for five years without any crashes.



IceWM is a window manager for the X Window System (freedesktop, XFree86).
The goal of IceWM is speed, simplicity, and not getting in the user's way.

- IceWM releases at SourceForge
- View Change Log 1.2.x branch, 1.3.x branch
- Unofficial but maintained branch on github.

packages can be found in the official Fedora repository
kages can be found in the Fedora EPEL repository
kages are present in their repository

- Slackware packages are available also on linuxpackages.net
- RPM packages you can find on ATrpms and DAG

IceWM is supplied with themes emulating Windows and OS/2, along with some unique ones, such as Infadel2.

> You'll see a familiar taskbar and Start menu panel along the bottom

VERDICT
Screamingly fast but requires some fiddling to get the menus set up.
★★★☆☆

Fluxbox

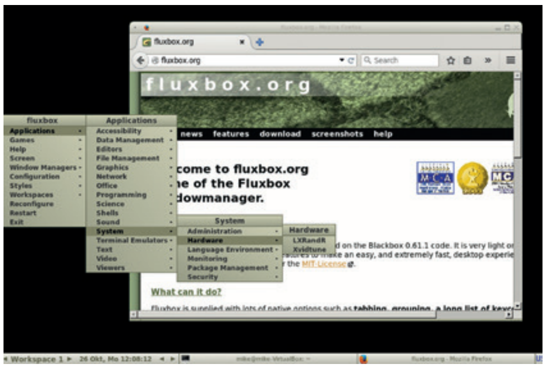
Dark and moody, this WM does its best to get out of your way.

Fluxbox is a fork of an older window manager called Blackbox, and dates back to 2001. It was hugely popular for a while among power users, before tiling window managers really reached their prime, although it's still in development and has an army of hardcore fans. Fluxbox boasts extremely low requirements and RAM usage thanks to its minimal dependencies – it uses the X Window System's own libraries for rendering and therefore doesn't need external toolkits hogging up RAM.

When you first start it, you'll notice a thin panel along the bottom with various elements contained therein. From left to right these are: a virtual desktop (workspace) switcher, taskbar and system tray area. Buttons are provided to switch between desktops and windows, but you can do the former with Ctrl+Alt and the left/right arrow keys, and the latter with good old Alt+Tab. Fluxbox is very minimal and low on effects – so there are no pretty transition effects when you switch between desktops.

Right-click on the desktop to bring up the main menu. Unlike with IceWM, this is automatically populated with software on the system, and it also provides a great deal of configurability without the need to poke around inside text files. Try searching in particular inside the Styles and Configuration submenus. If you look at the Window Manager submenu, you'll see that you can switch to another WM from Fluxbox without having to return to the login screen – a handy bonus if you're trying various WMs.

Fluxbox has a few extra features such as tabbed grouping of windows (drag titlebars with the middle mouse button together to join them), but by and large it's a simple, sleek and attractive window manager for those who need the basics but with no extra fluff. As it uses little screen space, it's ideal for reviving an old netbook with a fresher distro.



come to fluxbox.org
ne of the Fluxbox
dowmanager.

d on the Blackbox 0.61.1 code. It is very light o
e an easy, and extremely fast, desktop experie
the MIT License.

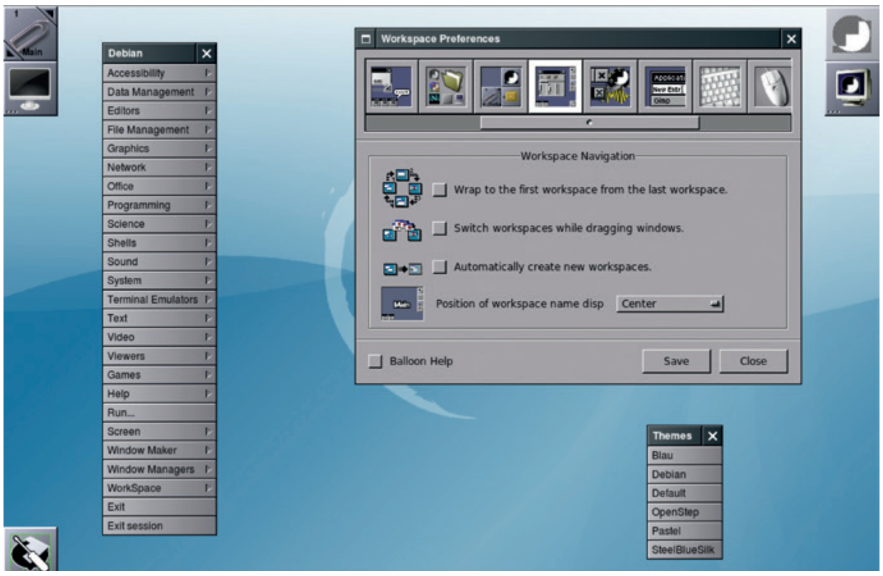
What can it do?

Fluxbox has a lot in common with IceWM, but with a more imaginative design and working menus out of the box.

VERDICT
A joy to use if you want to save RAM and don't need tiling facilities.
★★★☆☆

Window Maker

NeXT/OpenSTEP comes back to life.



Wrap to the first workspace from the last workspace.
Switch workspaces while dragging windows.
Automatically create new workspaces.
Position of workspace name disp: Center

Note that you can tear off menus and keep them separately on the screen by clicking their top bars, as we've done with Themes here.

Steve Jobs isn't the most popular figure in the Free Software world, but there's no denying that he had a massive impact on computing history. After being kicked out of Apple – the company he co-founded – back in 1985, Jobs created Next, which developed a high-end (and extremely expensive at $10,000) computer. This machine ran the NextStep operating system, which later morphed into OpenStep and became part of Mac OS X when Jobs returned to Apple in 1997.

Window Maker is an open source window manager that recreates much of the classic look and feel of NextStep, while still providing the ability to run modern FOSS programs. Unlike the other window managers on test here, Window Maker is a slightly larger project incorporating a widget set and extra libraries.

To get it running, install it via your distro's package manager, log out of your current session and choose it from the list of desktops at the login screen. The first thing you'll see is a pair of icons in the top-right corner – this is the Dock, and it contains program launchers. You can start a program by double-clicking its launcher, so try it with the default Terminal icon. Next, right-click on an empty part of the desktop to bring up the applications menu, which provides access to the software installed in your distro.

Try launching a program and you'll notice that its icon appears in the bottom-left corner of your screen. If you drag this icon onto the Dock – ie next to the Terminal icon – you'll see that you can add it to the Dock permanently. In this way you can quickly build up a set of launchers for your most commonly-used tools.

Add usefulness

The button in the top-left is the Clip. This lets you switch between workspaces (virtual desktops), of which by default there is only one, so right-click on the desktop and go to Workspace > Workspaces > Add New To Add More. You can add launcher icons to the Clip for specific workspaces, so you can use the Dock for general apps and the Clip for apps you only want to launch on certain workspaces.

Window Maker includes a comprehensive setup tool (double-click the uppermost icon in the Dock) and various themes (See WorkSpace > Appearance > Styles in the desktop menu). It's a mature, reliable and attractive desktop with a unique way of working, and we've spent many months hapily using it as our daily driver in the past.

VERDICT
Well tested, refined and a refreshing alternative to the typical approach.
★★★★☆

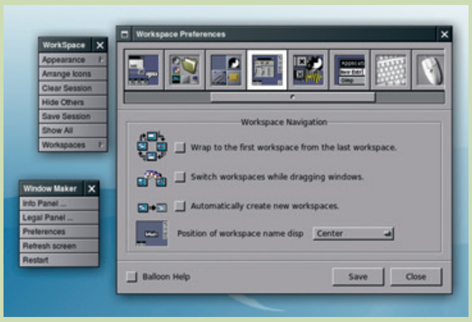
Beefing up your WM

Useful apps and tools to add.

Once you've settled down with your new choice of window manager, you may find your setup a bit lacking in comparison to a desktop environment. After all, the likes of Gnome and KDE provide integrated file managers, text editors and other tools – whereas your window manager serves primarily as a launcher for anything you happen to have installed. So you'll first want to install a good file manager, and for this purpose we can recommend *PCManFM* (**http://wiki.lxde.org/en/PCManFM**). It's used in the lightweight LXDE desktop, and does a great all-round job. Another alternative is *XFE* (**http://roland65.free.fr/xfe**), based on the *FOX* toolkit, which has fewer features but runs at light speed.

Another tool worth adding – especially if you care about eye-candy – is a compositing manager. This lets you add effects like drop-shadows and animations to windows, and the one we recommend is *Compton* (**https://github.com/chjj/compton**). It's available in almost every major distro, and once you have it installed, run it from a terminal with **compton -c** to get pretty shadows underneath windows and menus. See the manual page (**man compton**) to get an overview of all the available options.

For some WMs you may want to add a dock for your most commonly used applications. *Cairo Dock* (**http://glx-dock.org**) is a superb choice here, and while it's clearly heavily inspired by Mac OS X, it has plenty of features in its own right. To learn more about using these extra tools, and piecing them all together to create your own desktop environment, see our tutorial at **www.linuxvoice.com/create-your-own-desktop-environment**.



Wrap to the first workspace from the last workspace.
Switch workspaces while dragging windows.
Automatically create new workspaces.
Position of workspace name disp: Center

Here's Window Maker again, but with lurvely drop shadows around windows and menus thanks to the *Compton* compositor.

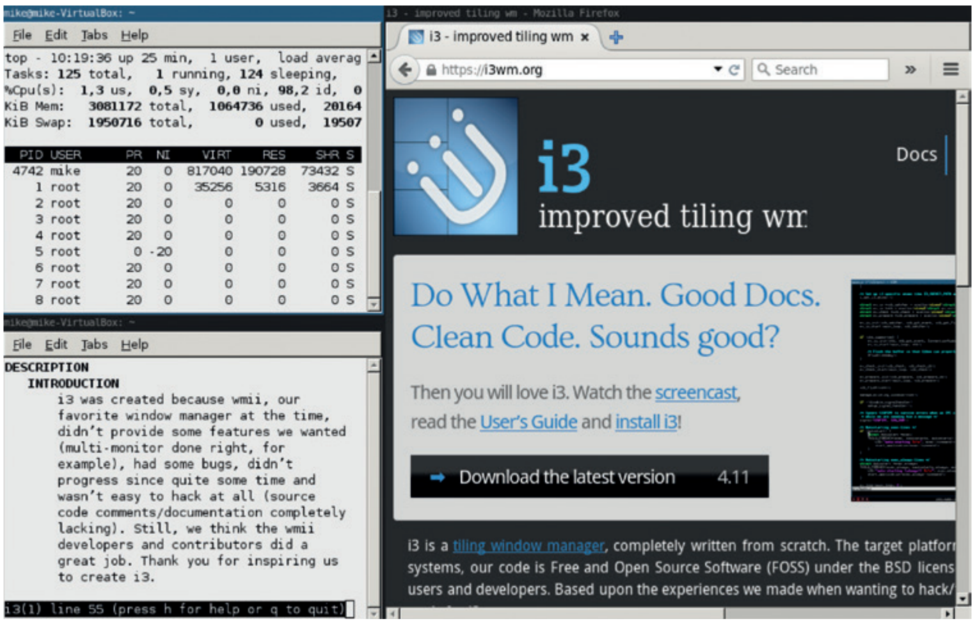
i3 vs Awesome

Tiling WMs go head-to-head.

Tiling WMs are becoming increasingly popular, especially among power users running large displays, and they can help you to work much more efficiently. But what is a tiling WM?

The best way to explain is by demonstrating. Install i3 from your distro's package manager and then select it at your login screen. You'll see two things when the WM starts up: a thin panel along the bottom of the screen (called the i3bar) and a "first configuration" box asking you if you want an automatically generated config file – hit Enter here and choose Alt as the default modifier when prompted.

And you're ready to go. Alt+Enter and a terminal window will appear, filling the whole screen. Now hit the same key combination again to spawn another terminal – and you'll see that they've automatically been placed side-by-side (or one above the other, depending on your display ratio).



```
top - 10:19:36 up 25 min,  1 user,  load averag
Tasks: 125 total,   1 running, 124 sleeping,
%Cpu(s):  1,3 us,  0,5 sy,  0,0 ni, 98,2 id,  0
KiB Mem:   3081172 total,  1064736 used,  20164
KiB Swap:  1950716 total,        0 used,  19507

  PID USER      PR  NI    VIRT    RES    SHR S
 4742 mike      20   0  817040 190728  73432 S
    1 root      20   0   35256   5316   3664 S
    2 root      20   0       0      0      0 S
    3 root      20   0       0      0      0 S
    4 root      20   0       0      0      0 S
    5 root       0 -20       0      0      0 S
    6 root      20   0       0      0      0 S
    7 root      20   0       0      0      0 S
    8 root      20   0       0      0      0 S
```

```
DESCRIPTION
   INTRODUCTION
       i3 was created because wmii, our
       favorite window manager at the time,
       didn't provide some features we wanted
       (multi-monitor done right, for
       example), had some bugs, didn't
       progress since quite some time and
       wasn't easy to hack at all (source
       code comments/documentation completely
       lacking). Still, we think the wmii
       developers and contributors did a
       great job. Thank you for inspiring us
       to create i3.

i3(1) line 55 (press h for help or q to quit)
```

Do What I Mean. Good Docs. Clean Code. Sounds good?

Then you will love i3. Watch the screencast, read the User's Guide and install i3!

i3 is a tiling window manager, completely written from scratch. The target platfor
systems, our code is Free and Open Source Software (FOSS) under the BSD licens
users and developers. Based upon the experiences we made when wanting to hack/

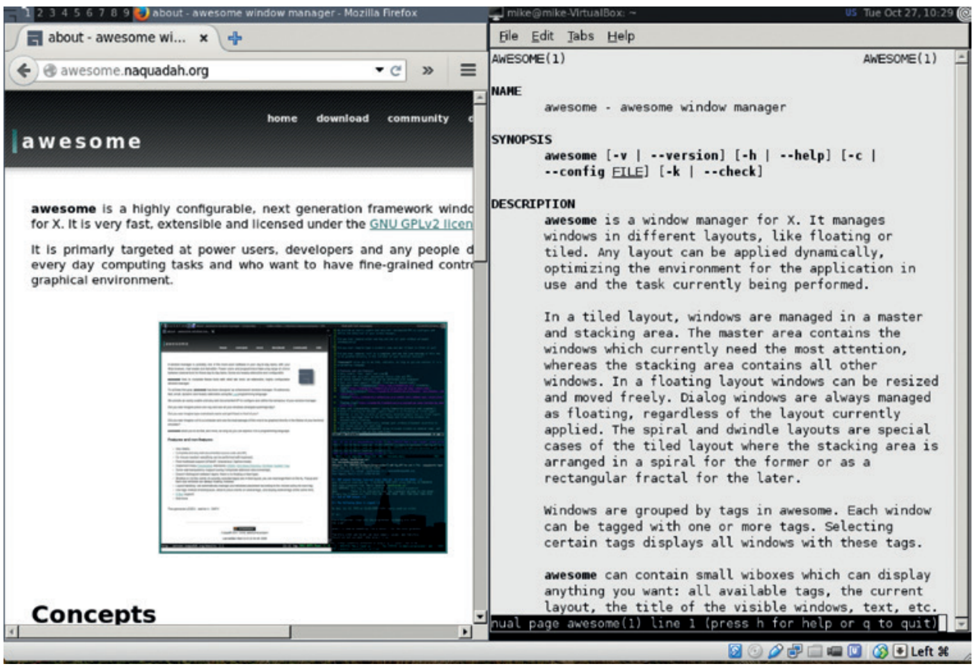
A typical i3 session, with vertical and horizontal splits in use. To exit the WM use Alt+Shift+E.

Save space with tiling

This is the tiling aspect of i3; it automatically places and resizes windows to make the best use of your screen space. To switch between windows, use Alt with the J, K, L and ; keys for left, down, up and right respectively (a bit like in the *Vi* editor). To switch between horizontal and vertical splitting, use Alt+E. And to close a window, use Alt+Shift+Q. To resize a window, hit Alt+R and then use the arrow keys followed by Enter when you're done – or use the mouse to grab the handle between windows if you don't want to say goodbye to the rodent completely. i3 is capable of much more, so see **https://i3wm.org/docs/userguide.html** for the complete guide.

Awesome, meanwhile, shares many of the same features as i3: it's a tiling window manager designed to maximise screen space usage and make you less dependent on the mouse. Unlike i3, however, it's a bit more friendly to mouse users out of the box, as you'll see if you right-click on the desktop (a program menu appears). To open a terminal, use Mod4 (usually the Windows key) and Enter – by default windows are in floating mode, so use Mod4+Space to switch to tiled mode, like i3. A full list of keybindings can be found at **http://awesome.naquadah.org/doc/manpages/awesome.1.html**.

Awesome is a hugely configurable WM with support for Lua extensions to add tabs, popup menus and even a *Space Invaders* game. Once you've spent a few days learning the keybindings for Awesome or i3, you'll never want to waste your life shoving windows around on your desktop ever again.



awesome is a highly configurable, next generation framework windo
for X. It is very fast, extensible and licensed under the GNU GPLv2 licen

It is primarly targeted at power users, developers and any people d
every day computing tasks and who want to have fine-grained contr
graphical environment.

Concepts

```
AWESOME(1)                                      AWESOME(1)

NAME
       awesome - awesome window manager

SYNOPSIS
       awesome [-v | --version] [-h | --help] [-c |
       --config FILE] [-k | --check]

DESCRIPTION
       awesome is a window manager for X. It manages
       windows in different layouts, like floating or
       tiled. Any layout can be applied dynamically,
       optimizing the environment for the application in
       use and the task currently being performed.

       In a tiled layout, windows are managed in a master
       and stacking area. The master area contains the
       windows which currently need the most attention,
       whereas the stacking area contains all other
       windows. In a floating layout windows can be resized
       and moved freely. Dialog windows are always managed
       as floating, regardless of the layout currently
       applied. The spiral and dwindle layouts are special
       cases of the tiled layout where the stacking area is
       arranged in a spiral for the former or as a
       rectangular fractal for the later.

       Windows are grouped by tags in awesome. Each window
       can be tagged with one or more tags. Selecting
       certain tags displays all windows with these tags.

       awesome can contain small wiboxes which can display
       anything you want: all available tags, the current
       layout, the title of the visible windows, text, etc.
nual page awesome(1) line 1 (press h for help or q to quit)
```

Awesome has a lot in common with i3, and is also extensible thanks to Lua scripting support.

VERDICT

I3 Simple, clean and effective – the perfect introduction to tiling window managers.
★★★★☆

AWESOME More complex than i3 but more versatile thanks to Lua extensions.
★★★½☆

OUR VERDICT

Window managers

Everyone has their own way of working, so we can't say which one of the six window managers here is perfect for you – but hopefully you've found a few that have whetted your appetite to try out for a couple of days or weeks. Just a few small changes in your working habits can have a huge effect on your productivity as time goes by.

From our perspective, i3 and Window Maker come out at the top of the bunch. i3 just makes so much sense for the type of work that many of us Linux geeks do, and while it takes a while to master, the learning curve is worth it. If you have plenty of screen space and want to neatly divide your display into sections so that you can work on multiple projects side-by-side – or just keep tabs on an **htop** session on a remote machine – then i3 is bliss. You'll wonder why you ever wasted so much time manually shuffling windows around with the mouse.

Of course, constant keyboard usage isn't for everyone, and if you have a good relationship with your rodent (or trackpad), Window Maker is well worth getting to grips with. Its use of the Dock and Clip creates an alternative workflow to the usual taskbar-and-system-tray setup, and the ability to dock menus around the screen can be highly useful as well. Plus, some of the in-built styles look great – they can be rather dark and stony, but look far better than the extreme flatness that's being adopted elsewhere these days.

So while we recommend that everyone gives i3 and Window Maker a go, there's still plenty worth investigating in the others. We'd choose IceWM if we were upgrading someone's old Windows XP machine, in that it provides a familiar layout and runs like the clappers even on dated hardware. Fluxbox's conservative use of screen real estate makes it ideal for old netbooks, while Awesome has plenty to sing about as well.

And if you want to create the WM of your dreams without hacking away on code, just spend a few months meticulously crafting an FVWM configuration file and live forever in peace.

i3 makes so much sense for geeks – while it takes a while to master, the learning curve is worth it.

Tmux: a window manager for your terminal

Yes, WMs even exist for command line programs. Tmux (**https://tmux.github.io**) is the best example, and is included by default in many major distros. Simply enter **tmux** to start it, and you'll see a green bar along the bottom. Hit Ctrl+B followed by C to create a new (full screen) window, and Ctrl+B followed by N or P to switch between windows. In the panel at the bottom, you'll see the names of programs running in each window.

Tmux provides a tiling option so you can have multiple programs running in the same terminal window next to each other. Even better, Tmux lets you detach from sessions to reconnect to them later. If you're SSHed into a system and running some programs inside Tmux, hit Ctrl+B followed by D to detach and return to the *Bash* prompt. You can now close the terminal window – the programs running on the remote server will continue. SSH back in to the server and run **tmux a** to reconnect, and everything will show up as it was when you detached.

Everyone should try a tiling window manager at least once in their lives, we reckon – you have nothing to fear but fear itself!

1st i3

Killer feature: Tiling heaven
www.i3wm.org
If you have a big monitor, you absolutely must try this – don't be put off by the learning curve.

2nd Window Maker

Killer feature: NextStep goodness
www.windowmaker.org
A great alternative to taskbar-based window managers, with some lush themes included.

3rd Awesome

Killer feature: Lua extensions
http://awesome.naquadah.org
More tiling fun, with the ability to customise and add heaps of extra functionality via extensions.

4th IceWM

Killer feature: Familiarity for Windows 9x users
www.icewm.org
Blazingly fast and providing a comfortable environment for those used to old-style Windows releases.

5th Fluxbox

Killer feature: Perfect minimalism
www.fluxbox.org
Keeps out of your way but still provides just enough to make you feel at home.

6th FVWM

Killer feature: Taking up your life
www.fvwm.org
It provides pretty much every customisation option you could imagine – if you're willing to read the giant man page.

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 LINUXVOICE

ON SALE THURSDAY 17 DECEMBER

THE INTERNET OF THINGS

Frankly, we were unimpressed with the Internet of Things buzzphrase – until we realised that we could hack everything!

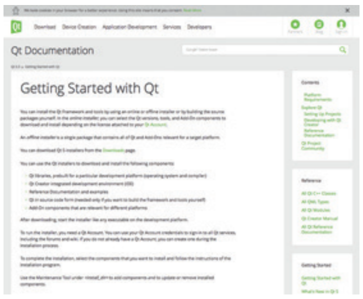
EVEN MORE AWESOME!

Bradley Kuhn

There are forces out there that want to take control of your computer away from you. Don't worry though: Bradley Kuhn has got your back. Cheers Bradley!

Automate RSS

The old technologies from the dawn of Web 2.0 are not dead; they are only resting. Reanimate RSS, and get your websites to talk to each other.



Getting Started with Qt

You can install the Qt Framework and tools by using an online or offline installer or by building the source packages yourself. In the online installer, you can select the Qt versions, tools, and Add-On components to download and install depending on the license attached to your Qt Account.

An offline installer is a single package that contains all of Qt and Add-Ons relevant for a target platform.

You can download Qt 5 installers from the Downloads page.

You can use the Qt installers to download and install the following components:

- Qt libraries, prebuilt for a particular development platform (operating system and compiler)
- Qt Creator integrated development environment (IDE)
- Reference Documentation and examples
- Qt in source code form (needed only if you want to build the framework and tools yourself)
- Add-On components that are relevant for different platforms

After downloading, start the installer like any executable on the development platform.

To run the installer, you need a Qt Account. You can use your Qt Account credentials to sign in to all Qt services, including the forums and wiki. If you do not already have a Qt Account, you can create one during the installation process.

To complete the installation, select the components that you want to install and follow the instructions of the installation program.

Use the Maintenance Tool under <install_dir> to add components and to update or remove installed components.

Documentation

The good, the bad and the ugly of Linux documentation – and what you can do to make Free Software better and more accessible for everyone.

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

Copyright Linux is a trademark of Linus Torvalds, and is used with permission. Anything in this magazine may not be reproduced without permission of the editor, until July 2016 when all content (including our images) is re-licensed CC-BY-SA.
©Linux Voice Ltd 2015
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

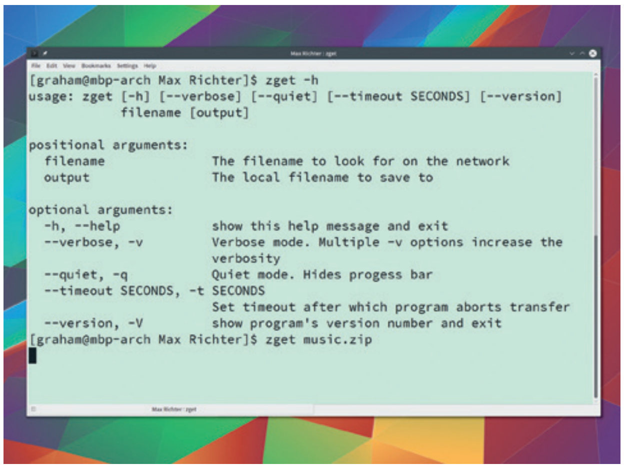
Out benevolent editorial overlord **Graham Morrison** tears himself away from updating Arch Linux to search for the best new free software.

Simple file transfer

Zget

How many times have you been sitting with your laptop and needed to quickly and instantly transfer a file to another computer? If you're anything like us, you've lost count. The answer is typically to find a USB thumbdrive, or use web-based email, or even a quick installation of SSH so that SFTP works. But thanks to a tip from @nlswrnr, we've got a solution which we've found to be far simpler to use and particularly well suited for when someone else is using your computer. **Zget** uses the magic of zero-configuration networking to automatically negotiate a connection across your network, so you don't need to worry about IP addresses or how your clients connect with one another. After installing the tool through *Pip*, the Python package manager, and a bit of configuration, transferring files is as simple as typing **zget filename** on one machine, and **zget filename** on the other. The two clients should find one another and instigate the transfer without any further user interference.

PROJECT WEBSITE
https://github.com/nils-werner/zget



```
[graham@mbp-arch Max Richter]$ zget -h
usage: zget [-h] [--verbose] [--quiet] [--timeout SECONDS] [--version]
            filename [output]

positional arguments:
  filename              The filename to look for on the network
  output                The local filename to save to

optional arguments:
  -h, --help            show this help message and exit
  --verbose, -v         Verbose mode. Multiple -v options increase the
                        verbosity
  --quiet, -q           Quiet mode. Hides progess bar
  --timeout SECONDS, -t SECONDS
                        Set timeout after which program aborts transfer
  --version, -V         show program's version number and exit
[graham@mbp-arch Max Richter]$ zget music.zip
```

Use **Zget** to quickly transfer files from one machine on a network to another.

Cloud music player

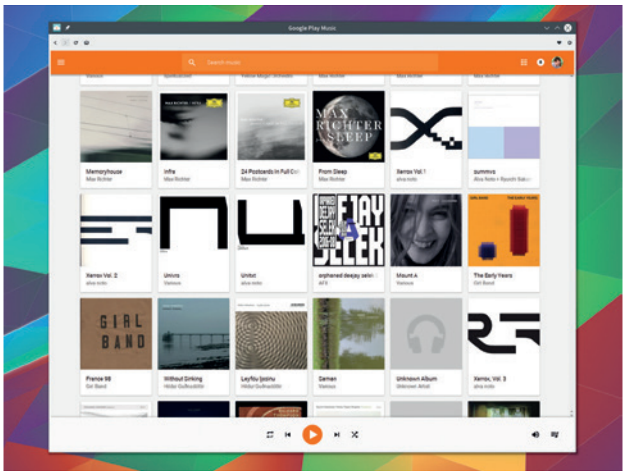
Nuvola Player 3 (beta)

Please forgive us. We've become smitten by online music services, mostly because they're so convenient and using them means you don't have to worry about taking your files with you, or synchronising your music before making a trip. *Nuvola Player* specialises in being a single portal to several of these cloud-based services, encapsulating their web interfaces into a single window on your desktop. We've used the old 2.x version before, especially on Ubuntu, and there's a major new update on the horizon, with the betas of version 3 now considered relatively stable.

You'll need the player and additional plugins for whichever providers you want to use. *Nuvola* currently supports 15 different streaming services, including Spotify, Google Play Music, Amazon, Deezer, Rdio, Plex, Tunin and even Logitech's Media Server, which could be useful for accessing your own local collection.

Simple sounds

With the plugins installed and the app running, you select one of the services to get started. The experience is exactly like using a web browser, because that's essentially what's running within the application window. Desktop integration includes notifications, Scrobbling, and lyrics. We'd love to see some of *Banshee*'s collection cross-referencing, where a playlist can be created from different sources, but *Nuvola* keeps things simple. And there's a lot to like about that.



You can upload 50,000 of your own music files to Google Play for free, and access them from anywhere.

PROJECT WEBSITE
https://tiliado.eu/nuvolaplayer

Terminal emulator

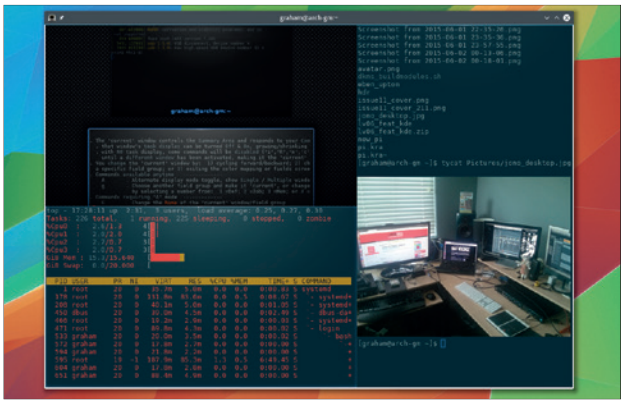
Terminology 0.9

The multiple promises of future computing interfaces, as seen in films like *Minority Report*, or perhaps the positronic brains of Asimov's robots, seem to make an anachronism of the manual command line. And yet it's 2015, and many of us use it daily.

But the command interpreter is only one part of the interface, with the other part being the host for the terminal session. Unless you still access *Bash* through a late 1970s video terminal, these host applications are called terminal emulators, mostly because they emulate the function of those late 1970s video terminals.These all provide a similar feature set, and integrate well with your chosen environments. But one thing they don't do is emulate the physical characteristics of those early terminals. But experiencing a few of those physical attributes is quite good fun, especially when they don't impede on function, and this is where *Terminology* comes in.

Terminology is a smooth, feature-packed terminal emulator that takes the emulator part seriously. By default, its simple *X-Term* startup configuration glows from the strip lighting of your virtual 1981-era computing laboratory. Right-click, and smooth scrolling menus let you change everything about its appearance. Behind the scenes it can work with the latest technology, such as OpenGL or Wayland, and some of the oldest, like the framebuffer, and it can act on files, URLs, email addresses and music just like a desktop would. There's even a visual virtual 'session' mode. But the best thing about *Terminology* is that it remains quick and responsive, turning what could have been a rudimentary *Bash* session into a modern terminal session with some neat references to the past.

Terminology is a smooth, feature-packed terminal emulator for Linux


Terminology is part of Enlightenment, but works brilliantly on any desktop.

PROJECT WEBSITE
https://www.enlightenment.org/about-terminology

Development environment

KDevelop 4.7.2

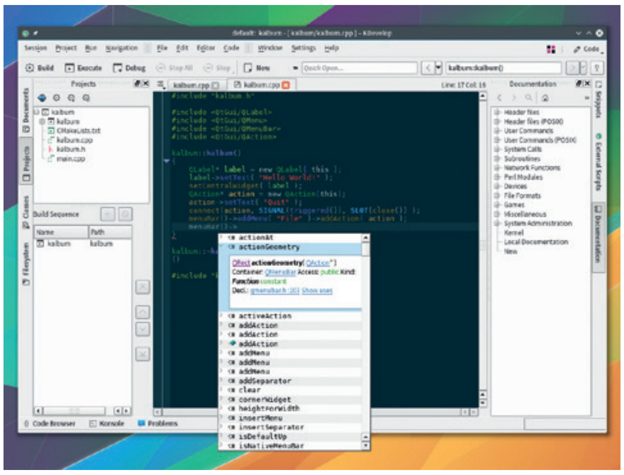
KDE has just celebrated its 19th birthday and for many of those years, it was its integrated development environment, *KDevelop*, that made KDE as accessible as possible to new developers. Like KDE itself, *KDevelop* lost its way when a major update was quickly followed by a long-promised code overhaul that eventually led to a complete rewrite.

At the same time, the *Qt* project – used heavily by all KDE developers – released its own much more functional development environment, the wonderful *Qt Creator*. *Qt Creator* benefited from a full-time team and the official backing of the *Qt* project, quickly making it slick, stable and just as capable as the older *KDevelop*.

Qt Creator is still a great open source development environment, but *Qt*'s recent emphasis on touch, mobile and proprietary modules has meant *KDevelop* is just as important as ever. Thankfully, with over five years since the first re-written 4.0 release, *KDevelop* has come successfully through its dark period, with this latest release being another strong revision to the major 4.7.0 update that appeared in 2014.

KDE made easy(er)

Help integration, code completion, huge performance improvements, PHP support and a powerful plugin system make it a must-try for KDE developers, and even other programmers looking for a modern IDE. There's Gnome and wxWidgets support, for example, along with support for languages like Python, Perl, Ada, Java and Ruby alongside the perennial C++. We like the way you can have multiple sessions open, which is useful if you're using one project to learn from while coding into another, but the best thing about *KDevelop* is that it's remained relatively lightweight and straightforward, unlike its previous incarnation.


KDevelop is just like a version of KDE's excellent *Kate* text editor augmented for developers, which is exactly what an IDE should be.

PROJECT WEBSITE
https://www.kdevelop.org

Interior design

Sweet Home 3D 5.1

Hell has indeed frozen over. Not only has Microsoft created its own Linux distribution, but we're now covering interior design software in the pages of a Linux and open source magazine! But fear not. Thanks to open source, we're already ahead of the game. We can reveal, for instance, that we've reached peak Château Grey and French Linen for colouring, and that next season's hues will be more vibrant, with yellow-based neutrals and velvet flocked paper coming back into style. We know this because we've experimented with our own palettes, textures, layouts and furniture, transforming our bedroom offices without spilling a drop of paint. And we've done this thanks to *Sweet Home 3D*.

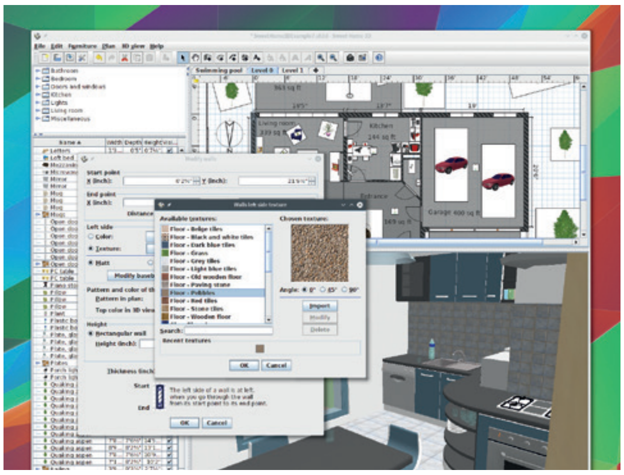
Sweet Home panorama

Despite initially sounding like an add-on for the popular life alternative, *The Sims*, *Sweet Home 3D* is a serious design tool that combines draughtsmanship, 3D modelling, texture mapping and rendering with a sense of homely belonging taken directly from Maslow's hierarchy of needs. You start by creating the floor plan, dragging the mouse to generate walls with satisfying angles. You can even add skirting boards and vary the thickness of your partitions, perfect for the slanted walls of a gym or swimming pool.

The background grid snapping and the distance/angle annotations make it easy to recreate a real environment – you'll just need to take a few measurements. You then drag and drop doors and windows into your scene, before moving on to the huge variety of furniture and fittings listed in the hierarchical list.

When you're happy with the layout, you get to play with colours and textures, and every surface can be modified and adapted according to your creative whimsy. We were a little disappointed that the palette selection tool didn't include the latest collection from Farrow & Ball, but it does include both the RAL and Creative Commons colour matching lists, and dialling in your isn't a problem. When you're finished, you can stroll around the real-time 3D view using WASD keys, or by dragging a symbol around the plan, and you can even render a 'photo' within the application, with a surprisingly photo-realistic quality (although it takes a long time), or output the entire scene as an object file for *Blender*. We found the entire process hugely enjoyable, and that's without any kind of motivation to tidy-up the place or get on makeover TV.

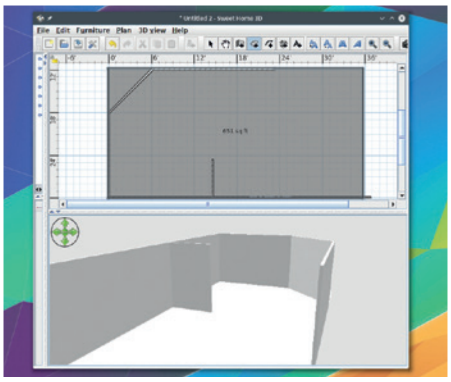
Sweet Home 3D combines 3D modelling, texture mapping and rendering



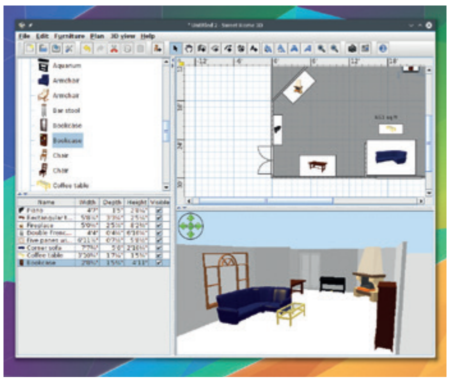
Bring a riot of colour to your garden room in complete and immersive 3D.

PROJECT WEBSITE
www.sweethome3d.com

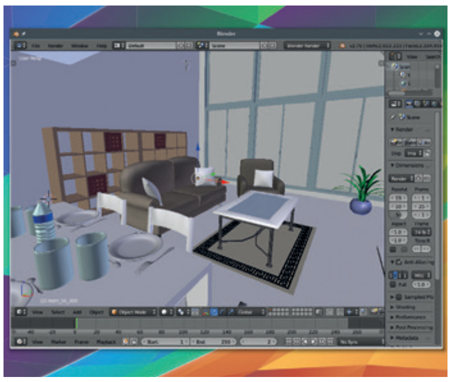
How it works: Build your perfect home



1 Use the 'Create walls' and 'Create rooms' modes to drag your layout into existence, using either your imagination or real-word coordinates from your own property.



2 Add the doors, windows, furniture and furnishings. Import external models if required. All colours, textures and materials can be changed according to taste.



3 The final output can be rendered within the application, or exported as an object file that can be processed or raytraced in a 3D application like *Blender*.

Minimal browser

Dillo 3.0.5

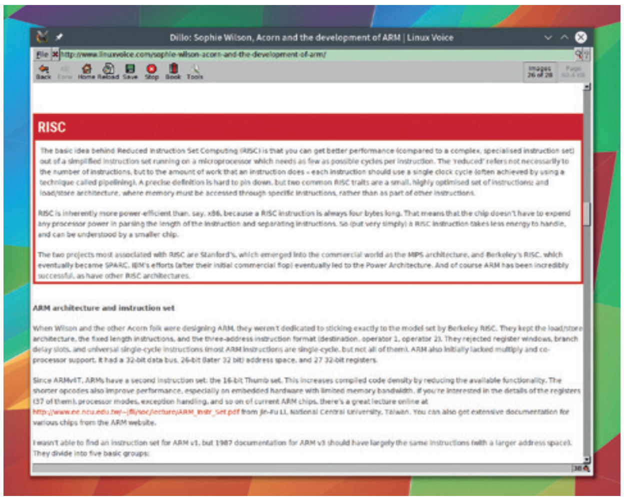
Although all of us on the team still love *Firefox*, and consider it one of the most important open source projects out there, there are murmurings of disillusioned after recent developments. In particular, *Firefox* is no longer a lightweight web browser, requiring significant amounts of RAM and CPU if you open more than a dozen tabs at once. This has left us eager to find an alternative, at least for the majority of browsing we do, which is searching for our own names and keeping abreast of /r/ToasterRights.

This is one of the reasons we found *Qutebrowser* so effective – the combination of low resource usage, good rendering and *Vim*-like shortcuts revolutionised our browsing behaviour. And it's also another reason we're taking a look at *Dillo*, which has recently been upgraded to version 3.

Dillo is lightning fast, which is the first thing you notice after launching its 725k binary. Its page loading speed takes you back to a time before the irony of responsive web design, before JavaScript and even before image maps.

But so too does its rendering, which is sparse, often appearing broken and the opposite of what many modern browsers would expect. However, as it should always be, the words are always legible and navigation remains clear. Like a Formula 1 racing car that's unsuitable for roads, *Dillo*'s compromises are all made for speed. We really enjoy using *Dillo*, and while it's slightly too minimal for day-to-day use, it's brilliant on devices like the Raspberry Pi, or on an older machine that rarely needs to render a web page.

> Dillo is lightning fast, which is the first thing you notice on launching its 725k binary


Dillo: Sophie Wilson, Acorn and the development of ARM | Linux Voice

http://www.linuxvoice.com/sophie-wilson-acorn-and-the-development-of-arm/

RISC

The basic idea behind Reduced Instruction Set Computing (RISC) is that you can get better performance (compared to a complex, specialised instruction set) out of a simplified instruction set running on a microprocessor which needs as few as possible cycles per instruction. The 'reduced' refers not necessarily to the number of instructions, but to the amount of work that an instruction does – each instruction should use a single clock cycle (often achieved by using a technique called pipelining). A precise definition is hard to pin down, but two common RISC traits are a small, highly optimised set of instructions and load/store architecture, where memory must be accessed through specific instructions, rather than as part of other instructions.

RISC is inherently more power-efficient than, say, x86, because a RISC instruction is always four bytes long. That means that the chip doesn't have to expend any processor power in parsing the length of the instruction and separating instructions. So (put very simply) a RISC instruction takes less energy to handle, and can be understood by a smaller chip.

The two projects most associated with RISC are Stanford's, which emerged into the commercial world as the MIPS architecture, and Berkeley's RISC, which eventually became SPARC. IBM's efforts (after their initial commercial flop) eventually led to the Power Architecture. And of course ARM has been incredibly successful, as have other RISC architectures.

ARM architecture and instruction set

When Wilson and the other Acorn folk were designing ARM, they weren't dedicated to sticking exactly to the model set by Berkeley RISC. They kept the load/store architecture, the fixed length instructions, and the three-address instruction format (destination, operator 1, operator 2). They rejected register windows, branch delay slots, and universal single-cycle instructions (most ARM instructions are single-cycle, but not all of them). ARM also initially lacked multiply and co-processor support. It had a 32-bit data bus, 26-bit (later 32 bit) address space, and 27 32-bit registers.

Since ARMv4T, ARMs have a second instruction set: the 16-bit Thumb set. This increases compiled code density by reducing the available functionality. The shorter opcodes also improve performance, especially on embedded hardware with limited memory bandwidth. If you're interested in the details of the registers (37 of them), processor modes, exception handling, and so on of current ARM chips, there's a great lecture online at http://www.ee.ncu.edu.tw/~jfli/soc/lecture/ARM_Instr_Set.pdf from Jin-Fu Li, National Central University, Taiwan. You can also get extensive documentation for various chips from the ARM website.

I wasn't able to find an instruction set for ARM v1, but 1987 documentation for ARM v3 should have largely the same instructions (with a larger address space). They divide into five basic groups:

Even *Dillo*'s cache is cleared when you quit the browser, helping both speed and your privacy.

PROJECT WEBSITE
www.dillo.org/screenshots/index.html

Secure messaging

Cutegram 2.7.0

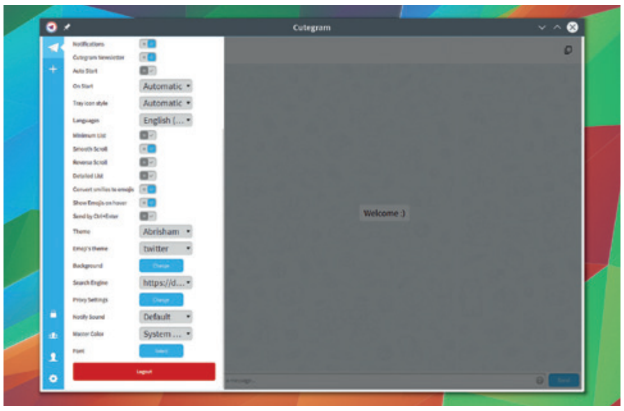
We've covered quite a few secure instant messaging platforms in the past, but *Cutegram* has become one of our favourites. As its name implies, this is a Telegram client built atop the *Qt* framework (pronounced 'cute' by developers).

Telegram is a well established platform for messaging securely and privately. There are official clients for nearly every device – including another *Qt*-based offering for Linux, and even a client that can be driven from the command line. They're all open source, but the server software that binds clients together is closed – however, many users trust Telegram enough to make it their default communications tool, and with a reported 60 million users sending 12 billion messages every day, it has become the go-to application for many. This is the other side of the Faustian pact in secure messaging. You need to use a client the other person is using too.

Always compromises

Ignoring the almost-impossible-to-certify security, Telegram is still a great platform, and there are several features specific to *Cutegram* that make it our favourite. The QML design is lightweight and fast, fitting in well with almost any desktop. Transitions are smooth and give a very modern style to the user-interface, and you can send and receive messages from more than one account too, even with 'emoticons', whatever they are. Telegram's best privacy/security feature is also easily accessible, providing end-to-end encryption with your contacts, theoretically making it very difficult for a third-party to crack the data.

We all now send many messages today, and it's very likely that all the protocols you use are insecure (look at SMS, for instance). Telegram isn't perfect, but it is a great application that's more open than most.



On Start: Automatic
Tray icon style: Automatic
Languages: English (...
Theme: Abrisham
Emoji's theme: twitter
Search Engine: https://d...
Notify Sound: Default
Master Color: System ...

Telegram's encryption reportedly uses Diffie-Hellman key exchange – we hope this has been updated in teh wake of the NSA's supposed learning how to crack this method.

PROJECT WEBSITE
http://aseman.co/en/products/cutegram

Software synthesizer

Oxe 1.3.3

At the very beginning of Tom Cruise's classic 1980s movie, *Top Gun*, Harold Faltermeyer's soundtrack begins with a very distinctive low chiming sound, and it's a sound that can be heard across many other recordings of the era. This sound is a preset from the most revolutionary synthesizer of the time, Yamaha's DX7, and the sound was called 'TUB BELLS' in reference to the instrument made famous by Mike Oldfield over a decade before.

The DX7 was revolutionary for two reasons. Firstly, its sound generator was driven by frequency modulation. This is where a simple 'carrier' waveform has its frequency modulated by another waveform operating within the audio range, creating almost infinite complexity. The sounds it generates are incredibly distinctive, especially for brass, bass, string and bell-like timbres, and they're quite different from the classic subtractive sounds of older synths.

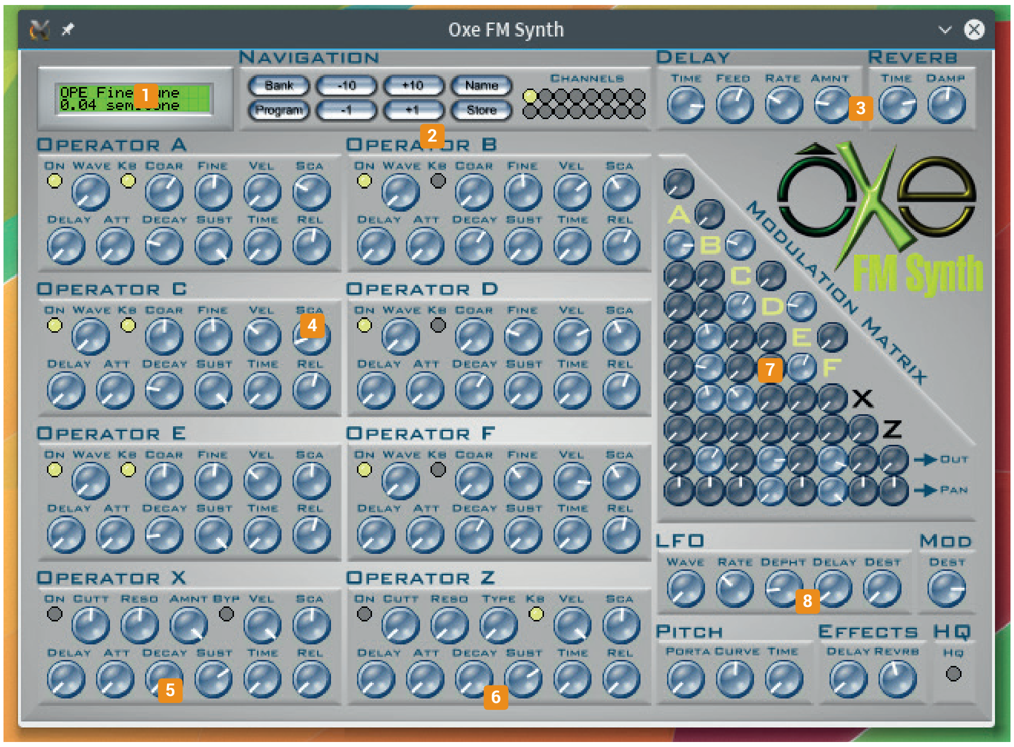
Oxe grinding

The second reason the DX7 was revolutionary was because it was digital. Yamaha cannily bought the rights to the algorithms in the 1970s, and by the 1980s, advances in integrated circuits meant it could finally implement in software what other synths were doing in hardware. They built a synthesizer that was technically superior while costing considerably less than the competition. The DX7 was cheap, robust, duo-timbral and polyphonic, unlike almost any other synth. This is why the sound of Yamaha's FM synthesis is all over the 80s. While early FM sounded fresh and modern, its ubiquity soon left it feeling cheap. You'll have heard them in almost every game from the mid-90s, for example, as a single chip of an FM synthesizer was nearly always bolted on to every soundcard and console.

As you'd imagine from one of the first digital synthesizers, there have been many recreations in software. But good ones are rare, which is why when one of the best Windows FM synths became open source and then started bundling a Linux version, we had to take notice.

> Oxe FM is a VST synth plugin that replicates the sound of the DX7 synth

Oxe FM is a VST synth plugin that recreates the sound of the DX7. There's even an optional skin to make it look like one. To install it, you'll need a VST-compatible host such as *QTractor* or *Ardour*. You then place the pre-compiled **.so** binary into a location that you add to the plugin path of your host. The plugin should then appear just like any other. FM synthesis is still complicated, and this makes the *Oxe* GUI look more intimidating than you might expect. There are six operators, just like the original DX7, plus a noise generator and a filter. These are all mixed together in a huge bank of knobs known as a modulation matrix. This cleverly allows you to mix the input from one source into another, as well as the final output. But you don't need to understand anything about FM to use the synth. It comes with a couple of banks of excellent presets, revealing the 1980s in all their Day-Glo glory, and you can obviously change and adapt these sounds to suit your own purposes. We think it sounds fabulous, and with FM making a retro-comeback, there's never been a better time to get re-acquainted with the DX7.



1 Display For parameter feedback and preset names **2 Presets** Switch between the 2 banks of 127 presets **3 Effects** Add excellent delay and reverb effects **4 Operators** These generate sound from a preselected waveform **5 Noise/Envelopes** Each operator has control of amplitude over time, plus pitch, except this noise generator **6 Filter** Mix outputs from the other operators into a simple filter **7 Mod Matrix** Set modulation levels for each operator, and mix values for filter, noise and outputs **8 LFO** Add repeating modulation and change mix levels.

PROJECT WEBSITE
www.oxesoft.com

FOSSPICKS Brain Relaxers

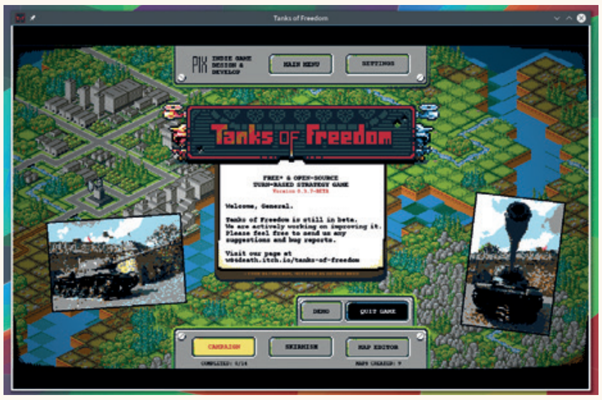
Strategy game

Tanks of Freedom 0.3-7 beta

If you read our tutorial in issue 20 on the *Godot* games creation engine and were intrigued by what kind of results might be possible, *Tanks of Freedom* is a perfect example. It's a excellent old-school turn-based strategy game, where you move and upgrade units to take best advantage of your resources. Its design is gorgeous and soaked in nostalgia thanks to its isometric pixel art, fabulous chip-tune music and artefact-laden speech synthesis. The 16-bit pixel art of older versions has been updated to 32-bit, but its isometric layout and the movement of the units very much feels like an old game. The gameplay will feel familiar to anyone who's played Westwood's old *Dune II* game, and you can play a campaign, or a one-off skirmish, and battle against other humans. There's even a map editing mode for creating your own scenarios, which is great fun in itself. Despite the game's beta status, we found performance was excellent, with the game already playable and addictive enough to keep you playing.

Without packages for our distribution (Arch) we needed to first grab the *Godot* games engine, which needed to be built, and then download the latest *Tanks of Freedom* files from the code repository. These totalled only 12MB and included the game logic, artwork and sound. All we then needed to do was add the configuration file as a new project within *Godot*, pressing Play to the launch the game. Launching from the *Godot* engine also means you can make your own contribution, or just dive in to see how it works.

Overall, this is a well thought out and designed game that genuinely brings that old-school RTS feeling to your Linux desktop, and definitely worth a look if you've got some time to fill.



FREE* & OPEN-SOURCE
TURN-BASED STRATEGY GAME
Version 0.3.7-BETA

Welcome, General.

Tanks of Freedom is still in beta.
We are actively working on improving it.
Please feel free to send us any
suggestions and bug reports.

Visit our page at
w84death.itch.io/tanks-of-freedom

We love the original soundtrack that comes with the game, complete with speech synthesized title screen effects.

PROJECT WEBSITE
http://w84death.itch.io/tanks-of-freedom

Platformer

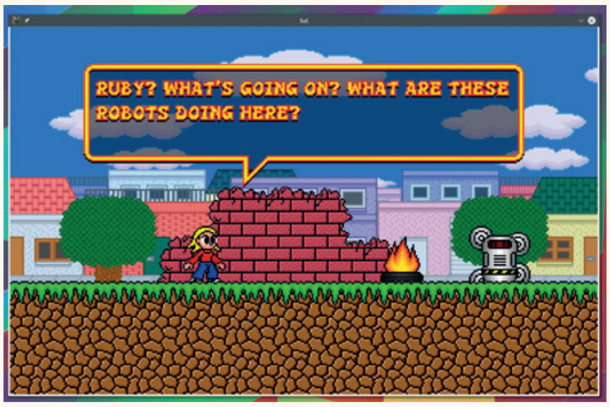
Sol 1.2

Sol is a brilliant platformer that's a little different to the games we usually feature in this section. That's because it's being sold for $14.99. But what's especially impressive is that the game really is open source, and you can still download, copy and build your own version from the GPLv3-licensed source code. However, as the website says, "We trust you to support us." is also what we do here at Linux Voice, so we'd highly recommend downloading the demo, playing the first three levels and buying the game if you like it.

The game itself is 18 levels of tough platform action. Graphically it reminds us of *Alex Kidd*, an arcade game from the 80s, but game mechanics are also borrowed from perennial classics like *Mario*. The game is bright, colourful and challenging. The visual style is primitive, but the level design is absolutely top-notch.

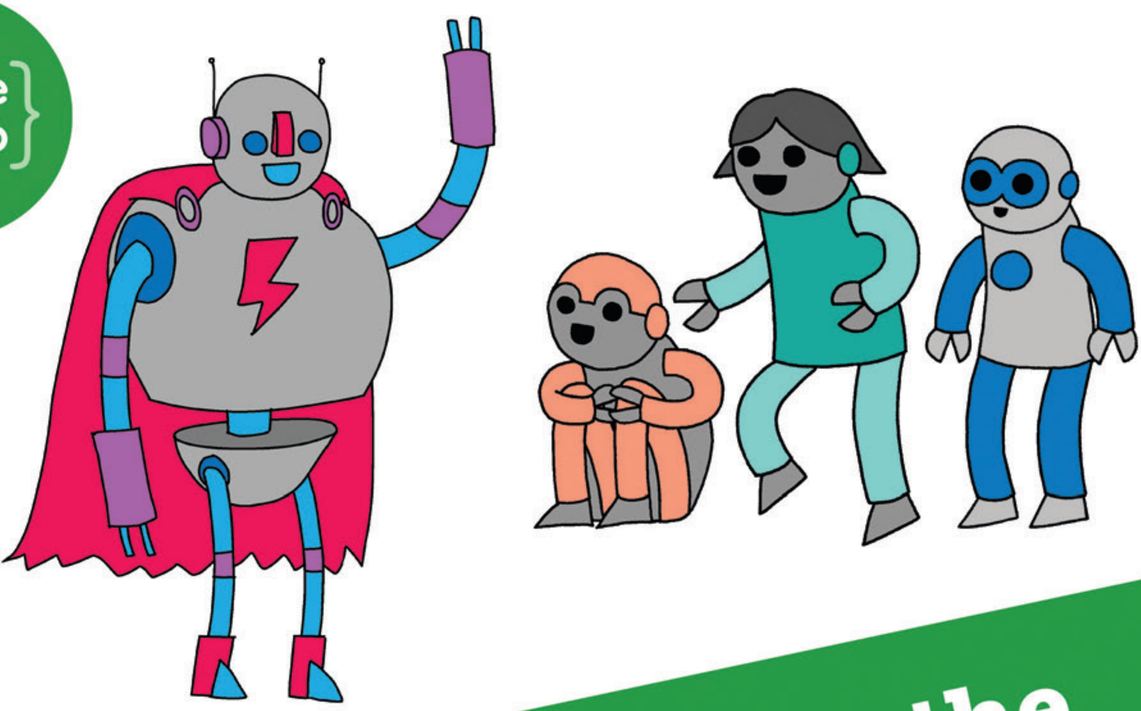
The toughness of the levels is countered by unlimited lives, which seems like an unusual choice for a game like this. It means you spend more time experimenting and simply enjoying the levels, but it removes much of the tension and stress that goes with a platform game, especially a game with aspirations for 1980s nostalgia. However, properly designed platformers (especially ones with properly thought-through story arcs) are difficult to find, and *Sol* is a great example. We'd love to see the game become successful enough that the developers write another, and release that as open source too. LV

The visual style is primitive, but the level design in Sol is absolutely top notch



***Sol* is a game you can buy and download; but you can also download the GPLv3 licensed source code and build it yourself.**

PROJECT WEBSITE
http://sol.azurasun.com



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**

This advertising space was kindly donated to Code Club by Linux Voice

TUTORIALS

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

Ben Everard
Makes mistakes, but tries to learn from them. You should too.

I've been battling hand pain for a little over a year now, and I finally feel like I've found the right combination of exercises, computer peripherals and medication to work comfortably again. It's not been a fun experience. You should learn from my mistakes: don't ignore twinges; don't put off going to the doctor because you feel you're too busy and don't wait until it's too late to make changes to your work setup.

We should consider using a computer to be a dangerous activity because, well, it is. There's a good chance that sitting at a desk for work will, at some point, leave you in pain.

This isn't something you can abdicate to your employer's health and safety team, because you'll suffer a lot more than them if anything goes wrong. Take a little time now to research the best posture and ergonomics. Be prepared to spend a little on a decent keyboard and mouse. Think very carefully before committing to using a laptop long term. They may be convenient, but do they enable you to sit in a safe position? Don't wait until it becomes a problem: take action now.

ben@linuxvoice.com

In this issue . . .

SHOWER: BUILD HTML-BASED PRESENTATIONS

Making a presentation should be no harder than writing a blog post.

VALENTINE SINITSYN

WHY DO THIS?

- HTML is a standard that renders the same everywhere
- Focus on content, and let the system handle styling
- Easily publish your slides online or export them to PDF

Whatever job you do, you're likely to do presentations from time to time. A *de-facto* standard slide maker is *Microsoft PowerPoint*. It's certainly powerful (perhaps too much for an occasional presenter), but non-free, and provides no support for Linux to date. *LibreOffice Impress* is a close free alternative, and it can even handle *PowerPoint* documents, up to a point.

The trouble is that quite often *PowerPoint* documents are rendered differently on different machines. You know what we mean: fonts could be different shapes and sizes, equations missing or garbage and so on. *Impress* can always export your slides to PDF: this way, you gain fidelity but lose much of the interactivity. Modern web browsers are quite powerful and flexible, too – so, why not use a browser as a presenter tool?

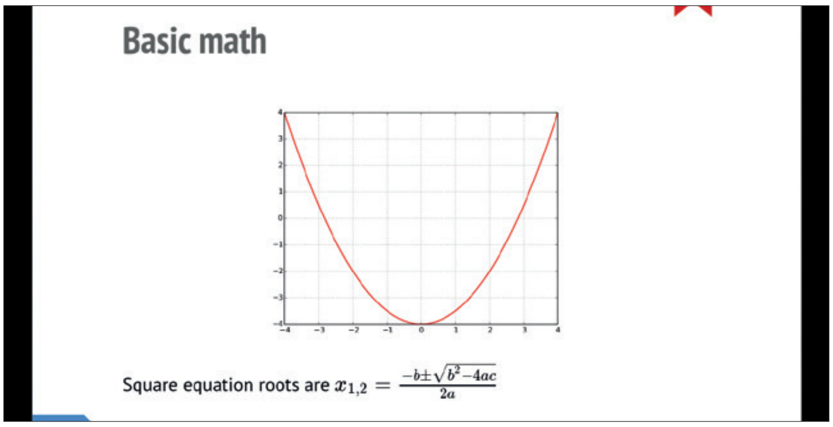
Shower is a JavaScript library that makes it easy to create presentations with plain HTML and CSS. There's no visual editor as in *PowerPoint*/*Impress*, but if you already use HTML or Markdown for your blog, everything should go smoothly.

To start a presentation, download **http://shwr.me/shower.zip** and unzip it. Now, open your favourite text editor and start making changes. The archive already contains many "lorem ipsum" slides of varying layouts you can use as templates or for reference. Usually, this is more effective than starting from scratch.

After you finish a slide or two, save your work and preview the presentation (**index.html**) in a browser.

PRO TIP

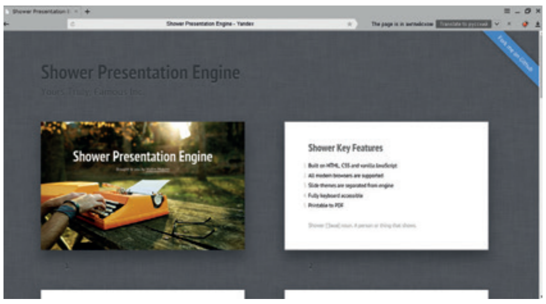
When you copy-paste a slide, remember to change its ID, otherwise *Shower* may behave oddly.



Basic math

Square equation roots are $x_{1,2} = \frac{-b\pm\sqrt{b^2-4ac}}{2a}$

Your slides may include not only images, but also equations, thanks to the *MathJax* library.



Shower Presentation Engine

Yours Truly, Famous Inc.

Shower Presentation Engine

Shower Key Features

1. Built on HTML, CSS and vanilla JavaScript
2. All modern browsers are supported
3. Slide themes are separated from engine
4. Fully keyboard accessible
5. Printable to PDF

Shower ['ʃəuə] noun. A person or thing that shows.

Simply toggle the browser window to full-screen, and press F5 to begin the presentation.

Shower recognises several hotkeys: Space/→/↓ moves to the next slide, Shift+Space/→/↑/ brings you back, and F5 toggles presentation mode, as in *Impress*. For that reason, you can't use F5 to reload a page. If this bothers you, use browser add-ons, like Auto Reload for *Firefox*.

Laying out slides

Shower treats everything with a **slide** class as a slide. Usually, it's **<section/>**, and the slide body goes wrapped in a **<div>** inside it. You can use any HTML markup you like, but as usual, avoid being too noisy. Better stick to lists, paragraphs, headings and images.

For starters, give your presentation a title. Simply edit the contents of **<title/>** in the page head, and **<header/>** in the body. Now, proceed to slides. Text goes in **<p>**, and headings use **<h2>**. Bullet (**<ul>**) and ordered (**<ul>**) lists, hyperlinks (**<a>**), quotations (**<blockquote>**) and even tables are readily supported and styled appropriately. Each slide has an ID (either explicitly assigned or an automatically generated ordinal: 1, 2, 3...), so you can create cross-references via **<a href="#id"/>**. Sometimes, a slide may carry just a few words, like "Questions?". In these cases, use the **shout** class to style it. Non-default *Shower* themes may define additional classes.

The **<footer>** tag is somewhat special. Its contents are hidden in presentation mode, but shown on a mouseover in the slides view, which is convenient for leaving notes to self:

```
<section class="slide"><div>
 <footer>Remember the milk</footer>
</div></section>
```

You can also define your own custom styles with **<style/>**. Usually you do it straight in a slide's body. Styles are reusable across slides, and they come handy to position images, for example:

```
<section class="slide centred"><div>
 <img src="images/chart.svg">
 <style>
  .centred img {
   width: 60%;
   margin-left: 20%;
   margin-right: 20%
  }
 </style>
</div></section>
```

This works best for scalable image formats, like SVG. Alternatively, the **cover** class stretches an inner image across the slide like background.

Shower doesn't sport funky animations like *Impress* or *PowerPoint*, but it does provide some interactivity. If you add the **next** class to any slide fragment, it will remain hidden until you advance to it with Space or another shortcut. This way, you can reveal contents as you proceed with your speech.

Going further

As you can see, *Shower*'s feature set is rather basic. Still, your presentation is just a web page, so there are many ways to enrich it.

Say you want to show a code snippet. *Shower* can do it out of the box, but that doesn't look particularly impressive. There are more capable JavaScript syntax highlighters available, and I usually choose *Prism* for simplicity and language support.

Start with downloading minified JavaScript and CSS files from **www.prismjs.com**. Use a configurator to select the theme and languages you need, then put the files, say, under the **prismjs** folder next to the presentation's **index.html**.

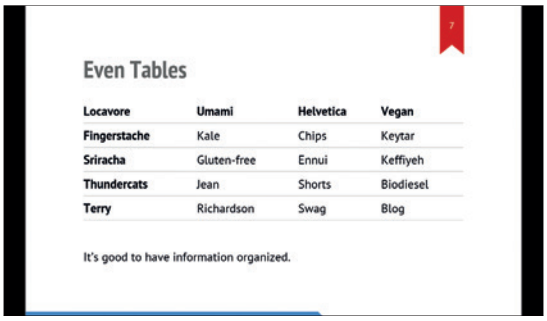
Now, include Prism's CSS in **<head>** and JS – in the bottom of **<body>**:

```
<head>
 ...
 <link rel="stylesheet" href="prismjs/prism.css">
</head>
```

Jekyller

You may not like or know HTML, but in a blogging era you are almost certain to use some other markup, like Markdown. If so, you can still run *Shower* thanks to *Jekyller*. As the name suggests, it's based on Jekyll (**www.jekyllrb.com**), a free static website generator written in Ruby.

Jekyller is especially handy if you have a GitHub account. Just fork it from **https://github.com/shower/jekyller**, make changes as needed, commit them, and push back to GitHub. In a minute, your slides will be translated to GitHub Pages and made available at **http://your_name_here.github.io/jekyller** for free! You don't even need to carry your presentation around on a USB stick anymore. Still, if you want to, you can convert your Markdown presentation to plain HTML locally with the **jekyll** command.



Even Tables

Locavore	Umami	Helvetica	Vegan
Fingerstache	Kale	Chips	Keytar
Sriracha	Gluten-free	Ennui	Keffiyeh
Thundercats	Jean	Shorts	Biodiesel
Terry	Richardson	Swag	Blog

It's good to have information organized.

Shower even makes your tables look stylish.

```
<body>
 ...
 <script src="prismjs/prism.js"></script>
</body>
```

That's it! Now, wrap your code snippet like this:

```
<pre><code class="language-python">print('Hi')</code></pre>
```

and it should render highlighted. Don't forget to escape HTML special characters like **<, >** or **&**. Also, *Shower*'s slides aren't big, so be picky and show relevant lines of code only.

If you're making a scientific report, it's equally easy to embed beautifully looking formulas. The *MathJax* library renders Tex, MathML or even ASCII math straight inside a browser. Grab it from **www.mathjax.org**, but this time it'd be a big download: the library spans more than 30,000 files counting towards 150 megabytes when unpacked. This can make syncing your slides to Dropbox or unzipping them rather slow.

Two solutions are possible. First, you can use *MathJax CDN*, and I'd opt for this unless you are unsure about internet availability. Or, you can trim *MathJax* locally, either with Grunt or a Python script (**https://github.com/yuexue/small_mathjax**). The former is an official option, but it relies on particular web developer tools. If you don't have them installed, the latter would probably be simpler.

Final touches

So far, we've covered all aspects of a typical presentation. But before you start a show, there are some minor issues to address. You may not like a progress bar going along the bottom of a slide, or you may not want the "Fork me on GitHub" ribbon, if your slides aren't really on GitHub.

Both are easy to remove. To get rid of the progress bar, delete **<div class="progress"/>**. **<p class="badge"/>** renders the ribbon, and you can remove it as well.

Finally, you may want a PDF version of your slides for 100% fidelity, or for hand-outs. *Shower* handles this easily: just open your presentation in list mode and use the "Save as PDF" option available in *Chromium*-based browsers. Alternatively, you can do it from the command line with the **wkhtmltopdf** tool.

Dr Valentine Sinitsyn teaches physics, develops high-loaded services and does other clever things with Python.

FAKING INPUT – TYPE WITHOUT A KEYBOARD

Fed up with typing? Write a script to inject keystrokes into any terminal.

JOHN LANE

WHY DO THIS?
- Learn how terminals work
- Automate keyboard input

Imagine a really complicated command – one that's far too complicated to type and one that differs every time it's used. One that even keyboard-junkies would baulk at. One that you still need to edit and use interactively. A lot.

You think to yourself "I know, I'll write a script to generate the command line for me and write it at the prompt ready for me to edit before pressing Enter to run it".

Fantastic! It should be easy, right? You get writing and suddenly realise that your **echo** and **printf** output, although looking pretty good, isn't showing at the prompt. Then the penny drops: standard output is the wrong place. What about writing to standard input ?

> One of the defining features of Unix is that everything is a file, and files can be read from and written to

But how? Surely there must be a way? Well there is, kind-of. In fact there are a few ways, and this month, we'll share them with you.

One of the defining features of Unix is that "everything is a file" and files can be read and written. Every process gets three of them by default that we know as the standard input, output and error. But they're actually the same thing. See this:

```
$ ls -l /dev/fd/
lrwx - - 1 john users 64 Jul  9 09:33 0 -> /dev/pts/22
lrwx - - 1 john users 64 Jul  9 09:33 1 -> /dev/pts/22
lrwx - - 1 john users 64 Jul  9 09:33 2 -> /dev/pts/22
```

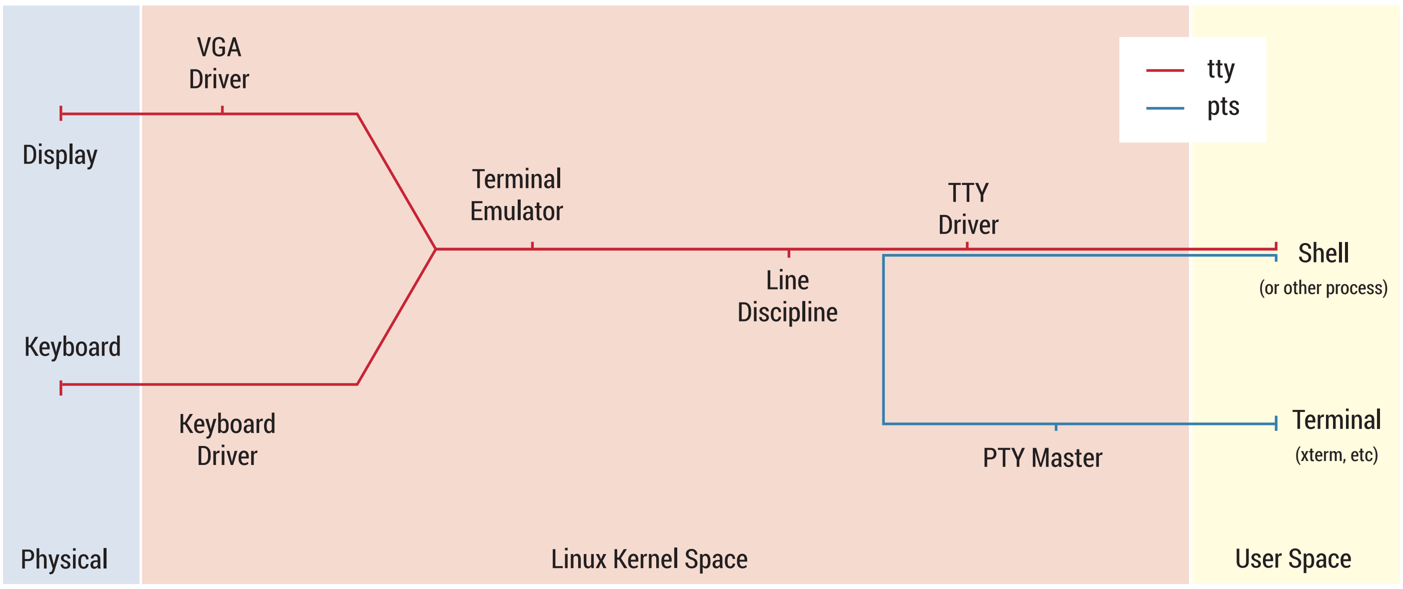
These files (technically they're "file descriptors" rather than real files) are for standard input (0), output (1) and error (2) but they're all just symbolic links to the same thing: a file representing the terminal that you're using. This is most likely a terminal window on your desktop (a pseudo-terminal, or **pts**, implemented in software) rather than being a real one. If you aren't running a graphical environment then you'll see something like **/dev/tty1**, which is the terminal implemented in the Linux kernel that displays text on your monitor and accepts input from your keyboard.

It's also possible, although less likely these days, that your terminal is separate hardware connected to your computer via a serial line (this remains a popular way to connect to embedded or ARM-based small-board computers). Whatever kind of terminal you're using, you can see its file:

```
$ tty
/dev/pts/22
```

What you type on your keyboard can be read from this file and anything written to it will be output. You

The route from terminal to process always goes through the kernel.



Other ways to inject keystrokes

If your environment meets certain prerequisites then you may have other methods available that you can use to inject input.

If you're in a desktop environment, **xdotool** is an X.Org utility that simulates mouse and keyboard activity, but your distro may not include it by default. You can try:

```
$ xdotool type "ls"
```

If you use **tmux**, the terminal multiplexer, you can do this:

```
$ tmux send-key -t session:pane ls
```

where **-t** selects which session and pane to inject. GNU **screen** has a similar capability with its **stuff** command:

```
$ screen -S session -p pane -X stuff ls
```

If your distro includes the **console-tools** package then you may have a **writevt** command that uses **ioctl** like our examples. Most distros have, however, deprecated this package in favour of **kbd**, which lacks this feature.

can try this: open another terminal and type

```
$ echo -e "ls\n" > /dev/pts/22
```

The command will appear in the other terminal but it won't be executed. Why? What you've done is effectively the same as writing to standard output: what gets written to the terminal gets displayed on the terminal. Think of that file as one end of a pipe. What you put in comes out the other end and you can only take out what's been put in at the other end. The other end is the terminal: if it's a **pts** then it's a desktop application such as *Gnome Terminal*, *Konsole* or maybe just *Xterm*. If it's a **tty** then we're talking about code inside the kernel. Remember that a shell like *Bash* is a separate process to the terminal. It's connected to the end of the pipe represented by the **pts** file; the "**s**" means "slave" and the "master" is the terminal's end.

To send data to the shell we need to put it in the master end of the pipe that's inside the terminal. Can we get the terminal to send something that didn't come from its keyboard?

Escape sequence initiated...

Since the days when most terminals were devices connected to serial ports, they have supported escape sequences. These are sequences of one or more characters that can be sent to the terminal that aren't displayed but are instead interpreted as commands. They were originally used to configure connection parameters but now have many purposes such as cursor positioning and colour. Try this:

```
$ echo -e "\e[31mThis is RED"
```

The **\e** is interpreted as the escape character (Esc, ASCII code 27). You should see some text displayed in red. What's interesting about this is that there are some escape sequences that cause the terminal to return other escape sequences – characters that didn't come from the keyboard! We can send a device status request and the terminal will respond that it's OK by sending four characters: **<ESC>[0n**.

We can then use a feature built in to *Bash* that replaces input characters with others. The **bind** command sets this up:

```
$ bind '"\e[0n": "ls"'
```

Now, whenever the terminal sends an OK status, the shell will output **ls** instead. It will appear at the prompt and you can hit Enter to execute it. We just need to request a device status from the terminal:

```
$ echo -e '\e[5n'
```

You'll see **ls** appear after the prompt with the cursor after it, just as if you'd typed it.

There are limitations to this approach, not least that the shell needs to support key binding. It won't work in sub-processes, which means that it won't work in scripts unless they are sourced instead of executed. It may suffice for some applications and relies solely on *Bash* internals and a terminal that plays along (practically all do). But there's a better way...

Tap the pipe

Let's go back to the pipe. It leaves the terminal at its master end and ends up with its slave end in the shell. The route it takes to get there goes through the kernel, as the diagram illustrates, passing the Line Discipline, which implements device semantics such as control characters (interrupt, kill, etc), and the TTY Driver.

The TTY driver is a kernel device driver and, like all device drivers, has a control interface that is accessible via the kernel **ioctl** system call – a generic function for sending commands to device drivers. The commands supported by the TTY driver are documented on the **tty_ioctl** man page and one of them is of interest to us.

The **TIOCSTI** (Terminal ioctl Send-Terminal-Input) command is used to inject characters into the input stream – they go straight into the pipe and come out in the userspace process when it reads its standard input. When the shell does this it displays the received characters at its prompt.

There is no command built into the shell for this; doing so requires an external command. There isn't such a command in the typical GNU/Linux distribution, but it isn't difficult to achieve with a little programming. Here's a shell function that uses Perl:

```
function inject() {
 perl -e 'ioctl(STDIN, 0x5412, $_) for split "", join " ", @ARGV' "$@"
}
```

You can then do:

```
$ inject ls -l
```

which prints **ls -l** after the prompt and followed by the cursor, ready to be executed when the user hits Enter.

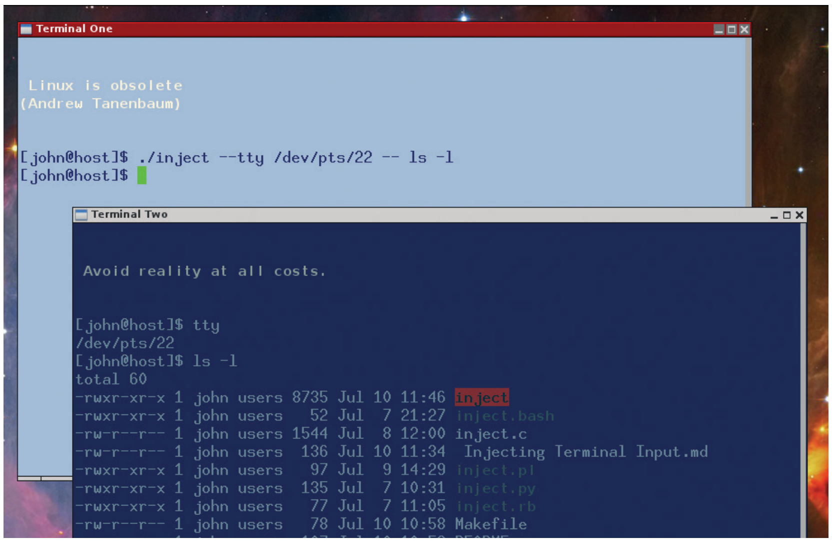
You may prefer to create standalone scripts in your favourite language. Here's one in Perl (**inject.pl**):

```
#!/usr/bin/perl
ioctl(STDIN, 0x5412, $_) for split "", join " ", @ARGV
```

0x5412 is the value of the **TIOCSTI** constant defined in the standard C header file. You can

PRO TIP

The origin of **tty**, which is an abbreviation of teletype, can be traced back to the invention of the stock ticker in the 19th century.



```
 Linux is obsolete
(Andrew Tanenbaum)

[john@host]$ ./inject --tty /dev/pts/22 -- ls -l
[john@host]$
```

```
 Avoid reality at all costs.

[john@host]$ tty
/dev/pts/22
[john@host]$ ls -l
total 60
-rwxr-xr-x 1 john users 8735 Jul 10 11:46 inject
-rwxr-xr-x 1 john users   52 Jul  7 21:27 inject.bash
-rw-r--r-- 1 john users 1544 Jul  8 12:00 inject.c
-rw-r--r-- 1 john users  136 Jul 10 11:34  Injecting Terminal Input.md
-rwxr-xr-x 1 john users   97 Jul  9 14:29 inject.pl
-rwxr-xr-x 1 john users  135 Jul  7 10:31 inject.py
-rwxr-xr-x 1 john users   77 Jul  7 11:05 inject.rb
-rw-r--r-- 1 john users   78 Jul 10 10:58 Makefile
```

With the right permissions it's possible to control another terminal.

generate the equivalent Perl header, **sys/ioctl.ph** and then use **TIOCSTI** instead of using the numeric value:

```
$ (cd /usr/include; sudo h2ph -a -l sys/ioctl.h)
```

Now the script can be written a little more legibly:

```
#!/usr/bin/perl
require "sys/ioctl.ph";
ioctl(STDIN, &TIOCSTI, $_) for split "", join " ", @ARGV
```

If you don't like Perl, perhaps Python is your thing (**inject.py**):

```
#!/usr/bin/python
import fcntl, sys, termios
del sys.argv[0]
for c in ' '.join(sys.argv):
 fcntl.ioctl(sys.stdin, termios.TIOCSTI, c)
```

or, perhaps Ruby (**inject.rb**):

```
#!/usr/bin/ruby
ARGV.join(' ').split('').each { |c| $stdin.ioctl(0x5412,c) }
```

or even C (**inject.c**):

```
#include <sys/ioctl.h>
int main(int argc, char *argv[])
{
 int a,c;
 for (a=1, c=0; a< argc; c=0 )
  {
   while (argv[a][c])
    ioctl(0, TIOCSTI, &argv[a][c++]);
   if (++a < argc) ioctl(0, TIOCSTI," ");
  }
 return 0;
}
```

Compile the C code to a binary

```
$ gcc -o inject inject.c
```

Control other terminals

Using **ioctl** to do this works in subshells. It is also possible to inject characters into another terminal, subject to having the appropriate permissions. Normally this means being "root" but we'll explain some other ways too. The only difference is that the relevant terminal file needs to be used. So, instead of using file descriptor zero (our own standard input) when calling **ioctl**, we need to open the relevant terminal file and use its file descriptor instead:

```
 fd = open(f, O_WRONLY|O_NONBLOCK);
```

where **f** is the required file (eg **/dev/pts/25**), and then

```
 ioctl(fd, TIOCSTI, &c);
```

It defaults to the current terminal but accepts a command line argument to specify another one. It also sends a newline by default but, similar to **echo**, it provides an option to suppress it. The GNU *ArgParse* library is used to process the command line options.

Compile it with **gcc -o inject inject.c**. Prefix the text to inject with **--** if it contains any hyphens to prevent the argument parser misinterpreting command-line options. See **./inject --help** for an explanation of the command line options and use it like this:

```
$ inject --tty /dev/pts/25 -- ls
```

or to inject the current terminal:

```
$ inject  -- ls
```

We mentioned that injecting into another terminal requires an administrative privilege and this can be obtained by:

- Running the command as root,
- With **sudo**,
- giving it the **CAP_SYS_ADMIN** capability or
- setting its setuid bit.

To assign **CAP_SYS_ADMIN**:

```
$  sudo setcap cap_sys_admin+ep inject
```

To assign "setuid":

```
$ sudo chown root:root inject
$ sudo chmod u+s inject
```

Keep it clean

You may have noticed that injected text appears ahead of the prompt as if it were typed before the prompt appeared (which, in effect, it was) but it then appears again after the prompt.

One way to hide the text that appears ahead of the prompt is to prepend the prompt with a carriage return (**\r**, not line-feed) and clear the current line (**<ESC>[M**):

```
$ PS1="\r\e[M$PS1"
```

However, this will only clear the line on which the prompt appears. If the injected text includes newlines then this won't work as intended. Another solution disables echoing of injected characters. A wrapper uses **stty**, a tool that uses **ioctl**, to do this:

```
saved_settings=$(stty -g)
stty -echo -icanon min 1 time 0
inject echo line one
inject echo line two
until read -t0; do
 sleep 0.02
done
stty "$saved_settings"
```

where **inject** is one of the solutions described above, or replaced by **printf '\e[5n'** if you're using the escape sequence method instead of **TIOCSTI**.

John Lane types a lot but he has a script that takes over when he's tired.

YOUR AD HERE

LV

Email andrew@linuxvoice.com to advertise here

PYTHON 3: BUILD A QUIZ MACHINE

Programming logic meets cardboard and sellotape in our latest Python/Pi project.

LES POUNDER

WHY DO THIS?
- Learn how to control a stepper motor
- Learn Python 3
- Control hardware using the GPIO
- Interface a touch screen
- Learn logic

TOOLS REQUIRED
- A Raspberry Pi
- A touchscreen
- Speaker
- Stepper motor http://bit.ly/LV22-Stepper-Motor
- Arts and crafts materials

Learning to code is a great experience but how can we make it more fun? In the past coding has been a rewarding, if daunting experience that comes with many successes and failures. With the rise of the Raspberry Pi we see a new era of physical computing, merging software with homebrew hardware, which is a great method to teach children as there are many physical outputs to keep interest high and reward learning. But what if we could build a machine that could test the knowledge of our children and be a great source of fun and tinkering? Well we have: it's called the Vend-A-Python.

For this project we shall be using the latest Raspbian image from the Raspberry Pi website. Jessie, the latest release, now comes with a new method to access the GPIO pins. Previously only the root user or a user using sudo was able to access the GPIO, but with Jessie any user can access the GPIO and hack hardware. Raspbian Jessie also comes with four of the five Python libraries that we shall be using, these are **RPi.GPIO**, **Time**, **Random** and **Pygame**, and we'll need to install one more, which is **Easygui**. In a terminal type the following, then press Enter.

```
sudo pip-3.2 install easygui
```

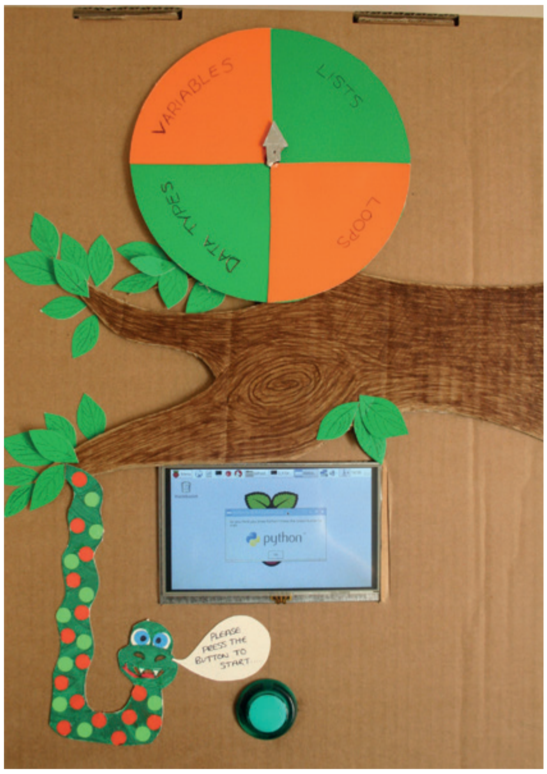
> With the rise of the Raspberry Pi we see a new era of physical computing, which is a great way to teach children

Easygui, as its name suggests, is an easy library to create menu and interfaces with Python, but more on that later.

Setting up the hardware

We'll start by setting up our stepper motor, which is a motor with a high degree of precision (512 steps, which control a full revolution). Using these steps we can precisely control the position of the motor, and later in our code we shall divide the faces of our wheel into four sections, effectively creating four zones each with 128 steps.

Our stepper comes with a controller board with four pins labelled IN1–IN4. Using female-to-female jumper cables, connect these as follows to the GPIO.

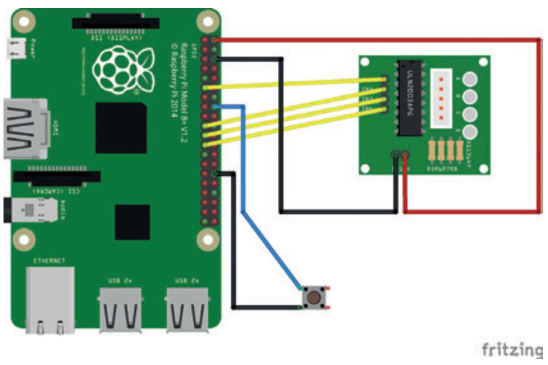


Our finished project combines motors, buttons, screens and crafting into one project. This project could also ask questions from other subjects.

Stepper	GPIO
IN1	17
IN2	10
IN3	9
IN4	11

We're using the Broadcom (BCM) pin mapping for the GPIO pins, which is the standard supported by the Raspberry Pi Foundation in all of its resources. For further reference please see **http://pi.gadgetoid.com/pinout**.

Also present are two pins labelled 5–12v. These two pins are + and -, and are power (+) for the motor and Ground (GND, -). From the GPIO of your Raspberry Pi connect 5V to the + and GND (Ground) to -. If you wish you can also connect these pins to an external power source. Next let's connect a button to the GPIO. The button is used to trigger the process and is easy to connect. We used an arcade button, as we had it lying around, a simple micro switch can be used in its place. Connect one side of the button to pin 23, remember we are using the BCM numbering, and the other to GND. Refer to the diagram, below, for details.



For the touchscreen we used the Adafruit 5-inch HDMI backpack, which required an extra step to configure the touchscreen for use. However, we'd recommend picking up the new official Raspberry Pi display screen as the touchscreen and display work out of the box with Raspbian Jessie. Assemble the screen and mount it as you see fit.

Connect your speaker to the 3.5mm port on your Pi; you can change the output method by right-clicking on the speaker icon in the top-right of the screen.

Coding!

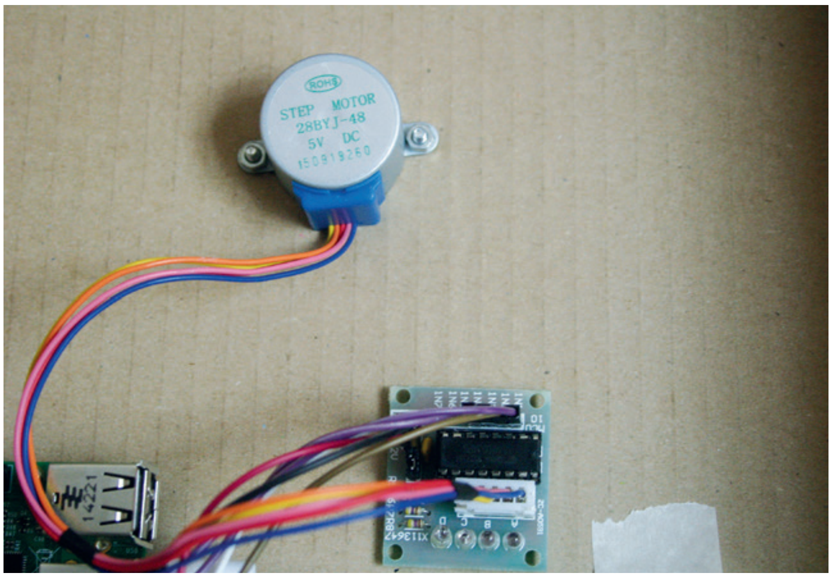
Before we commit any code, let's step back and look at the logic that will control our project.

The project starts waiting for the user to press the push button. Once it is pressed, music is played while the stepper motor rotates around the wheel, which is split into four areas, with each area covering a particular Python topic. How far it travels is handled via a function that uses a randomly generated number between 1 and 512. Once the stepper returns to the top of the wheel, the user is asked a Python question based upon the topic where the stepper motor stopped on the wheel. The topic of the question is chosen by using the randomly generated number and a series of conditional statements that check the value against hard-coded values. The user answers the question by pressing the correct answer on the touchscreen, which triggers another conditional statement to check their answer. If correct, the player is rewarded, if incorrect the player is chastised. The project then resets and is ready to play again.

For this project we shall be using Python 3 via the *Idle* editor. As we're using Jessie we do not have to invoke *Idle* via the terminal using sudo, but can open it from the Programming menu.

With *Idle* open click on File > New Window to open a new blank document, and save it as **Vend-A-Python.py** before proceeding.

As always we start coding our project by importing a few libraries. First we import the **RPi.GPIO** library and rename it to **GPIO** for easier use. Next we import the **time** and **random** libraries. We import **easygui** and rename it to **eg** before finally importing the **pygame** library.


Our stepper motor came from eBay and cost less than £2. It provides a handy controller board that can be easily interface with a Raspberry Pi, Arduino or other single-board computers.

```
import RPi.GPIO as GPIO
import time, random
import easygui as eg
import pygame
```

Now we setup the GPIO pins. We instruct the Pi that we shall be referring to them using the Broadcom layout (**GPIO.BCM**) we also instruct the Pi to turn off any warning messages.

```
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
```

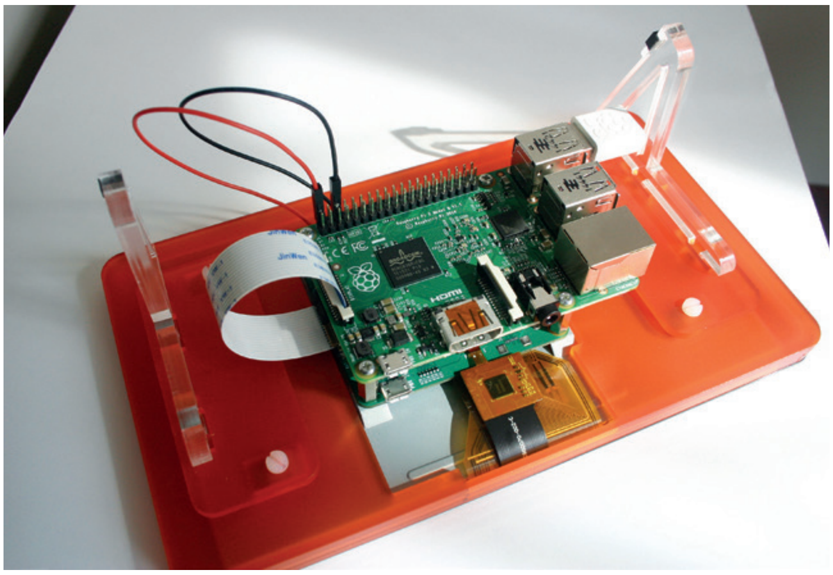
In this project we use variables to contain references to the GPIO pins and to control the delay used for our stepper motor. Here we can see the pins IN1–IN4 referenced on the stepper motor board and the pin used for the push button. We use their reference to store the GPIO pin used on the Pi.

```
IN1 = 17
IN2 = 10
IN3 = 9
IN4 = 11
button = 23
delay = 0.01
```

We store the values of the variables IN1–IN4 in a list and then use a **for** loop to iterate through each of the values stored in the list, the pins of the GPIO used for the stepper motor, and configure each of them to be an output. This means that current will flow from the GPIO to the pins on the stepper motor controller board.

```
outputs = \[IN1,IN2,IN3,IN4\]
for pin in outputs:
 GPIO.setup(pin, GPIO.OUT)
```

We now set up the push button to be an input and set the GPIO pin to be turned on. So when the button is pressed it will momentarily connect the pin, pulled high, to the Ground pin. This will cause the state of the GPIO pin to be pulled low, which will form the trigger


The official Raspberry Pi screen can easily hold a Raspberry Pi upon its back. It also offers a power passthrough, reducing the number of power supplies required.

used to start this project.

```
GPIO.setup(button, GPIO.IN, GPIO.PUD\_UP)
```

We now move on to the functions that will be used to contain more complex aspects of the project. Our first function is used to play audio. We define the name and then give the function an argument, denoted by the word in the brackets. In this case, the argument is the file name of the audio file to play. We then write the code that is to be run when the function is called. First we initialise the audio mixer. Next, we create a variable called **sound** to contain loading the audio file into the mixer. Finally we trigger the mixer to play the music once.

```
def audio(file):
    pygame.mixer.init()
    sound = pygame.mixer.music.load(file)
    pygame.mixer.music.play(1)
```

Our next function, called **cw** (short for clockwise), is used to control the stepper motor so that it moves in a clockwise direction. This function takes two arguments: the number of steps to move; and the delay between each part of the step. Inside the function we use the number of steps to create a **for** loop that will repeat for that number of steps. In the **for** loop we turn on each of the stepper pins, IN1 to IN4 in order, by using True to turn the pin on and False to turn it off. So for IN1 we turn it on, and the others are turned off. The code waits for the delay of 0.01 seconds, before turning the next pin on and turning the others off. This repeats for all of the pins IN1 to IN4 and causes the motor to spin once. Once the number of steps has been reached the code will wait for 5 seconds before calling another function.

```
def cw(steps,delay):
    for i in range(steps):
        GPIO.output(IN1, True)
        GPIO.output(IN2, False)
        GPIO.output(IN3, False)
        GPIO.output(IN4, False)
        time.sleep(delay)
        GPIO.output(IN1, False)
        GPIO.output(IN2, True)
        GPIO.output(IN3, False)
        GPIO.output(IN4, False)
        time.sleep(delay)
        GPIO.output(IN1, False)
        GPIO.output(IN2, False)
        GPIO.output(IN3, True)
        GPIO.output(IN4, False)
        time.sleep(delay)
        GPIO.output(IN1, False)
        GPIO.output(IN1, False)
        GPIO.output(IN1, False)
        GPIO.output(IN1, True)
        time.sleep(delay)
    time.sleep(5)
    ccw(steps,delay)
```

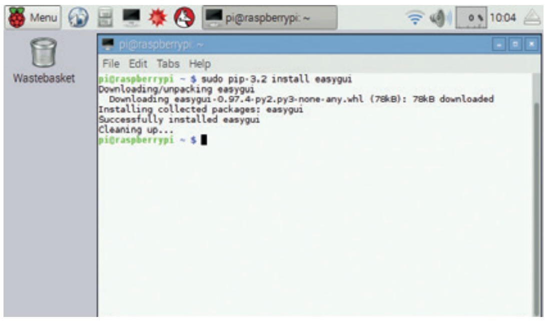
For the next function, **ccw** (short for counter clockwise) we reuse the same structure as **cw()**, but change the pin sequence so that the stepper moves in reverse.

This ends the functions, and now we move to the main body of code. We now start using **Easygui** to greet the player using a messagebox dialog. This function has three arguments: the title of the dialog box; an image to decorate the dialog; and finally the message to the user.

```
eg.msgbox(title="Welcome to the Python Quiz",image="./python.gif",msg="So you think you know Python? Press the Green button to start")
```

We use a **while True** loop to constantly check via an **if** conditional statement that the push button has been pressed. When the button is pressed its state changes from high to low, True to False. So when the pin reports False we call the **audio** function with the name of an audio file to play. When calling the audio file it is important to provide the path to the file as a string – simply wrap the file path in **""**. We can use an absolute file path which will precisely show the location of the file, or we can use a relative file path that will show the location of the file in relation to where our project code is.

Next we create a variable called **steps** and in there we use the random integer function from the **Random** library to pick an integer between 1 and 512, 512 being one full rotation of the stepper motor. Finally we call the **cw()** function and pass it the arguments **steps** and **delay** to control how far to rotate the stepper and



```
pi@raspberrypi ~ $ sudo pip-3.2 install easygui
Downloading/unpacking easygui
  Downloading easygui-0.97.4-py2.py3-none-any.whl (78kB): 78kB downloaded
Installing collected packages: easygui
Successfully installed easygui
Cleaning up...
pi@raspberrypi ~ $
```

Installing **Easygui** via the terminal is easy. Just remember to connect to the internet before trying to install…

how quickly to do so.

```
while True:
 if GPIO.input(button) == False:
   audio("./tilburg.mp3")
   steps = random.randint(1,512)
   cw(steps,delay)
```

Next we create an **if** conditional statement, which is used to compare a condition against a value. In this case we compare the value stored in the variable **steps** with two hard-coded values. For the **if** condition we check to see if **steps** is greater than 0 and less than 128, effectively covering the first quarter of the wheel similar to a clock face 12 to 3.

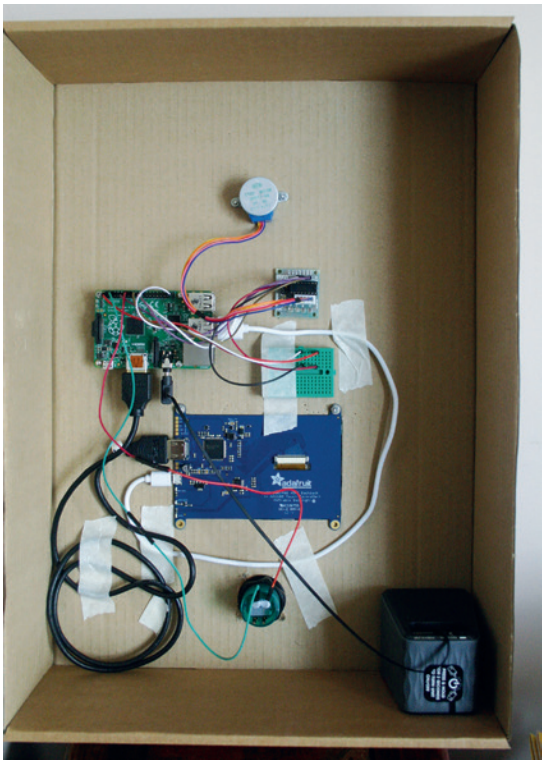
```
if steps \>0 and steps < 128:
```

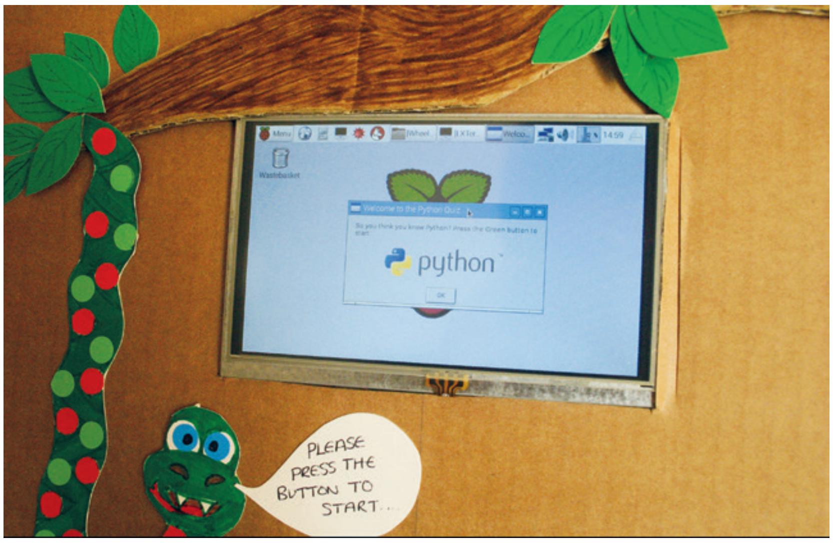
If this condition is true, and the stepper stops between 0 and 128, we ask the user a question based upon the subject area, which in this case is all about variables. To ask the question we first create a variable called **answer** and we use that to store the answer to the question posed via *EasyGUI*'s **choice** dialog. We use the **choicebox** function and give it three arguments: the title of the dialog; the message to the user; and the choices that can be made. When the user makes a choice it is stored in the variable for later use.

```
answer = eg.choicebox(title="Question",msg="Which variable is storing a string?",choices=("a='Hello World'","b = 5","c = 2.0"))
```

Next we compare the answer given to the correct answer, and if it is correct we reward the player with a pleasant piece of audio and use a message dialog box to inform the player of the achievement.

```
if answer == "a='Hello World'":
 audio("./correct.mp3")
```


We used an old cardboard box to house our project.


Welcome to the Python Quiz
python
PLEASE PRESS THE BUTTON TO START...
The *EasyGUI* library enables use of a simple GUI creation system that can be integrated into any Python project. It is compatible with Python 2.x and 3.x.

```
 eg.msgbox(title="CORRECT",image="./tick.gif",msg="RIGHT ANSWER")
```

If the player chooses a wrong answer a different audio clip is played and the text for the message dialog box is changed to reflect their status.

```
else:
 audio("./wrong.mp3")
 eg.msgbox(title="INCORRECT",image="./cross.gif",msg="WRONG ANSWER")
```

This process repeats using a series of **elif**, **else if** conditions to compare the position of the stepper motor for the other sections of the wheel. With all of the **elif** conditions complete we now break from this conditional statement and return to the main **if...else** conditional statement, which handles waiting for the button to be pressed. While it waits for input it simply prints "Waiting" to the Python shell before sleeping for 0.1 seconds, then repeating the process until the button is pressed.

This ends the code for this project. Remember to save your work and when ready click on Run > Run Module to run the code via *Idle*. The Python shell will now print "Waiting" to the shell. Press the push button and the stepper motor will come to life and start the quiz. At present this project only has one question per topic but it can be easily extended to add further questions, which can be chosen at random using the **random.choice** function from the **random** library. Use this tutorial as a platform to craft your own version of the project.

By completing this project we have learnt more about stepper motors, how connect a push button to the GPIO and the flow of the project has introduced loops, conditional logic and multimedia.

All of the code for this project can be found via our GitHub repository at **http://bit.ly/LV22Code**, or you can download a Zip file containing all of the project files from **http://bit.ly/LV22CodeZIP**.

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

KEEPS TABS ON YOUR PC WITH **BLUETOOTH**

Monitor your computer when away from your desk with a wireless link to your phone.

BEN EVERARD

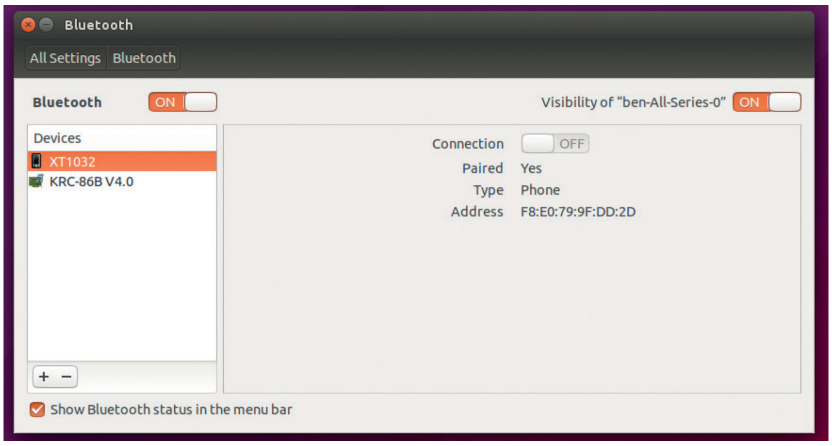
WHY DO THIS?
- Gain an extra mini screen
- Keep an eye on processes when you're away from your desk
- Learn to use serial ports for easy communication

Almost all smartphones, most laptops and quite a lot of desktops have Bluetooth hardware, yet it's rarely used beyond sending audio to wireless speakers. This is a shame, because it's powerful enough to send any data you want between devices, and gives you endless opportunities for hacking together new features.

In this tutorial, we're going to take a look how to use standard Linux tools to stream real-time data about our PC to our phone to give us an extra, portable screen to use to monitor our computer. We'll do this by using Bluetooth to create a serial port between our phone and our Linux machine. Serial ports fit very well with the Unix mantra that 'everything is a file', because they're created as files in the **/dev** folder and you can write data to them (to send) and read data from them (to receive). Using this, we can communicate with just the standard command line tools.

> We're going to use standard Linux tools to stream real-time data to our phone, to create an extra, portable screen

Before we get to this, though, we have to set up the software. First, you'll need a Bluetooth terminal app on your Android phone. There are a few options for this. The best open source option is *Bluetooth Terminal*, which is available via F-Droid at **https://f-droid.org/repository/browse/?fdid=ru.sash0k.bluetooth_terminal**. If you would rather install via the Google Play store, *BlueTerm* by **pymasde.es** also works.



Pairing in Bluetooth is the process of setting up two devices in anticipation of creating a connection. It only needs to be done once for each combination of devices.

Install the software

You'll also need some software on your machine, which your distro may have installed by default. This will include some software to handle the initial connection between the phone and the computer, and some software to set up a serial connection. To handle the initial connection between the two machines (known in Bluetooth terminology as pairing), you should find some graphical software on your desktop. In Gnome, this is *Gnome Bluetooth Manager*, in KDE this is *Blue Devil*; and in Unity this is the Bluetooth option in the Ubuntu Settings Manager. The process for all these is roughly the same.

First you need to make sure that Bluetooth is turned on in your phone's settings and that the phone is discoverable. This is done by going to the Bluetooth page in the settings app (you need to keep this page open to make the phone discoverable). Once this is open and turned on, open your Bluetooth manager on your Linux machine and make sure that Bluetooth is turned on, then select the option to add a new device. This should scan and find your phone. Follow the settings on the Bluetooth manager, and it should set up everything you need.

The pairing process sets up a general connection between the two device that can be used to create specific connections to share audio, send files or stream serial data. In order to send the data we want, we need to create a serial connection. This is a two-stage process where we first set up a Bluetooth serial port on the computer, and then connect the phone to it. You'll need the *rfcomm* tool, which you may have already installed – if not you'll need to get it from your package manager (in Debian-based systems, this is in the **bluez** package).

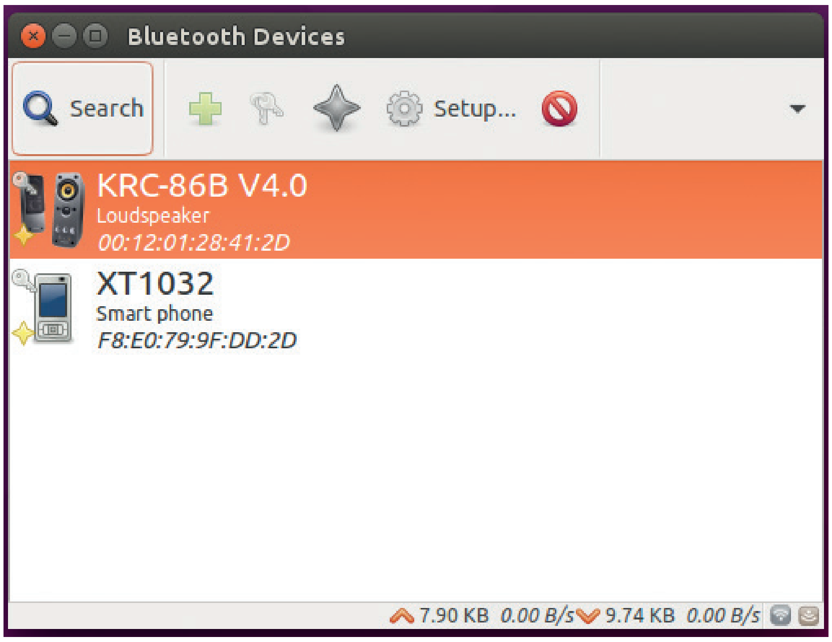
Due to an outstanding bug, the *rfcomm* software only runs if the user is root, so you'll notice that we have to run a lot of commands as root. The command to create a new Bluetooth serial port is:

```
sudo rfcomm listen 0
```

Now your PC is listening, you just need to point your

Turn your PC into a Bluetooth speaker

Ok, so using a PC as a Bluetooth speaker isn't exactly a great way to save money, but there are some occasions where it can be useful. Perhaps you have a home theatre PC set up already and you want to use it to play music off your phone as well. Perhaps you just need a way to play music on your PCs speakers for a one-off event. Whatever the reason, the method is straight forward. Firstly, you need to pair the phone and the PC in the way described in the main text. Once this is done, you need to tell the PC to treat the connection as an audio source. Using the Blueman-manager graphical Bluetooth management software, this is done by right-clicking on the connection and selecting Audio Source. After this, and audio from the phone will go through the PC rather than the phone.



Blueman is a little more capable than Unity's Bluetooth settings, so can be useful for Ubuntu users looking to set up audio or file transfers.

phone at the appropriate Bluetooth connection (which should already be set up since the devices are paired). This is just a case of opening the Bluetooth terminal app on your phone and in the connection options, select the PC. This will create the file **/dev/rfcomm0** on your PC (you can create more than one Bluetooth serial port at a time by increasing the number on the **rfcomm** command to create **/dev/rfcomm1**, **2**, etc).

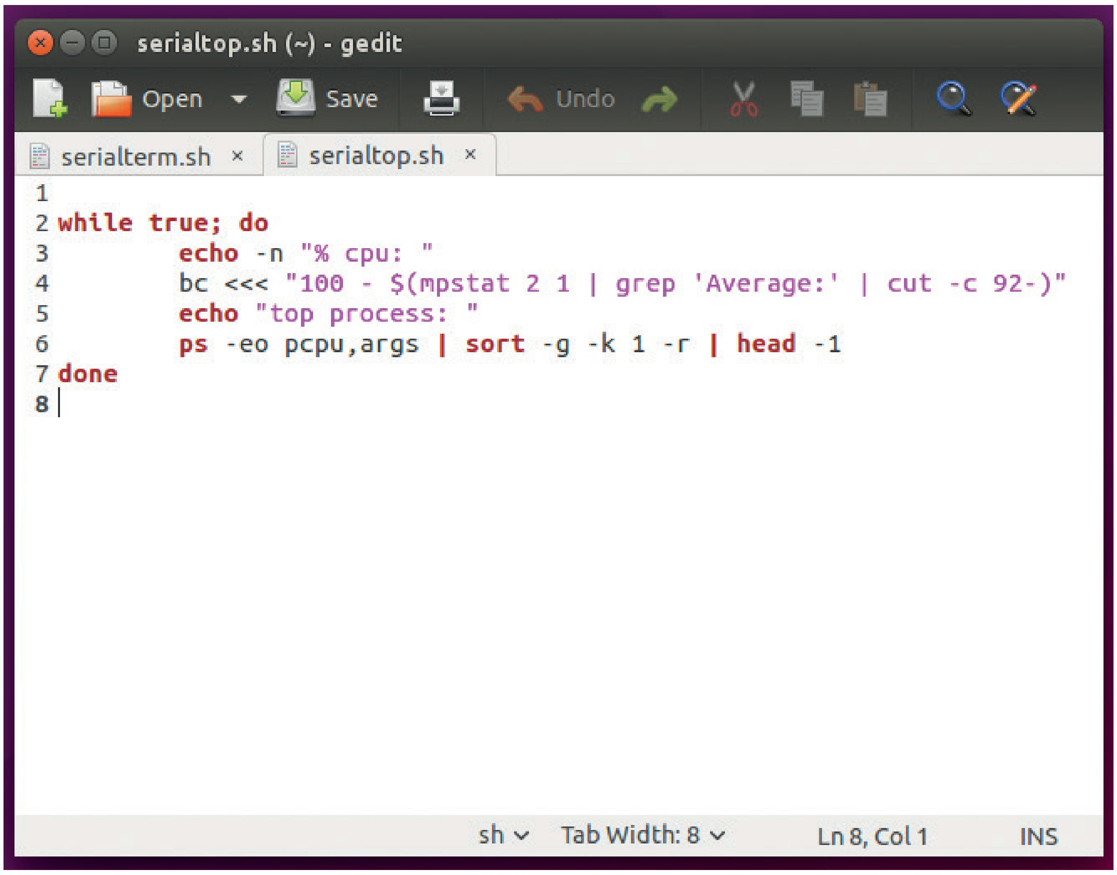
Anything you write to this file is sent to the Bluetooth terminal on the phone, so a simple test that everything's working is:

```
sudo bash -c 'echo "hello world" > /dev/rfcomm0'
```

The **echo** command outputs the text "hello world", and the 'greater than' sign tells the shell to send that text to the **/dev/rfcomm0** file (which is our serial connection). This command is a little more complicated than a regular **sudo** command, because we need the output redirect to run as root. If we'd run the command with just **sudo** as follows, it wouldn't have worked.

```
sudo echo "hello world" > /dev/rfcomm0
```

In this case, the **echo** command is run as root, however, the output (**> /dev/rfcomm0**) runs as the normal user. Instead, we need to use **sudo** to start a new *Bash* session running as root, and run the full

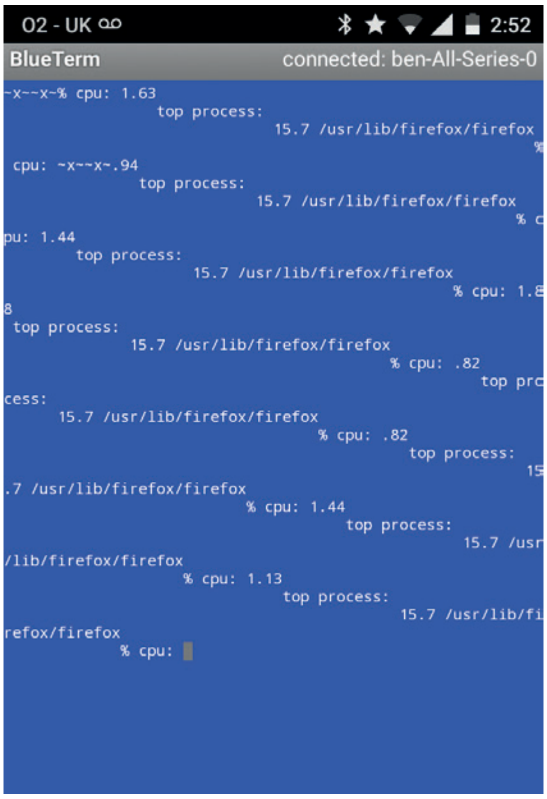


```

while true; do
        echo -n "% cpu: "
        bc <<< "100 - $(mpstat 2 1 | grep 'Average:' | cut -c 92-)"
        echo "top process: "
        ps -eo pcpu,args | sort -g -k 1 -r | head -1
done

```

A few lines of *Bash* script is all you need to send diagnostic information to your phone.



```
02 - UK                                    2:52
BlueTerm          connected: ben-All-Series-0
-x--x-% cpu: 1.63
                 top process:
                              15.7 /usr/lib/firefox/firefox
                                                           %
 cpu: ~x--x~.94
               top process:
                            15.7 /usr/lib/firefox/firefox
                                                         % c
pu: 1.44
        top process:
                     15.7 /usr/lib/firefox/firefox
                                                  % cpu: 1.8
8
 top process:
              15.7 /usr/lib/firefox/firefox
                                           % cpu: .82
                                                     top pro
cess:
      15.7 /usr/lib/firefox/firefox
                                   % cpu: .82
                                             top process:
                                                          15
.7 /usr/lib/firefox/firefox
                           % cpu: 1.44
                                      top process:
                                                   15.7 /usr
/lib/firefox/firefox
                    % cpu: 1.13
                               top process:
                                            15.7 /usr/lib/fi
refox/firefox
             % cpu:
```

Some Bluetooth serial terminal phone apps expect Windows-style line endings and can behave a little odd with the newer Linux line endings. This doesn't affect the content though.

echo and redirect in this root session. We did this in the first command by running **bash -c**.

There is a simpler option: **tee**. This command takes standard input and does two things. Firstly, it writes the input to a file and secondly it passes the input onwards to standard output. Since the file is written by the command itself, we can just run that as sudo. The above command can then be run as:

```
echo “hello world” | sudo tee /dev/rfcomm0
```

Getting interactive

Since tee sends the input to both the output and the file, you will see 'hello world' appear on both the Linux terminal and the phone's Bluetooth terminal. This is the first way we'll use our phone to monitor our machine. It's particularly useful if you want to set a long command running, and want to leave your machine unattended until it finishes. Pipe the output to **sudo tee /dev/rfcomm0**, and you can leave your machine alone, and make sure that it's still running by checking the Bluetooth terminal on your phone.

There is a slight problem with this approach – if the command doesn't give any output, you don't know when it's finished. You can solve this by running two commands one after the other, which is done using the semicolon. For example, the following will update a Debian system (sending the output to both the terminal and the phone), and then end with the word 'finished':

```
sudo apt-get upgrade | sudo tee /dev/rfcomm0 ; echo
“finished” | sudo tee /dev/rfcomm0
```

So far, we've used our Bluetooth serial port as a sink into which we've poured data, but haven't gotten anything back from it. If you look at the app on your phone, you'll see that you have the ability to send lines of text; however, because of the way we've used the connection so far (with commands that only write data out and not read data in), anything you send this way will be lost.

The simplest way to read the data you send is with the **tail** command. This just outputs the end of a file, and if you use the **-f** (follow) flag, it will continually monitor the file and output anything that gets written to the end of it. Usually, this is used to monitor log files as new data comes in, but it's also useful here. Since we want to show the whole file, not just the end, we also have to use the **-n +1** argument, which tells **tail** to show the lines starting with the first. The command to output the text sent from the phone to the computer is:

```
sudo tail -fn +1 /dev/rfcomm0
```

This in isn't itself very useful, because all it enables us to do is send text from the phone to the computer. In principal, you could create a very rudimentary chat system by using **echo** to send data one way and **tail** to receive it the other way, but this is fairly pointless.

Pipes are useful

Fortunately, we don't have to limit ourselves to just spitting text out onto a screen. Instead, we can pipe this data into other commands. A simple way to use this is to read the data from **/dev/rfcomm0**, evaluate it in *Bash*, and then pass the output back to the serial port. This can be done with the following script:

```
while read -r line < /dev/rfcomm0; do
            $line > /dev/rfcomm0
done
```

This uses the **read** command to step through the data that comes in the serial port. The first line starts a **while** loop that will continue to operate until

ObexFTP: Send files back and forth

The easiest way to send files over Bluetooth is using the ObexFTP protocol. You may find that you need to install additional software for this to work. On your phone, you'll need an app that understands the protocol, and there are quite a few options in the Google Play store. We used Bluetooth File Transfer, but others should work. On your Linux machine, you'll also need software to handle the communication.

Most graphical Bluetooth tools have some options for sending files, but it's often useful to be able to incorporate file transfer into scripts. For example, you could create a script that runs at a certain point every day (when you'll be at your desk) and backs up the data on your phone.

There's a command line tool called *ObexFTP*, which is in most distro's repositories. You can use it to get a list of all the files on your phone from your PC with:

```
obexftp -b -l
```

The result comes in a slightly awkward XML file, but you should be able to see what's going on. The general format for getting files is:

```
obexftp  -c <directory> -g <file>
```

There's some code and examples of how to do more complex things with *ObexFTP* at the tool's website: **http://dev.zuckschwerdt.org/openobex/wiki/ObexFtp**.

A very brief history of Bluetooth

Bluetooth is a set of specifications created by the Bluetooth Special Interest Group (SIG). The SIG came into being on 20 May 1998, and since then has continued to develop the standard as the technology has improved, and as the technology landscape changes to require different features. The latest incarnation (Bluetooth 4) includes a new specification for low-power devices that run off small batteries, and is designed with the internet of things in mind. There's also a version of the protocol designed for devices where data transfer rates are more important than power usage (Bluetooth High Speed), which can send data at up to 24Mb/s. These new technologies are helping to make sure that the wireless protocol is still relevant today despite being over 15 years old.

it reaches the end of the file, and since serial port files don't have an end (that is, they never return an end of file, they just don't always have data to return), this loop will keep going until the serial port is closed. The second line just evaluates the contents of the line in the shell, and passes the output of this back to the serial port. If you save this as a file called **serialterm.sh**, you can launch it with:

```
sudo bash -c "bash serialterm.sh 2> /dev/rfcomm0"
```

The last part of the command (**2> /dev/rfcomm0**) is needed to redirect any errors that occur in the execution of the script on to the serial port. Discretion is advised here as this will create a root terminal on your phone (though only when within Bluetooth range of your machine).

Top dog

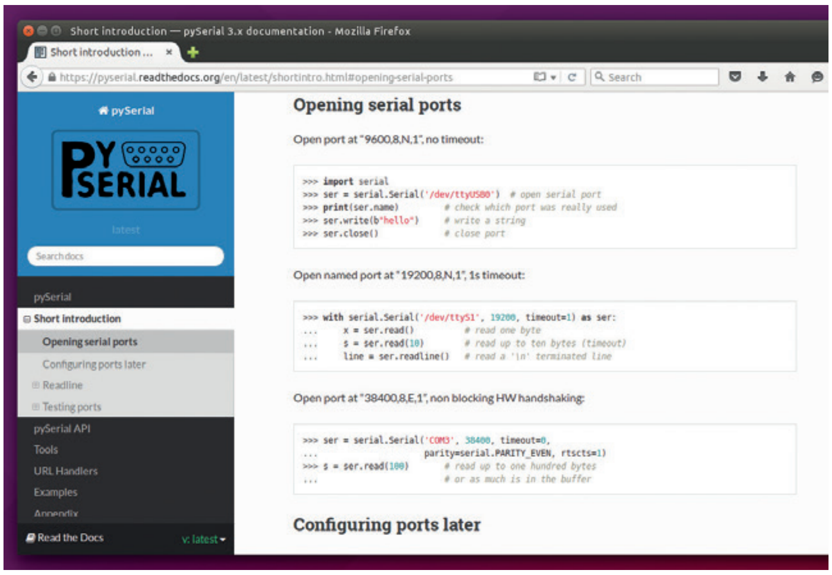
One particularly useful thing to do with our serial Bluetooth connection is monitor how much load there is on the CPU when we can't see the screen. This could be, for example, when using full-screen graphical applications. You could just pipe the output of the **top** command straight to the phone, but the different layout of the screen on the phone makes it a little awkward to read the data. Instead, we're going to create a stripped-down version of **top** that just outputs the CPU usage and the process that's using most of the CPU.

We'll use the **mpstat** command to get the processor utilisation and **ps** to get the CPU utilisation per process. The full script is:

```
while true; do
  echo -n "% cpu: "
  bc <<< "100 - $(mpstat 2 1 | grep 'Average:' | cut -c 92-)"
  echo "top process: "
  ps -eo pcpu,args | sort -g -k 1 -r | head -1
done
```

In the third line, we use **grep** to select just the line of the **mpstat** output that contains the average data, then **cut** to return just the characters that contain the percentage of time the CPU is idle. The **bc** command is a calculator, so we just send the input of 100 – the idle time to get the CPU utilisation.

The per-process utilisation from **ps** is piped into **sort** (the **-g** flag uses numeric sorting), and then into **head** to get the line with the highest processor utilisation.



Opening serial ports

Open port at "9600,8,N,1", no timeout:

```
>>> import serial
>>> ser = serial.Serial('/dev/ttyUSB0')  # open serial port
>>> print(ser.name)         # check which port was really used
>>> ser.write(b"hello")     # write a string
>>> ser.close()             # close port
```

Open named port at "19200,8,N,1", 1s timeout:

```
>>> with serial.Serial('/dev/ttyS1', 19200, timeout=1) as ser:
...     x = ser.read()          # read one byte
...     s = ser.read(10)        # read up to ten bytes (timeout)
...     line = ser.readline()   # read a '\n' terminated line
```

Open port at "38400,8,E,1", non blocking HW handshaking:

```
>>> ser = serial.Serial('COM3', 38400, timeout=0,
...                     parity=serial.PARITY_EVEN, rtscts=1)
>>> s = ser.read(100)       # read up to one hundred bytes
...                         # or as much is in the buffer
```

Configuring ports later

Most popular langages, including Python, have libraries for dealing with serial connections if you need more control over the data sent.

There is a little difference between the figures returned by the two commands. **Mpstat** will calculate the CPU utilisation over a short period (in this case, two seconds), while **ps** will calculate the average processor utilisation over the life of the process.

To run the data from this to your phone, just pipe the data through like we have done before. If you save the script as **serialtop.sh**, this is done with:

```
sudo bash -c "bash serialtop.sh > /dev/rfcomm0"
```

After this, everything should be piped through to your phone and you can keep an eye on your CPU usage even when the main screen is taken over by other programs.

These, of course, are just a few examples of what you can do with Bluetooth serial connections between your phone and your computer. If you want to take things further, you can make a serial connection

You could easily take the output from our final monitoring script and visualise it as something like a speedometer

from inside a custom-written application on your phone, which can take a particular format of data and process it in any way you wish. For example, you could easily take the output from our final monitoring script and visualise it, perhaps as something like a speedometer to show you how fast the computer is running at a particular time. Serial connections are almost endlessly flexible to allow a huge range of uses, but at the same time, as you have seen, they can be very easy to use.

In an unusual twist, Ben Everard is also monitoring GCHQ's machines from his Android phone.

SERVER 101: BRUSH UP YOUR DATABASE SKILLS

Part 2: Learn how to interact with a database using PHP, and build the killer web apps/tax dodging walled gardens of tomorrow.

MIKE SAUNDERS

WHY DO THIS?

- Discover how web apps work under the hood
- Explore data stored by *WordPress*, *OwnCloud* and co.
- Learn SQL to perform powerful search queries

Last issue we looked at the basics of databases: why they're important, how they work, and how to set up one from scratch. We also explored a handful of vital SQL commands to manipulate data and search for results. (If you're missing issue 21, grab it from **http://shop.linuxvoice.com** – or take out a subscription and get free access to every single one of our digital back issues!)

In this second part of the tutorial we'll delve further into SQL with commands to modify data, perform more advanced searches, and link search results across multiple tables. We'll then move on to accessing databases with the PHP programming language, providing you with the building blocks to make websites. By the end you'll have the skills to poke around inside web apps such as *OwnCloud*, *PhpBB*, and many others that are written in PHP and make extensive use of databases.

Advanced SQL

Let's continue with the database and table we set up last issue. Log in to *MariaDB* like so:

```
mysql -ulvuser -p
```

Enter **pass123** when prompted for the password. Switch to the **lvtest** database and list the tables it contains:

```
use lvtest;
show tables;
```

In the previous tutorial we used **select *** to retrieve all fields of a database entry, but it's important to note that we can narrow them down like so:

```
select ID, Name from login_dates;
```

This just shows the **ID** and **Name** columns, and omits anything else. We can restrict the results further:

```
select ID, Name from login_dates where ID > 1;
```

To change the data inside an existing row in a table, we use the **update** command, providing the name of the column we wish to change, its new value, and a reference to the specific row. For instance, if we want to modify the third row in our table, and change Graham to Ben, we would use this:

```
update login_dates set Name = 'Ben' where ID = 3;
```

Here are some other commands worth knowing. The first deletes a row from a table, while the second and third add and remove columns respectively. Remember that *MySQL* and *MariaDB* don't hold your hand when you're working – they'll happily delete vast amounts of data with just a few keystrokes! You get no chance to confirm, so when you're working with real-world data, tread carefully...

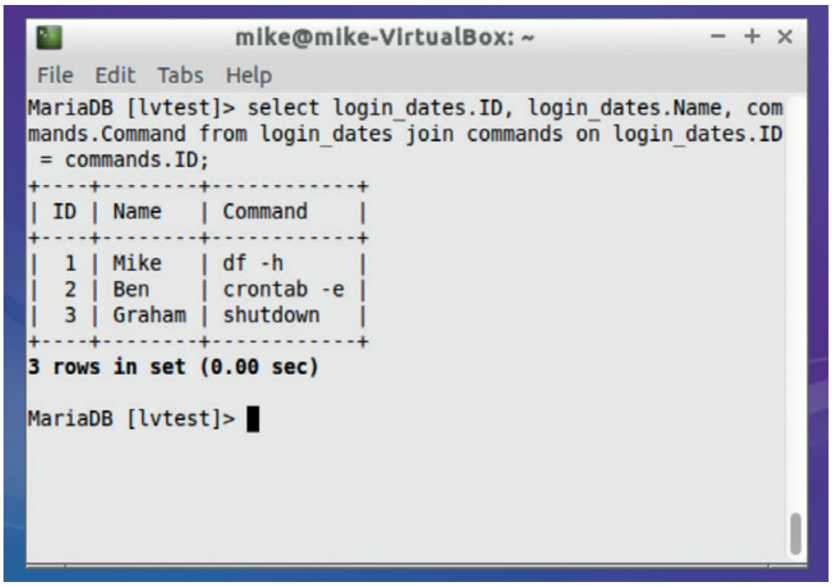
```
delete from login_dates where ID = 3;
alter table login_dates add Shell varchar(20) after Name;
alter table login_dates drop column Shell;
```

In the first **alter** command here, we add a column called **Shell**, which contains a string of up to 20 characters, and place it after the **Name** column. (If we omitted the **after** part, the column would simply be added on to the end of the table.) The second **alter** command removes this column and any data that it may contain.

Turning the tables

When we explored the concepts behind relational databases last issue, we used an example of a second table to go alongside the one we've set up, containing a command that was executed and its exit code. These tables both use the ID columns as primary keys, so we can cross-reference data between them. Let's create and populate this second table:

```
create table commands(ID int auto_increment primary key, Command varchar(255), ExitCode int);
```



```
MariaDB [lvtest]> select login_dates.ID, login_dates.Name, com
mands.Command from login_dates join commands on login_dates.ID
 = commands.ID;
+----+--------+------------+
| ID | Name   | Command    |
+----+--------+------------+
|  1 | Mike   | df -h      |
|  2 | Ben    | crontab -e |
|  3 | Graham | shutdown   |
+----+--------+------------+
3 rows in set (0.00 sec)

MariaDB [lvtest]> 
```

Screenshot 1: Here's the results of our 'join' operation, combining the Name column from one table with the Command column from another.

```
insert into commands values(0, 'df -h', 0);
insert into commands values(0, 'crontab -e', 1);
insert into commands values(0, 'shutdown', 1);
```

If you now enter **select * from commands;** you'll see the data we've just inputted. And if you look at each table separately, you can work out that for ID 2, Ben logged in at 2015-04-25 and ran the command **crontab -e**, which exited with code 1 (failure). But how do we pull this data together with SQL? What happens when we want some columns from one table, another set of columns from another table, but everything together in a single set of results?

This is where the mightily useful **join** command in SQL comes into play. Let's say we want to generate results showing the ID column, the name from the **login_dates** table, and the command that was executed from the commands table:

```
select login_dates.ID, login_dates.Name, commands.
Command from login_dates join commands on login_
dates.ID = commands.ID;
```

Phew – that was a mouthful! Let's go through it bit-by-bit. We start off by saying we want to generate results in three columns: **ID** and **Name** from the **login_dates** table, and **Command** from the **commands** table. We use **join** to insert data from the **commands** table into the results, and want results where the **ID** column matches in both tables.

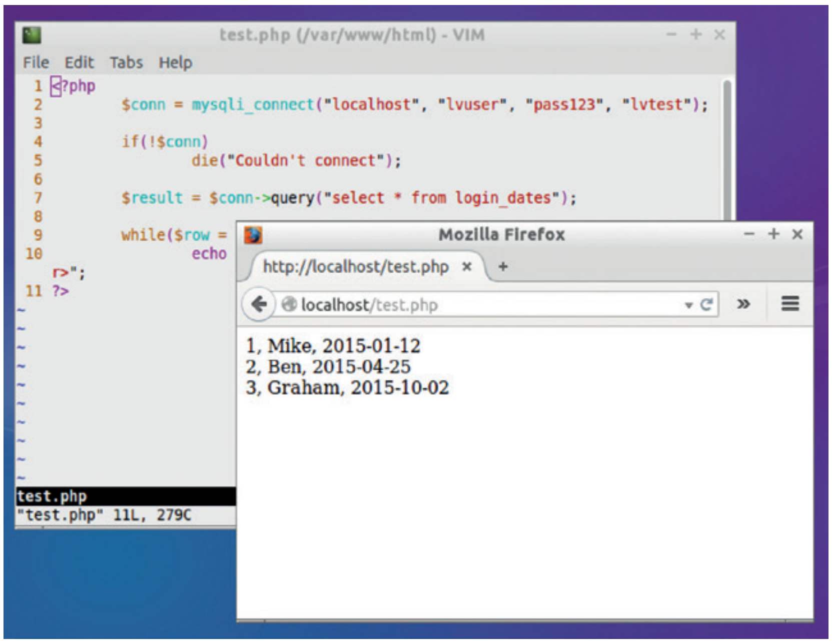
NoSQL: databases done differently

Relational databases power the web. MySQL/MariaDB, *PostgreSQL*, *Oracle*, *Microsoft SQL Server* and others chew through vast amounts of data every day, and they'll be with us for decades to come. But a new breed of databases that eschew the familiar table and relational models are coming up – and receiving a lot of attention. *NoSQL* is the moniker given to database software that takes a different approach.

MongoDB (**www.mongodb.org**) is one of the most famous *NoSQL* databases, and stores its information inside JSON (JavaScript Object Notation) documents rather than tables. JSON uses attribute-value pairs, and is somewhat like XML but designed to be easier to parse in JavaScript. Here's an example:

```
{
 "ID": 1,
 "Name": "Graham",
 "Commands": ["crontab -e", "shutdown"]
 "ExitCodes": [0, 1]
}
```

This shows a login entry with ID 1 for Graham, and in the **Commands** and **ExitCodes** fields you can see arrays denoted by square brackets. Potential benefits of the *NoSQL* approach include simpler database designs and better performance when scaling up to large clusters of computers, and many startup websites are going the *NoSQL* route. (Note that despite the name, some *NoSQL* databases still let you interact with data using SQL-like commands, making the transition easier.)



```
<?php
        $conn = mysqli_connect("localhost", "lvuser", "pass123", "lvtest");

        if(!$conn)
                die("Couldn't connect");

        $result = $conn->query("select * from login_dates");

        while($row =
                echo
r>";
?>
```

```
test.php
"test.php" 11L, 279C
```

http://localhost/test.php

localhost/test.php

```
1, Mike, 2015-01-12
2, Ben, 2015-04-25
3, Graham, 2015-10-02
```

Screenshot 2: It only takes a few lines of PHP to extract information from a database and render it as HTML.

Got that? See screenshot 1 for the results. SQL syntax – and especially **join** instructions – can get very complicated, which is why some admins use uppercase for commands, as noted last issue, to distinguish them from table and column names. When you're working with large data sets across multiple tables, **join** operations are immensely useful for narrowing down the information that you need.

Onto the web

Interacting with your data via the command line is rather tedious and completely unsuitable in the long run, so what are some alternatives? You could build a native application that talks to a database – eg to make a collection manager, human resources system or similar program. But a quicker (and more cross-

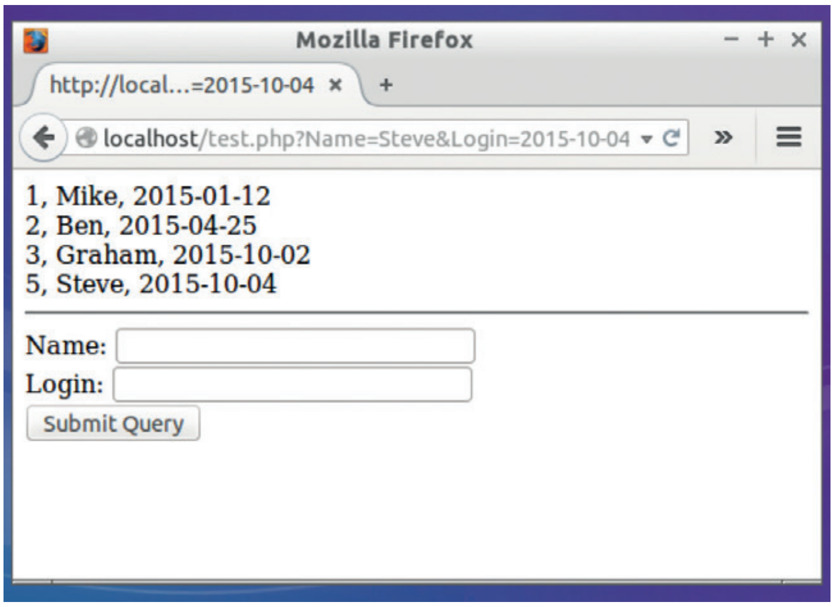
MySQL and MariaDB will happily delete vast amounts of data with just a few keystrokes, so tread carefully…

platform) approach is to make a website that hooks up to a database. Thanks to the PHP programming language, this is rather easy, and involves just a smattering of HTML and coding knowledge.

To get started, you'll need to install the *Apache* web server, the PHP language and a module that links both together. In Debian and Ubuntu-based distros you can grab the packages with:

```
sudo apt-get install apache2 php5 libapache2-mod-php5
php5-mysql
```

(If you're using a different family of distros, search through your package manager to find packages of the same or similar names.) Once the software is installed, visit **http://localhost** (or **http://127.0.0.1**) in your web browser to view *Apache* running on the


Mozilla Firefox
http://local...=2015-10-04
localhost/test.php?Name=Steve&Login=2015-10-04
Name:
Login:
Submit Query

1, Mike, 2015-01-12
2, Ben, 2015-04-25
3, Graham, 2015-10-02
5, Steve, 2015-10-04

Screenshot 3: The command line is fiddly, so we've provided a method for adding data via a HTML form. You can tart this up with a spot of CSS.

local machine. All being well, you'll see an "It works" message, so you can start using PHP.

Switch into the **/var/www/html** directory and create a file called **test.php** with the following contents:

```
<?php
        echo "PHP works!";
?>
```

If you've never used PHP before, it has a C-like

> Now you have all the skills required to create interactive websites that use databases as back-ends

syntax and all code must be supplied between **<?php** and **?>** tags, to differentiate it from HTML. So, open **http://localhost/test.php** in your browser, and if the language was installed correctly, you'll see the "PHP works" message. We're ready to go!

To work with the database, first we need to initiate a connection and associate that connection with an object. If the connection fails, we need to quit out (die) before doing anything else; otherwise we perform a query on the connection and store its results in a variable. Then we go through the results, parsing out the individual fields from the database. So let's use PHP to grab the data from our **login_dates** table and display it – save this again as **test.php**:

```
<?php
  $conn = mysqli_connect("localhost", "lvuser",
"pass123", "lvtest");

  if(!$conn)
        die("Couldn't connect");

  $result = $conn->query("select * from login_dates");

  while($row = $result->fetch_assoc())
        echo $row['ID'] . ", " . $row['Name'] . ", " .
$row['Login'] . "<br />";
?>
```

Most of this should be self-explanatory. The **mysqli_connect()** function is provided by the **php5-mysql** package that we installed earlier, and we pass four parameters to it: the hostname or IP address of the server to which we want to connect, a username, the password for that username, and then the database that we want to use. This function returns an object, which we store in **$conn**. We then check to see if **$conn** contains anything – if not, it means that the connection failed for some reason (such as invalid login details, or the database isn't running), in which case we quit with an error message.

If everything works, we then perform an SQL query, just like we would at the command line, passing the results back into the newly created **$result** variable. The last two lines may faze you a bit: essentially, after doing the database query, **$result** contains a number of rows. So in the **while** loop we go through each row and extract its contents into an associative array – in other words, an array where each element has its own name. In our case, these element names are the columns from our table, so we have **ID**, **Name** and **Login**.

Using PHP's **echo** command we spit this out as text, joining the three elements together with commas and spaces, and tacking a **<br />** tag onto the end to make the results more readable. The end result will be like in screenshot 2 – a (very rudimentary) HTML version of our **login_dates** table!

Of course, if you're a dab hand with HTML and CSS, you could now improve the output by using proper tables, divs and other fluff to make everything look nice. We're not going to focus on that here, as we have other things to do, but now you know how websites connect to and extract information from databases.

So that's displaying data – but what about feeding

Back up your data!

When you start to build up a lot of data, you'll want to make regular backups. *MariaDB* is a pretty robust database, but it can't save your hide if you suffer a major hardware failure or your hard drive throws in the towel. While *MariaDB* stores its data in rather complicated binary files, you can generate text versions for backup purposes using the **mysqldump** utility (at the command line) like so:

```
mysqldump -ulvuser -p lvtest > backup.sql
```

If you now look at **backup.sql** in a text editor, you'll see all of the SQL commands required to recreate and populate the tables inside the **lvtest** database, so you can gzip this up and store it somewhere as a backup. Later on, if you need to recreate the tables you use the regular **mysql** tool with the database name and filename:

```
mysql -ulvuser -p lvtest < backup.sql
```

You could, of course, automate the backup step by placing it inside a *Cron* job and running it every day (or multiple times a day, if you have enough disk space).

new data back into the database? What's the best way to go about this? There are a few options, but the simplest is to use a HTML form and some PHP to handle the results. At the end of **test.php**, after the **?>** (which terminates the PHP code), add this HTML:

```
<hr />
<form action="test.php">
Name: <input type="text" name="Name" /><br />
Login: <input type="text" name="Login" /><br />
<input type="submit" />
</form>
```

This is a simple HTML form that calls back to the same file (**test.php**) when the Submit button is clicked, and it has two text fields: **Name** and **Login**, as per our **login_dates** table. So this HTML table now appears under the information we extract from the database. In order to process the information when we submit the form, however, we need to do some work in the PHP section at the top. Underneath the "die" line, add these two lines:

```
if($_GET['Name'])
    $result = $conn->query("insert into login_dates
values(0, '$_GET[Name]', '$_GET[Login]')");
```

By default, when a HTML form is submitted its form fields are passed to the "action" file (in our case, the same test.php file) in an array called $_GET. This also means that the fields are supplied as part of the URL, as you'll see when you submit the form.

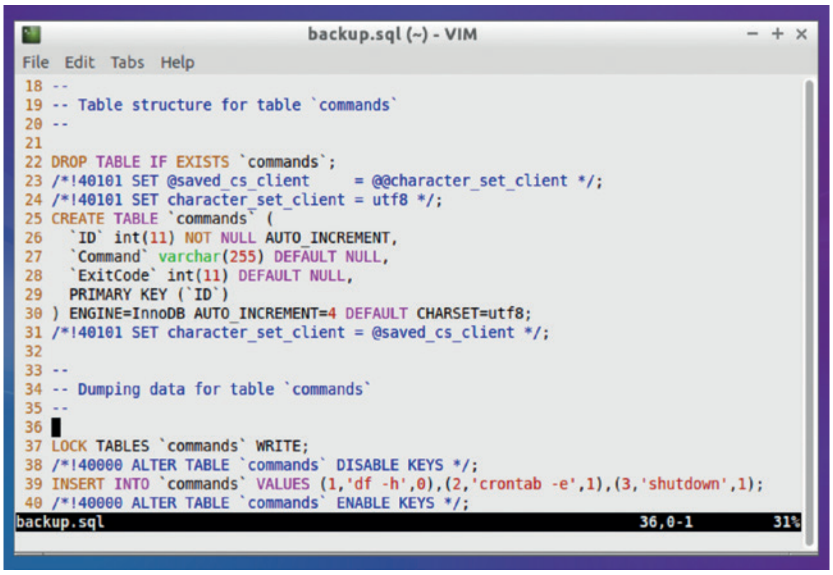
So we first check to see if anything was entered into the **Name** field – ie if it's not blank – and then we perform an SQL query, inserting the data as we've explained previously. Note that this is an extremely quick and simple way to perform the SQL query; in a real-world scenario, you'd want to perform many more validation and security checks against the data to make sure someone isn't craftily trying to submit executable PHP via the form! That's worth a whole other tutorial though...

So **test.php** now does three things: it shows the contents of the **login_dates** table, it provides a form for adding new data, and it processes the data and adds it to the database if the form is filled out. Give it a try – enter some text in the **Name** box and a suitably formatted date (eg 2015-10-04), click the **Submit** button, and you'll see the new row in the database when the page reloads as in screenshot 3.

And that's all the weather!

So now you have the basic skills required to create interactive websites that use databases as back-ends. And more importantly, you understand exactly how it works, down to the raw SQL instructions. There are countless web frameworks and abstraction layers out there that do all the hard work for you, and completely separate you from the gritty job of talking to the database – and they're hugely useful if you're making the next big Web 3.0 (or are we at 4.0 now?) website.

But as with assembly language or the build-it-yourself Linux From Scratch project, nothing beats knowing what's going on under the hood. Next time you're using a website with forms and data, you'll have a pretty good idea of how the website works and what it takes to store and retrieve such information.



```
--
-- Table structure for table `commands`
--

DROP TABLE IF EXISTS `commands`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `commands` (
  `ID` int(11) NOT NULL AUTO_INCREMENT,
  `Command` varchar(255) DEFAULT NULL,
  `ExitCode` int(11) DEFAULT NULL,
  PRIMARY KEY (`ID`)
) ENGINE=InnoDB AUTO_INCREMENT=4 DEFAULT CHARSET=utf8;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `commands`
--

LOCK TABLES `commands` WRITE;
/*!40000 ALTER TABLE `commands` DISABLE KEYS */;
INSERT INTO `commands` VALUES (1,'df -h',0),(2,'crontab -e',1),(3,'shutdown',1);
/*!40000 ALTER TABLE `commands` ENABLE KEYS */;
backup.sql                                                        36,0-1        31%
```

The Vim editor has syntax highlighting for almost everything under the sun, including SQL (useful if you're rummaging around in backups).

PHP, databases and related topics are all huge beasts themselves, so if you'd like us to dedicate some pages to one of them, just drop us a line. In particular, if you'd like to explore other databases such as *PostgreSQL*, or try interacting with databases using other programming languages, let us know.

In the meantime, here are a few tasks you can try with your new skills – if you get stuck, someone should be able to help at **http://forums.linuxvoice.com**:

PRO TIP
You may have come across the term 'LAMP' before: this refers to a stack of software typically used to serve up websites. The letters stand for Linux, *Apache*, *MySQL*/*MariaDB* and PHP – although the latter can be replaced by Perl or even Python in some instances. Similarly, some sites have moved away from Apache and are using more lightweight alternatives like Nginx.

1 Use tables or divs to make the **login_dates** HTML look much better – maybe spruce it up with some CSS too.
2 Check that the user has filled in the **Login** field as well as **Name** in the form. You can combine multiple tests together into the same **if** operation.
3 Provide a way for a user to delete an item. For instance, you could achieve this using a drop-down list, performing a separate SQL query if a number is selected in the list.
4 Check that the dates in the **Login** field are of a valid format. This is where PHP's string handling facilities come into play.

Some of these may require more PHP knowledge than we've gone over here, so have a look at the excellent tutorials at **www.w3schools.com/php** for ideas. Happy hacking!

Mike Saunders is working on his own database, MikeSQL, written entirely in 16-bit x86 assembly language.

MINSKY: DYNAMIC SYSTEMS MODELLING

Get stuck into some complex maths, without having to learn complex maths.

ANDREW CONWAY

WHY DO THIS?
- Do mathematics without equations
- Program without coding
- Hack the economy!

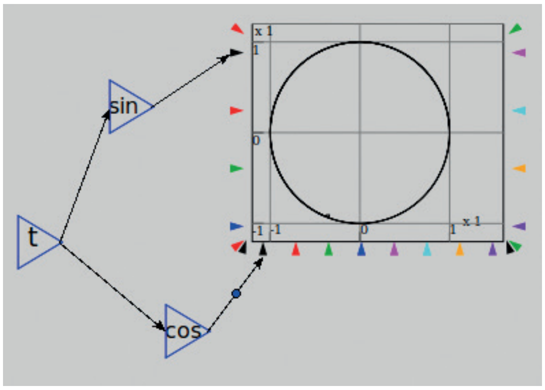
In many disciplines – science, engineering, medicine and economics – models are used to predict behaviour to save on costly, dangerous or impossible real-world experiments. Computers are used to run numerical models that describe everything from fish populations to the behaviour of the Universe in its first moments.

Building these models requires skills in mathematics and programming in addition to an understanding of the particular subject involved. For some disciplines that are already very reliant on mathematics, such as physics, this sits well, but for others a different approach is needed.

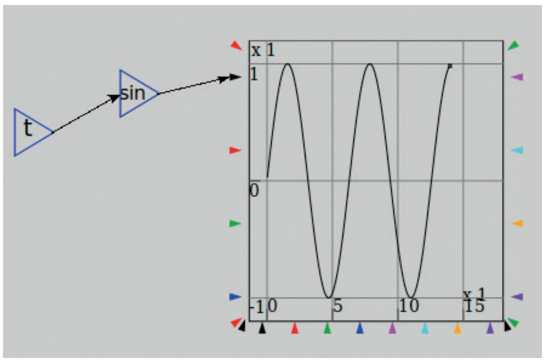
This is where *Minsky* comes in – it's a FOSS tool than enables you to construct complex dynamic numerical models using a graphical interface. Rather than formulate equations, then writing code, you drag and drop icons to see the results in a graph. Not only can you share this with mathematically minded friends, but you'll probably find yourself learning a lot about mathematics in an agreeably intuitive fashion.

Minsky is released under GNU GPL v3, so you can build it from the source code available from **sourceforge.net/projects/minsky**, but be warned: it does have a few fiddly dependencies. It probably won't be available in your distro's package manager, but it can be found in the OpenSUSE build service. Check for specific instructions provided for your distro, but on Xubuntu 15.04 the installation involved the following commands:

```
sudo sh -c "echo 'deb http://download.opensuse.org/
repositories/home:/hpcoder1/xUbuntu_15.04/ /' >> /etc/
apt/sources.list.d/minsky.list"
```



Plot sin against cos and you end up with a circle – odd, but true.



Here's a sine wave plotted in *Minsky*.

```
sudo apt-get update
sudo apt-get install minsky
```

We're using version 1.D037, which was available from the OpenSUSE build service, but there's a more recent beta if you don't mind building it yourself.

All the screenshots produced for this article have a corresponding **.mky** you can load up for yourself, though we recommend you try to wire up at least the most basic models. You can grab the **.mky** files from **https://github.com/mcnalu/linuxvoice-minsky** and load them using the Open item under the File menu.

Simple building blocks

Minsky's interface is straightforward. There's a menu bar, controls for playing, stopping and adjusting simulation speed, a palette of components, and the main canvas.

Let's start with something simple: plotting a straight line. See the boxout for detailed instructions, but in short we create a time component and wire it to the graph, then start the simulation.

Now let's plot the sine function. Right-click on the wire connecting **t** to the graph and select Delete Wire from the context menu. Now click on the icon with **sin** inside it and place it between **t** and the graph. Connect a wire from **t** to **sin** and from **sin** to the black port on the graph. Run the simulation and you'll see the sine function being plotted. If you don't see the sine wave shown in the screenshot, hit the square Stop button to reset the simulation and try again.

You can speed up or slow down the simulation using the slider at the top of the screen, or move it forward just one step at a time using the button to the left of the slider. You can zoom in and out using the mouse wheel and resize a graph by right-clicking on it and selecting Resize.

Let's now plot a second curve on the same graph. Drag the **cos** component and place it beneath the **sin** one, and wire it to the **t** component and to the red arrow on the graph. Run the simulation and you'll see that cosine has the same shape as the sine function, but shifted to the left.

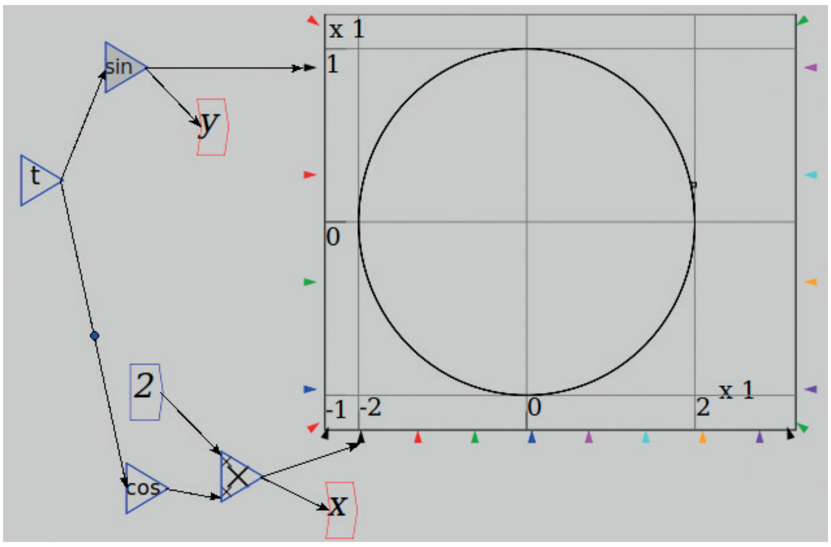
Going in circles

Now delete the wire from **cos** to the graph and move it underneath the graph, then connect the **cos** component to the black arrow at the bottom of the graph, as shown in the image above.

Before this example we didn't connect anything to the horizontal axis of the graph, so *Minsky* assumed we wanted to plot our function against time. But now we're telling *Minsky* to plot **sin** against **cos**. In other words, at each time step **t**, the y co-ordinate of the graph will be sin(t) and x co-ordinate will be cos(t). As you can see, the result is that we've made a circle.

Let's now combine components. Delete the wire from the **cos** component to the graph. Next, place the multiplication **(&times;)** component to the right of **cos**. Then place the component labelled **const** above **cos**. This represents a constant. Set its value to 2 in the window that appears, or you can do so via the Edit item in the context menu by right-clicking afterwards. Now drag wires from the constant, which will be labelled 2, and from cos, to the left-hand side of the multiplication symbol. Then join its tip to the black port at the bottom of the graph.

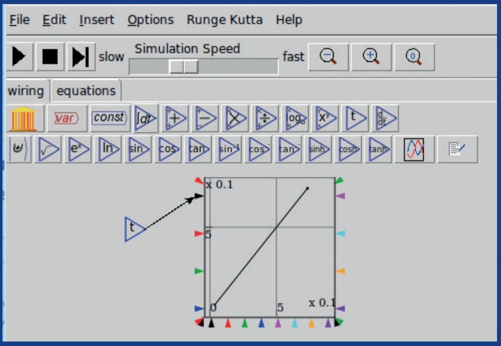
When you run the simulation you'll see a circle again, but notice that the horizontal scale is no longer -1 to 1, but -2 to 2. We now have an ellipse that's 4 units in width and 2 units high. If you're keen to see something on the graph that's less round, feel free to experiment. For example, move the 2 and multiplication constants before **cos** so that it receives two times **t** as its input.



Add a constant to our graph of sin aganst cos and you get an ellipse (really, it is, look closely at the axes!).

Simple line

- Click once on the graph icon, then click anywhere on the canvas to place it.
- Click on the triangular icon with a **t** inside it and place it to the left of the graph on the canvas.
- Drag from the tip of **t**'s triangle to the port (black arrow) on the left-hand side of the graph – this wires it to the graph.
- Click on the square Stop button to reset the simulation.
- Click the Play button underneath the menu bar.



Making equations

In the screenshot for the ellipse we've placed two red components containing x and y. These are variables that we placed using the **var** item in the component palette. These appear to be handy labels, but there's more to them than that: we can use them to output some equations like the ones shown.

Go to the File menu, select Output LaTeX and enter a filename which, by convention, should end in **.tex**. Let's call it **ellipse.tex**.

Now outside *Minsky*, either on the command line or using your favourite text editor, open up **ellipse.tex**. Inside are *LaTeX* commands for formatting equations. To see the equations themselves you'll need to process them within *LaTeX*. Open up a terminal, **cd** to the appropriate directory and on the command line type:

```
latex ellipse.tex
```

If the **latex** command isn't recognised then you'll need to install *LaTeX*. On Ubuntu you can do so as follows:

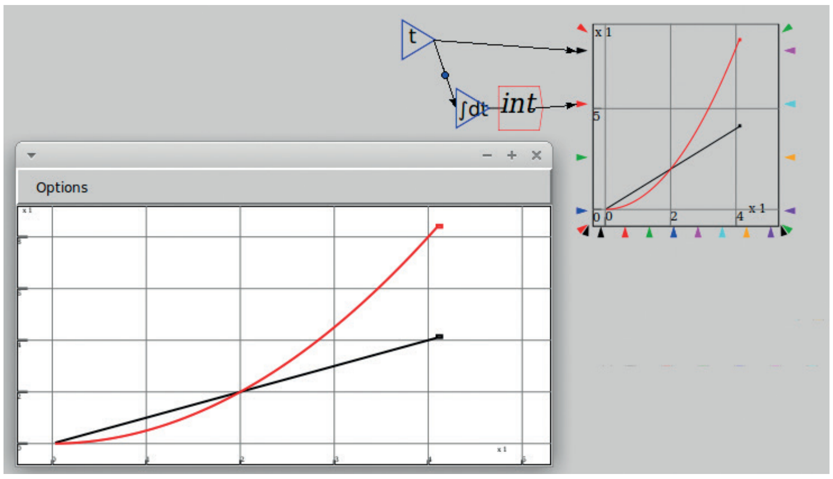
```
sudo apt-get install texlive-latex-base
```

The first time we ran the **latex** command we got an error about **breqn.sty** not being found. To work around this bug we had to go into the **Options->Preferences** menu in *Minsky* and check and then uncheck the 'Wrap Long Equations In LaTeX Export' option and then perform the export again.

The **latex** command will produce a file called **ellipse.dvi**, which should open if you click on it in your file manager – both *Evince* and *Okular* will open a DVI

PRO TIP

There is some innate complexity in modelling dynamic systems that no GUI can hide away. Also, as an evolving FOSS project that's grown from grants and a Kickstarter crowdfunding, *Minsky* is not complete nor is it bug-free. But despite the odd glitch and its rather basic appearance, *Minsky*'s core functionality has been well established and we found it surprisingly intuitive to use.



Integration: about all we remember from A-level maths.

file. If you'd like to turn it into a PDF file, do this:

```
dvips ellipse.dvi
```

Area under the graph

Let's go back to our first graph – the very exciting one that plotted time against time. Click the component that's to the right of **const**, the one that's got a strange flattened S symbol with **dt** next to it. If you hover the tool tip over it you'll notice it's called **integrate**. We'll explain why in a bit. Place the **integrate** component between **t** and the red port on the left-hand side of the graph and connect up the wires.

When you run the simulation you'll see the black straight line and a new red curve. Pause the simulation (press the P button again) after it has just passed 4 units of time. You may want to slow the simulation down. Now right-click on the graph and choose **Expand** and a window will appear with a clearer version of the graph with a finer grid.

The value of the red curve at any value of **t** is the area under the black line up to that value of **t**. Let's check this. At **t=4** the black line has value 4 on the vertical axis, and area under the graph is half of the square with corners at (0,0) and (4,4), ie half of **4 x 4 (= 16)**, which is 8. And the value of the red curve at **t=4** is indeed 8.

To integrate a function just means to calculate the area under the curve of the function plotted on a graph. It has many uses in practice, for example, if the black line were the speed of an accelerating car, then the red curve tells you the distance travelled since t=0. Alternatively, if the black line is the amount of money saved into a bank account each day, then the red curve would be the total amount saved.

PRO TIP

Minsky is named after economist Hyman Minsky and its creation was motivated by a desire by economist Prof. Steve Keen and the code's main author, Prof. Russell Standish, to open up the field of economic modelling.

Derivatives

Let's now use *Minsky* to take the derivative, a process known as differentiation. In this example we multiply **t** by itself to make **t** squared, then divide by 2 and plot the result of that as the red curve. But we also take the result and pass it through the **differentiate** component to produce the black line.

If you run the simulation you will find that it produces the same graph as before. This is not an accident. What we have demonstrated is that integrating the function **t** (black line) gives you **t<sup>2</sup>/2** (red curve), and differentiating that gives you **t**. In other words, differentiation is the reverse of the integration process.

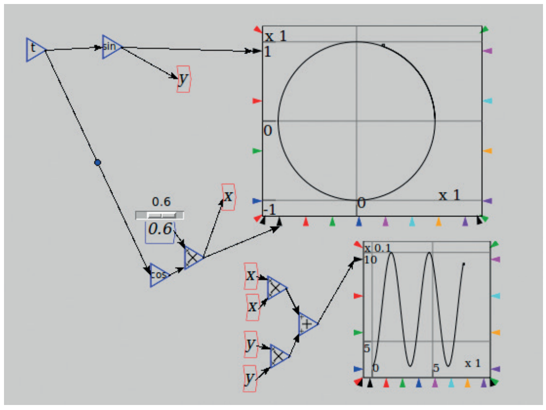
Integration and differentiation are the basic tools of calculus, much like addition and subtraction are the basic operators in arithmetic. They are needed in almost any situation in which we want to build a model of something changing over time (or space).

Constants and variables

We've already met constants: they are values that do not change during the run of a simulation, unless you want to intervene and alter them. Load up the ellipse simulation, then right-click on the 2 and choose Edit from the context menu. Set its value to 1 and the Slider Bounds to have Max 1 and Min 0. Next, right click, choose Slider and you will see a little slider appear above the constant. Start the simulation and reduce the value slightly with the slider and you'll see that the ellipse's width decreases. Sliders are handy for changing constant values on the fly.

We saw above that variables can be used to output *LaTeX* equations, but they have a much more important use. Again, starting with the ellipse simulation, place a new graph below the existing one. Right click on x and choose Copy and you'll be able to place a copy of **x**. Put it to the left of the new graph. Do the same for **y** and put it below the copy of **x**. Now place a plus symbol to their right and wire it up so that x+y is sent to the black port on the left-hand side of the graph. When you run the simulation you'll see that this new graph displays something like the sin or cos graphs we saw earlier.

Delete the wires from **x** and **y** to the plus component. Now create another **x** and another **y** and use times components to make **x** squared and **y** squared. Then wire them to the plus component as before. You should end up with what is shown in the screenshot below. Notice that we could have done this without variables by running four wires (two from

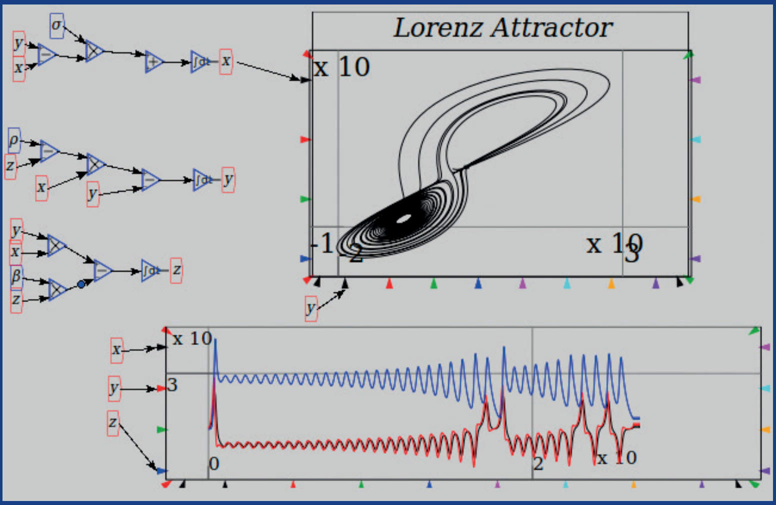


Constants and variables, shows the variation of x squared plus y squared in the bottom graph.

What is the Lorenz Attractor?

In the 1960s Edward Lorenz was using numerical models to describe motions of air in the atmosphere, but he soon realised they exhibited some surprising behaviour. This prompted a swell of interest from mathematicians to work on what is now known as chaos theory. Lorenz was the person who coined the term 'butterfly effect', referring to the fact that a small change in a chaotic system can lead to dramatic consequences in how it evolves: a butterfly flapping its wings (so the theory goes) could cause a hurricane at the other side of the world.

The state of the Lorenz system is described by three variables (x, y and z) and how it evolves from one time step to the next is determined by three equations, represented by the three main blocks in *Minsky* that end in x, y and z. These equations have three parameters represented by Greek letters: **ρ**, **σ** and **β**. Many values exhibit chaos, though the "classic" ones originally used by Lorenz are **ρ=28**, **σ=10** and **β=8/3**. The Lorenz attractor is the shape shown on the x-y plot. This is actually a 2D projection of it (or shadow), because the attractor is a 3D object (x, y and z).



The Lorenz Attractor was originally dreamed up to model convection.

sin, two from **cos**) down to the times components, but that would be messy and hard to read. Instead we define the x and y variables as outputs from **cos** and **sin** respectively, and use them when constructing an input to the bottom graph.

Now use the slider to put the constant back to a value of 1 and run the simulation. It may not seem tremendously exciting, but the graph will show a constant value of 1. If you slow the simulation down and watch the moving dot on the first graph you should be able to tell what's going on. The second graph is showing the distance of the current point on the first graph from the centre (0,0). For a circle this distance is equal to the radius, which in this case is 1. For an ellipse the distance to the centre varies with time. Try varying the slider as the simulation runs to verify this.

What we've shown here is that the distance of a point from the origin (0,0) is x squared plus y squared. In fact, we've used *Minsky* to illustrate a mathematical theorem originally stated by the Greek chap Pythagoras. We've also proved what's called a trigonometric identity: the square of **cos(t)** plus the square of **sin(t)** is equal to 1 for any value of **t**.

Let's bring this incidental tour of fundamental mathematics to an end for now and turn to generating some chaos.

The Lorenz Attractor

The Lorenz Attractor may sound like a long lost episode of Star Trek, but it's actually a feature of a famous chaotic system first described by Edward Lorenz. To produce it only involves the components we introduced above, but as it's a little more complex you might want to load up the file **lv8_lorenz.mky** via the GitHub link above.

If you load up the **lv8_lorenz.mky** file, the system has the "classic" parameter values mentioned in the boxout and starts with x=1 and y=z=0. When you run the simulation it will soon settle down into an orbit on the x-y plot, but notice that it's not periodic – it's not repeating the orbit exactly. You can see this also in the graph of all three parameters at the bottom. After some time the system will break out of the first orbit to the lower left and enter another orbit that's above and to the right. These two orbits gives the Lorenz attractor its distinctive figure-of-eight shape as shown in the image.

A parameter that determines chaotic behaviour is **ρ** (Rho) and if you reset the simulation and change its value to 10 then you'll see the system is no longer chaotic but spirals into the centre of the lower-left

We've used Minsky to illustrate a mathematical theorem originally stated by the Greek chap Pythagoras

orbit. If you set **ρ** to 350, you'll find that the system starts out appearing chaotic but eventually settles down into what appears to be periodic behaviour, ie repeating the same orbit.

And there's more

We've covered the basics of *Minsky* but haven't yet touched on its *raison d'être* – economic modelling. We'll get stuck into this in part 2, but in the meantime you can learn more about Minsky at **www.debtdeflation.com/blogs/minsky** and we recommend Prof. Steve Keen's video tutorials that you'll find there. The later ones do involve a bit of economic theory, but the first few will nicely complement what we've described in this article and show you a few more tips and tricks.

Andrew Conway watches the solar system, but also keeps a keen eye on Free Software and global macroeconomics.

CODE NINJA: MAKE A FILESYSTEM WITH FUSE

Combine Python and Fuse to build a new directory structure into your distro.

BEN EVERARD

WHY DO THIS?

- Understand the Fuse method for building filesystems
- Integrate your data with your operating system at a fundamental level
- Add another string to your Python bow

Most of the time, filesystems are data structures stored on some physical storage (such as a hard disk) that enable us to save and read data. That's actually not the whole story though, since files and directories are just a way for our computers to organise information for us to use. As well as the sort of disk filesystems we're used to, we can create filesystems that return any type of data to us.

Traditionally, filesystems were created by the kernel, but now we can use Filesystems in USErspace (Fuse) to write programs that can create filesystems from outside the kernel. In this tutorial, we're going to create a filesystem in Python. Our really simple filesystem will just include just one file, called **date**, and the contents of this file will be the current date.

First, make sure you have Fuse installed. In Ubuntu, this is done with:

```
sudo apt-get install fuse
```

Then you'll need to install the Python module we'll be using to create our filesystem:

```
sudo pip install fusepy
```

Now that we have everything we need, we can write the code. The majority of our code is taken up by a

> Our really simple filesystem will include just one file, and the contents of this file will be the current date

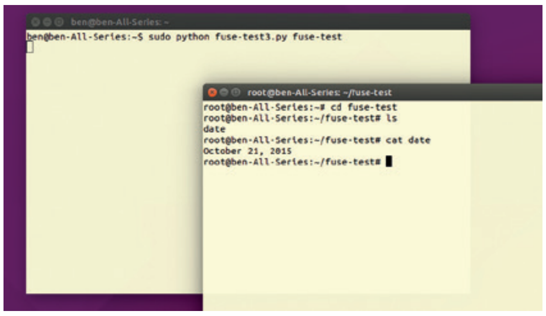
class that defines our filesystem. The outline for this class is:

```
class Context(LoggingMixIn, Operations):
  def getattr(self, path, fh=None):
    #code

  def read(self, path, size, offset, fh):
    #code

  def readdir(self, path, fh):
    #code

  access = None
  flush = None
  getxattr = None
```



```
ben@ben-All-Series:~$ sudo python fuse-test3.py fuse-test
```

```
root@ben-All-Series:~# cd fuse-test
root@ben-All-Series:~/fuse-test# ls
date
root@ben-All-Series:~/fuse-test# cat date
October 21, 2015
root@ben-All-Series:~/fuse-test# █
```

Our very own filesystem running and displaying the date.

```
  listxattr = None
  open = None
  opendir = None
  release = None
  releasedir = None
  statfs = None
```

As you can see, there are 12 operations that users could perform on the filesystem, although only three are relevant to our simple program. We've assigned all the others a value of **None** to avoid any problems if they're called by the user. The three operations we're interested in are **get attributes**, **read file** and **read directory**. Each of these methods will need fleshing out to return the right results when they're called.

Our attributes

First, let's take a look at **getattr**. The operating system will call this function when it needs the attributes of a file. It'll pass two pieces of information; the path and the file handle (we only use the path). The OS will expect this function to return a dictionary containing all the relevant attributes for the file. Our simple filesystem will only have two different paths: **/**, which is the root of the filesystem, and **/date**, which is the file containing the current date. Our code to process these is:

```
  def getattr(self, path, fh=None):
    if path == '/':
      attr = dict(st_mode=(S_IFDIR | 0755), st_nlink=2)
    elif path == '/date':
      attr = dict(st_mode=(S_IFREG | 0444), st_size=30)

    attr['st_ctime'] = attr['st_mtime'] = attr['st_atime'] =
time()
```

```
    return attr
```

Since **/** is a directory and **/data** is a file, they require slightly different attributes. They both need to have a mode which is calculated using the flags imported from the **stat** module and the number that corresponds to the Linux permissions for the file. They both also have a created time, modified time and an access time. For our filesystem, these aren't really relevant, so we've just set them to the current time.

The directory also needs an attribute with the number of hardlinks pointing to the directory. This, for a directory with no subdirectories, is 2. The file also needs a size. We've cheated a bit on this one and just hard coded in a size of 30, but it could vary depending on the actual date.

The second method we need is **read**. This will be called whenever the OS wants the content of a file. In our case, there's only one possible file, so we only need to check that that's the file being read and then return a string with the current date:

```
  def read(self, path, size, offset, fh):
    if path == '/date':
      return datetime.datetime.now().strftime("%B %d,
%Y") + '\n'
```

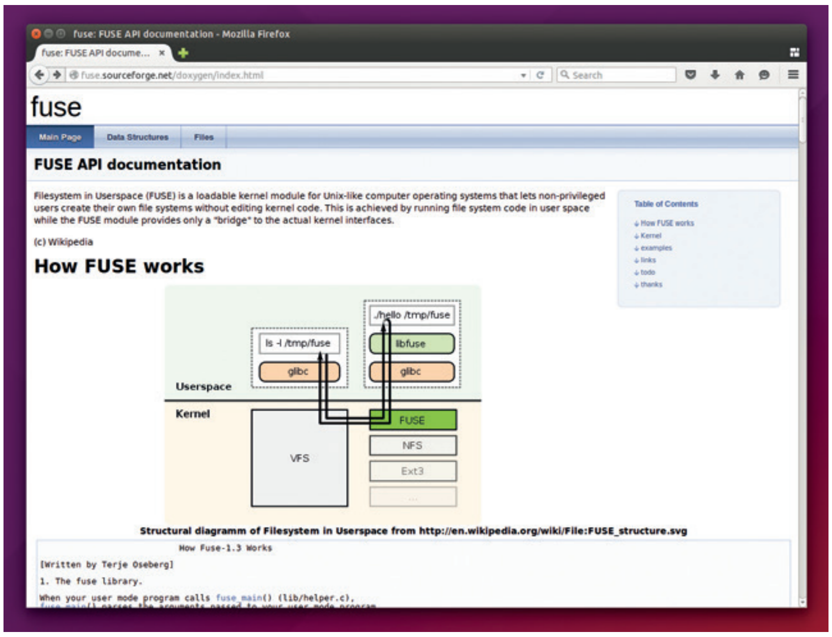
The final method is called whenever the OS wants the content of a directory. Again, we only have one directory, so all we do is return a list of the contents of a directory:

```
  def readdir(self, path, fh):
    return ['.', '..', 'date']
```

That's our main class complete. Now we just need the rest of the program to wrap this class up and launch the new filesystem.

```
from stat import S_IFDIR, S_IFREG
from sys import argv, exit
from time import time
from fuse import FUSE, Operations, LoggingMixIn
import datetime
class Context(LoggingMixIn, Operations):
  #code from above
if __name__ == '__main__':
  if len(argv) != 2:
    print('usage: %s <mountpoint>' % argv[0])
    exit(1)

  fuse = FUSE(Context(), argv[1], foreground=True,
ro=True)
```


fuse: FUSE API documentation - Mozilla Firefox

fuse.sourceforge.net/doxygen/index.html

fuse

Main Page | Data Structures | Files

FUSE API documentation

Filesystem in Userspace (FUSE) is a loadable kernel module for Unix-like computer operating systems that lets non-privileged users create their own file systems without editing kernel code. This is achieved by running file system code in user space while the FUSE module provides only a "bridge" to the actual kernel interfaces.

(c) Wikipedia

How FUSE works

Structural diagramm of Filesystem in Userspace from http://en.wikipedia.org/wiki/File:FUSE_structure.svg

```
                    How Fuse-1.3 Works
[Written by Terje Oseberg]
1. The fuse library.
When your user mode program calls fuse_main() (lib/helper.c),
```

The fusepy documentation is a little lacklustre, so if you need more information about what a particular operation does, check out the documentation for the main version of Fuse: **fuse.sourceforge.net**.

The first block imports all the modules we need. The line **if __name == '__main__:'** looks a little odd, but is a useful Python snippet for any code that can both be run from the command line and called from other pieces of code. The expression evaluates to True if the file is the main program being run. In our case, we use it to launch the Fuse filesystem if we're running this as a program, but also enables our Python file to be included as a module in other programs. The final line uses the imported **FUSE** function to launch the filesystem. The first two arguments are our new class and the location to mount the filesystem (this is taken from the argument passed across on the command line when the filesystem is launched. The others just set the standard filesystem parameters.

With all this code in place, you can launch the filesystem from the command line. The permissions needed to launch a filesystem vary from distro to distro. For testing purposes, it's easiest to run everything as root. You'll need two terminal sessions. In the first terminal session, get everything ready with:

```
mkdir fuse-test
sudo python fusedate.py fuse-test
```

In the second session, you can then navigate the new filesystem and read the current date:

```
sudo bash
cd fuse-test
cat date
```

That's all there is to creating filesystems. Obviously ours is very limited, but the basic techniques are exactly the same regardless of how many files or directories there are.

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

Fuse filesystems

Fuse isn't just for creating toy filesystems. It can also be really useful as it lowers the barrier to entry and makes it possible for non-kernel hackers to create new filesystems. This also makes it easier to distribute new filesystems as they don't require the user to compile them as kernel modules. Here are a few of our favourite:

- **SSHFS** Mount remote filesystems using just SSH with no other software required on the remote server.
- **EncFS** Create encrypted filesystems to keep your data safe.
- **Archivemount** Use compressed archives such as tarballs as though they were normal directories without unzipping them.

HASKELL: PROGRAMMING BACKWARDS

Rock-like reliability, a solid safety record in embedded systems, and a cool name.

JULIET KEMP

Haskell wasn't the first functional language, but it was the one that consolidated functional programming, and if you're looking for a pure functional language today, it's your best choice. Haskell has also become a bit more popular lately as functional ideas come into wider usage. It's quite different from imperative or OO languages, and, as with some of the other languages we've looked at, wrapping your head round it can be a challenge. (It doesn't, for example, have a **for** loop.) But it's fun to try out, and the excellent interactive interpreter makes it easy to experiment with.

History

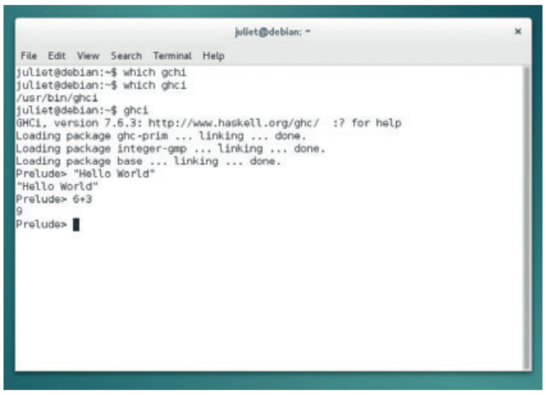
Lambda calculus, developed by Alonzo Church long before modern computers existed, is basically a way of thinking about functions and computability (we looked at Lambda functions in LV008's Code Ninja). Unsurprisingly, it was a major part of thinking about computer languages in the 1950s and 1960s. Lisp development owed quite a bit to lambda calculus, and Robin Milner used the same ideas when developing the functional language ML in the early 1970s.

There was quite a bit of interest in functional languages and lazy evaluation (evaluating an expression when it is needed and not before, which has the potential to massively reduce running time) at the time, but the first commercial lazy and purely functional language was Miranda, produced by David Turner at Research Software Ltd in 1985. Miranda programs consisted of a set of equations, defining functions and data types. As with Haskell, the order of the set was irrelevant, and indentation was used to minimise the need for brackets and avoid statement terminators (as in Python). Lists and tuples were important to Miranda, something else that made its way into Haskell.

The original plan for the Haskell committee was to use Miranda as a jumping-off point for the new language, but Turner politely declined the request, preferring to maintain Miranda as a single-dialect language. Haskell still owed a great deal to Miranda, but having to start from a blank page, while meaning a great deal more work, did give them more scope to make some potentially more radical decisions.

The first meetings of the committee, including the one where the name was decided upon (it is named after the logician Haskell Curry, but Haskell was felt to be a better and less pun-inducing name than Curry), were face-to-face, but after that the work was all done over email. Haskell 1.0 was defined in 1990, and improvements were made over the next seven years, finally producing Haskell 98 in the form of The Haskell 98 Report (all 150 + 89 pages of it). This consisted of a minimal core language and a standard library. Haskell is intended to be easy to extend and vary.

More recently, the borrowing of functional ideas into languages like Python and Ruby has made Haskell a more popular language outside of academia, and there's now an active coder community. Further improvements have also been made to the specification, with the most recent release, Haskell 2010, including bindings to other languages (the foreign function interface), and various extensions. There's an open-source library repository, Hackage, maintained by the community, and a useful wiki is also available from the Haskell webpage.



```
juliet@debian:~$ which gchi
juliet@debian:~$ which ghci
/usr/bin/ghci
juliet@debian:~$ ghci
GHCi, version 7.6.3: http://www.haskell.org/ghc/  :? for help
Loading package ghc-prim ... linking ... done.
Loading package integer-gmp ... linking ... done.
Loading package base ... linking ... done.
Prelude> "Hello World"
"Hello World"
Prelude> 6+3
9
Prelude> 
```

Experimenting with the interpreter **ghci**.

Getting started

Several distros offer packaged versions of the *Glasgow Haskell Compiler* (*GHC*) and the interactive interpreter. For Debian/Ubuntu, install **ghc**, and for other distros check out the Haskell website.

Once you've started **ghci**, the interactive interpreter, try a few expressions:

```
Prelude> "Hello world"
"Hello world"
Prelude> 6 + 3
9
```

Prelude is a standard module that's imported by default. It includes various functions including string functions, list functions, and basic I/O operations.

You can test a lot of expressions in the interpreter, and it's good for experimenting, but what if you want to write an actual function? You can't write functions directly in the interpreter; instead you need to create a source code file and load it in. Create a file **hello.hs**:

```
helloworld = print "Hello World"
```

Load and run it in *GHCI*:

```
Prelude> :load hello.hs
[1 of 1] Compiling Main    ( hello.hs, interpreted ) Ok,
modules loaded: Main.
*Main> helloworld
"Hello World"
```

So, you can define a function with just an equals sign, just like a variable. Just remember, once an assignment is made, you can't change it. Try this:

```
helloworld = print "Hello World"
helloworld = print "Hello World!"
```

Load and run that, and you'll get an error:

```
Prelude> :load hello.hs
[1 of 1] Compiling Main          ( hello.hs, interpreted )
hello.hs:2:1:
  Multiple declarations of `helloworld'
  Declared at: hello.hs:1:1
         hello.hs:2:1
Failed, modules loaded: none.
```

This applies to variables as well as functions (in fact, variables and functions are basically the same type of thing); see the boxout for more on code purity and functionality.

Tic Tac Toe

Let's try writing a tic-tac-toe program. This will be a very basic text-based input/output, rather than anything graphical, but it will show some aspects of Haskell. The first part will set up a list of the numbers of the squares, and output them tidily:

```
numberedSquareList = ["0", "1", "2", "3", "4", "5", "6", "7",
"8"]
```

The Haskell Working Group, Oxford, 1992.

```
printSquare input =
          putStr (" " ++ input ++ "")
outputLine lineList = do
          mapM_ printSquare lineList
  putStrLn " "
outputWholeThing list = do
          let (topOfSquare, restOfSquare) = splitAt 3 list
          let (middleOfSquare, bottomOfSquare) =
splitAt 3 restOfSquare
          outputLine topOfSquare
          outputLine middleOfSquare
          outputLine bottomOfSquare
main = do
  outputWholeThing numberedSquareList
```

- **numberedSquareList** will be used to give a number to each of the 9 squares in a standard tic-tac-toe board. The main function just outputs **numberedSquareList** as a three-by-three square, which is set up by the rest of the functions.
- **printSquare** takes a single input. It's possible to specify the type signature of a function, and we'll do this for a function later, but as a rule Haskell can guess it from your code. Here, the input is a string,

Haskell: functional and pure

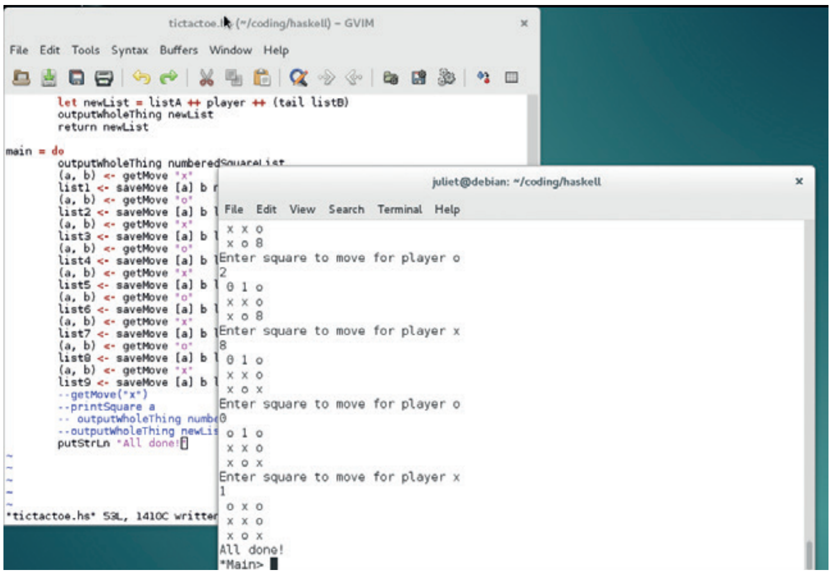
In an imperative language, you give the computer a sequence of actions to perform in a specific order. In a functional language, like Haskell, you give it instead a collection of expressions, so it knows what to compute, but not how or when to do it.

In order for this to work, it's important that functions should have no side-effects. That means that a functional expression must not change any part of the program state, and the result of a function must depend only on its input, and not on anything else happening elsewhere in the program.

This makes life a bit difficult if you want to do any input/output: I/O actions necessarily have side effects, as they interact with the outside world and can alter system or program state. To deal with this, Haskell divides code into 'pure' and 'IO'. Pure code has no side effects, and never alters state. Impure code (which includes system commands, modification of global variables, and I/O) may have side effects or alter state.

A corollary of all of this is that within pure code, variables mustn't vary, but remain the same once set. Otherwise the result of a function that refers to the variable foo might differ depending on whether or not foo changed at another point in the program.

Given no side effects and no changes to variables, the expressions in a program can be evaluated in any order. This supports Haskell's "lazy" approach: Haskell will evaluate an expression when and only when its result is needed. This doesn't matter, because the program has no moving parts; whenever you evaluate the expression, the result will be the same. Functional languages also make it easy to pass functions into one another, as well as sticking them together. The map function is an example: it takes a function and a list as parameters, and applies the function to every element of the list. Haskell functions can also return functions, as well as having them as parameters. These ideas all arise from lambda calculus, and will be familiar if you know any Lisp. Functional programming can be a bit of a challenge if you're familiar with imperative programming, but it has some real advantages for certain sorts of project. It entirely avoids a certain class of bugs, those which are due to unanticipated side effects; and makes testing easier. It also makes it possible to automatically parallelise the pure parts of your code (recent versions of **ghc** will do this for you), as side effects are one of the big issues with parallelised code.



```
tictactoe.hs (~/coding/haskell) - GVIM
        let newList = listA ++ player ++ (tail listB)
        outputWholeThing newList
        return newList

main = do
        outputWholeThing numberedSquareList
        (a, b) <- getMove "x"
        list1 <- saveMove [a] b r
        (a, b) <- getMove "o"
        list2 <- saveMove [a] b l
        (a, b) <- getMove "x"
        list3 <- saveMove [a] b l
        (a, b) <- getMove "o"
        list4 <- saveMove [a] b l
        (a, b) <- getMove "x"
        list5 <- saveMove [a] b l
        (a, b) <- getMove "o"
        list6 <- saveMove [a] b l
        (a, b) <- getMove "x"
        list7 <- saveMove [a] b l
        (a, b) <- getMove "o"
        list8 <- saveMove [a] b l
        (a, b) <- getMove "x"
        list9 <- saveMove [a] b l
        --getMove("x")
        --printSquare a
        --  outputWholeThing numbe
        --outputWholeThing newLis
        putStrLn "All done!"
~
~
~
~
~
"tictactoe.hs" 53L, 1410C writter
```

```
juliet@debian: ~/coding/haskell
 x x o
 x o 8
Enter square to move for player o
2
 0 1 o
 x x o
 x o 8
Enter square to move for player x
8
 0 1 o
 x x o
 x o x
Enter square to move for player o
0
 o 1 o
 x x o
 x o x
Enter square to move for player x
1
 o x o
 x x o
 x o x
All done!
*Main>
```

It's a draw, as tic-tac-toe tends to be when both players know what they're doing.

and the function outputs it to the screen with a space on each side.

- **outputLine** takes a list, and applies **printSquare** to each member of the list. **mapM_** and **mapM** are the functions that handle applying functions to lists, and they're really useful. The syntax, as shown here, is **mapM function list**
- **mapM** also collects and outputs the return value of the function as it is repeatedly applied. Here, we're not really interested in the return value, so we use **mapM_**, which discards the return value. **mapM** and **mapM_** deal with monads, whereas **map** does not; see the boxout for more on monads.
- **outputWholeThing** takes a list and outputs it three elements at a time. The **splitAt** function does what you might expect: it splits a list at the given element (note that lists in Haskell are indexed from zero). So first we split the list into the first three elements and the rest of the list, then we split the rest of the list again into the first three elements and the remainder. (This doesn't check for errors, like a list that is the wrong size; it just assumes that we're getting in a 9-element list.) Then **outputLine** outputs each section of the list, creating our tic-tac-toe grid.

Now let's try to get a player's move, and then save it:

```
getMove player = do
  putStrLn ("Enter square to move for player " ++ player)
  square <- getLine
  return (player, read square :: Int)
saveMove player square = do
  let (listA, listB) = splitAt square numberedSquareList
  let numberedSquareList = listA ++ player ++ (tail listB)
  outputWholeThing numberedSquareList
main = do
  outputWholeThing numberedSquareList
  (a, b) <- getMove "x"
  saveMove [a] b
  outputWholeThing numberedSquareList
```

getMove uses **getLine** (self-explanatory) to get the square for the move from the user. This will be read in as a string, so when returning it, we use **read** to translate it into an **Int**. (Note that **read** doesn't do any error-checking; you could look into using reads.)

saveMove uses a couple of useful list functions. **splitAt** does what you'd expect: it splits the given list at the given index (with that index starting the second list). We then stick the two lists back together, adding the **player** value (which will be X or O, in tic-tac-toe) between them, and dropping the first value of the second list. (The function **tail list** returns all but the first value of a list.) Since we split the list at the index point, this effectively creates a list that has the new move in the place where the index number used to be. So if player X chose square 6, the list now has X instead of 6. We then output the whole thing to show the player what the board now looks like. At this stage, we're only getting player X to play, and only once.

You might notice that sometimes we use **let x = y**, and sometimes we use **x <- y**. The former is used for 'pure' code, and the latter for I/O (or other impure) code. If in doubt, experiment, and the compiler will tell you if you've got it wrong.

This all looks good, but if you run it, you'll find that the final output no longer has the x in the 'saved' position. This is because Haskell is a pure language; you can't reassign variables once they've been assigned. In **saveMove**, you're not actually replacing **numberedSquareList**. You're creating a new local variable, also called **numberedSquareList**, which only exists for the lifetime of that particular function. Once we return to **main**, the local variable disappears, and the original **numberedSquareList** hasn't changed.

One way to get around this is to get **saveMove** to return a value, and keep creating new lists:

```
saveMove :: [String] -> Int -> [String] -> IO [String]
saveMove player square oldList = do
  let (listA, listB) = splitAt square oldList
  let newList = listA ++ player ++ (tail listB)
  outputWholeThing newList
  return newList
main = do
  outputWholeThing numberedSquareList
  (a, b) <- getMove "x"
  list1 <- saveMove [a] b numberedSquareList
  (a, b) <- getMove "o"
  list2 <- saveMove [a] b list1
  (a, b) <- getMove "x"
  list3 <- saveMove [a] b list2
  (a, b) <- getMove "o"
  list4 <- saveMove [a] b list3
  (a, b) <- getMove "x"
  list5 <- saveMove [a] b list4
  (a, b) <- getMove "o"
  list6 <- saveMove [a] b list5
  (a, b) <- getMove "x"
  list7 <- saveMove [a] b list6
  (a, b) <- getMove "o"
  list8 <- saveMove [a] b list7
  (a, b) <- getMove "x"
  list9 <- saveMove [a] b list8
  putStrLn "All done!"
```

Monads

Monads are part of functional programming, and enable Haskell to maintain its pure/not pure division by isolating parts of code that might have side-effects (like IO code). They enable you to guarantee that certain calculations are done in a particular order; as discussed above, this isn't usually the case in Haskell. Monadic actions enable you to pass elements and results from one function to the next in a way that is illegitimate in pure code. They also contain a way of turning a type into a monad type – the IO monad can turn a String into an IO String, for example – again, allowing the separation between pure and 'real-world-acting' code.

IO is a common monad, as is Maybe, which we don't use in this tutorial but which is useful for computations that could fail. There are many, many explanations of monads online, and different people understand them in different ways. You can't get all that far in Haskell without them, but once you come to grips with them, they're incredibly useful.

saveMove now takes another argument: a list to act on. It also returns a list. We've also added a type signature at the top to make it more maintainable (this is good practice to do for all your functions, although not necessary).

```
[String] -> Int -> [String] -> IO [String]
```

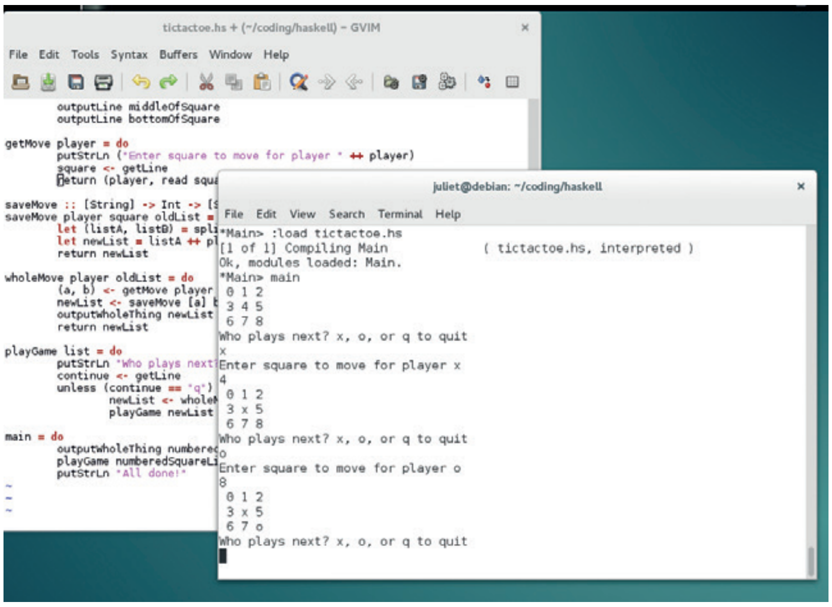
means that the function takes a String array, an Int, and another String array, and outputs an IO String array. If you're struggling to work out a type signature, you can use **:type functionname** in the interpreter and it will tell you what it thinks the type is.

In main, we repeatedly call **saveMove** on the current list, then use the list it returns as the input the next time. This also means we're getting moves from the X and O players alternately, which is handy. Note that the last statement in a **do** block must be an expression, so we need that last **putStrLn** line.

The downsides are, firstly, that it's rather untidy, and secondly, that there's no way of cutting a game short if someone wins. Haskell doesn't really do iteration (though there is a way of iterating over lists), but a very common Haskell idiom is recursion. Let's try a recursive approach to our game:

```
import Control.Monad
saveMove
  -- as before, but delete outputWholeThing line
wholeMove player oldList = do
  (a, b) <- getMove player
  newList <- saveMove [a] b oldList
  outputWholeThing newList
  return newList
playGame list = do
  putStrLn "Who plays next? x, o, or q to quit"
  continue <- getLine
  unless (continue == "q")
    newList <- wholeMove continue list
    playGame newList
main = do
  outputWholeThing numberedSquareList
  playGame numberedSquareList
  putStrLn "All done!"
```

Control.Monad contains some useful functions to use with monads, including **unless** and **when**. **wholeMove** is just a helper function to get and save a specific move; there's no new code.


tictactoe.hs + (~/coding/haskell) - GVIM
File Edit Tools Syntax Buffers Window Help
juliet@debian: ~/coding/haskell
File Edit View Search Terminal Help

```
        outputLine middleOfSquare
        outputLine bottomOfSquare

getMove player = do
        putStrLn ("Enter square to move for player " ++ player)
        square <- getLine
        return (player, read squa

saveMove :: [String] -> Int -> [S
saveMove player square oldList =
        let (listA, listB) = spli
        let newList = listA ++ pl
        return newList

wholeMove player oldList = do
        (a, b) <- getMove player
        newList <- saveMove [a] b
        outputWholeThing newList
        return newList

playGame list = do
        putStrLn "Who plays next?
        continue <- getLine
        unless (continue == "q")
                newList <- wholeM
                playGame newList

main = do
        outputWholeThing numbered
        playGame numberedSquareLi
        putStrLn "All done!"
~
~
~
```

```
*Main> :load tictactoe.hs
[1 of 1] Compiling Main             ( tictactoe.hs, interpreted )
Ok, modules loaded: Main.
*Main> main
 0 1 2
 3 4 5
 6 7 8
Who plays next? x, o, or q to quit
x
Enter square to move for player x
4
 0 1 2
 3 x 5
 6 7 8
Who plays next? x, o, or q to quit
o
Enter square to move for player o
8
 0 1 2
 3 x 5
 6 7 o
Who plays next? x, o, or q to quit
```

Playing the game! Still very basic though...

playGame is the clever bit. It takes a list as an argument: this is the current state of the game. First we ask which player has the next move (and offer the chance to quit), and get the answer. If the answer is **q**, the function ends. Otherwise (**unless q**), we perform the next move, get a new list out again, and then call **playGame** again on the new list, which has the new state of the game. We keep going around until the user types **q** at the prompt, passing the new state back into the method each time.

main now outputs the initial (blank, numbered) grid, then passes that into **playGame** to start the recursion. When the user answers **q**, we jump back to **main**, and output "All done".

There are a few ways you could improve on this code: You could look into the **State** monad functions to find other ways of passing state around.

Currently, you can keep playing even once all the squares are blank; you'll just overwrite them. You could add something to stop the game once that happens. Similarly, the user has to decide who has won; could you find a way of checking for that?

More fundamentally, this code isn't super-Haskell-y in that it could probably have a better separation of pure and IO code. Improving that would be a great way of finding out more about how monads and type signatures work.

If you want to get stuck into improving this code or writing your own, there are plenty of online resources available. Try *Learn You A Haskell For Greater Good* (free online, or in print) for a bunch of great tutorials. The Haskell wiki is a good reference, and there's a thorough Haskell book on Wikibooks. *Real World Haskell* is also available online. Have fun!

Juliet Kemp is a friendly polymath, and is the author of Apress's *Linux System Administration Recipes*.

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.

Code compilation

Join us for a fantastic voyage trip to the internals of a process in which plain English words are melt into executable machine codes.

Computers speak machine language. Humans usually don't. Machine code is just too primitive, too low-level for our brains, which are used to higher-level abstractions. When we design a house, we decide on materials, the number of rooms, and which colour the ceiling will be, not how the bricks will stick together. The same applies to most programs we write.

Except for specific system stuff, our software uses high-level programming languages. They are great for programmers, but all Greek to computers. So, what we need is some way to translate these languages into machine code.

This is basically what compilers and interpreters are all about. Today, we'll see how this conversion occurs in various situations. Consider a trivial C program:

```
int main()
{
  return 0;
}
```

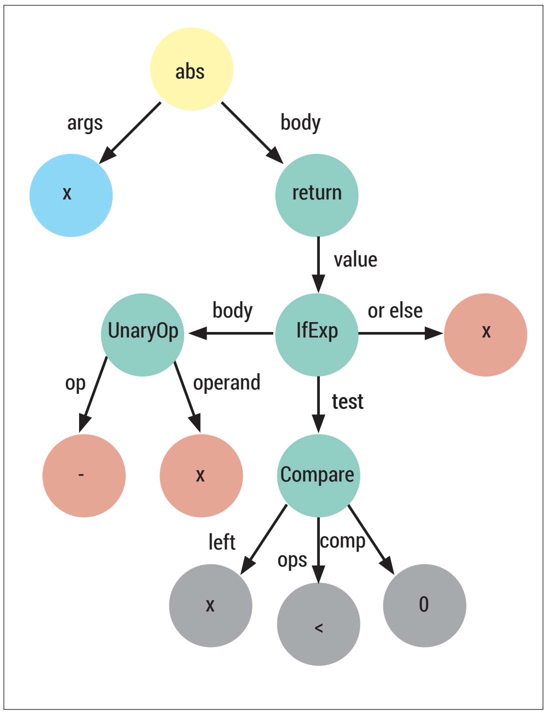


Figure 1. Simplified AST tree for **abs(x)** function as seen by the Python compiler.

How to run it? Unless you are very new to Linux (welcome!), the answer is straightforward:

```
$ gcc -o trivial trivial.c
$ ./trivial
```

gcc is the GNU C Compiler, and it is part of *GCC*, which stands for GNU Compiler Collection (still loving your recursive acronyms, yeah?). Essentially, this command transforms C code into machine instructions and packs them in an ELF executable (LV018). This is sometimes called Ahead-of-time Compilation or AOT, because the program is built prior to execution.

The **gcc** command is really a shortcut for whole pipeline of things. First, the lexer recognises the tokens (like keywords or variable identifiers) that your code is made of. Tokens form syntactic constructions (say, loops or function definitions) that the parser recognises. If the parser comes across something it doesn't understand (for instance, two tokens that don't fit together, like **if** and **for**), you get a compilation error. Otherwise, an Abstract Syntax Tree or AST is built in the compiler's memory (see Figure 1), which is a program's representation that's not tied to input language syntax.

AST is well suited for semantic analysis and, in particular, optimisation. Optimisation is a tricky topic, and although **gcc** provides command-line switches to fine-tune individual optimisations, most often you just set the desired optimisation level with **-O<something>**.

At the next pipeline stage, the compiler walks through the optimised tree and emits native machine instructions for your program. Usually, this step is invisible, but you can instruct **gcc** to stop here to see the assembler; just use the **-S** switch. This is what **trivial**'s **main()** function looks like:

```
main:
        pushq     %rbp
        movq      %rsp, %rbp
        movl      $0, %eax
        popq      %rbp
        ret
```

Compiled code units are saved as object files, which conventionally carry the **.o** suffix.

Finally, object files are combined together in one executable, or shared library. This is the linking stage, and in fact it isn't part of the compiler. A separate program called a linker (**ld**, in the case of *GCC*) resolves external references (like the **extern** variables or library functions) and lays out everything to produce a valid ELF binary. Or, it can produce something different, as a thing named a "linker script" dictates. Practically, you don't write linker scripts (they are rather low-level) or call **ld** directly. Everything happens behind the **gcc** curtains.

Ahead-of-time (or simply traditional) compilation has several benefits. As it runs "offline" on a build farm and not in real time on an end-user device, it can involve deeper and more time-consuming optimisations. Together with native code generation this yields a more efficient binary. However, it would necessarily be system-specific (or non-portable), and you can't just copy an ELF image from your x86 PC to an ARM smartphone and hope it will work properly. This could be a problem, and if platform independence is a priority, another approach might be helpful.

In the meantime

The trick is not to target any specific processor architecture or operating system during the code generation phase. Instead, the compiler emits instructions of a virtual processor, often called "bytecode". The problem is that virtual processors don't exist in silicon, so you need to implement them in software. This is the approach usually taken by interpreters and language virtual machines (VMs). Some languages (such as Python) bundle the compiler and virtual machine together, while others (Java) keep them separate.

Bytecode doesn't need to be as low-level as real machine instructions. For instance, the Java virtual machine has an instruction to get an array's length, something that isn't readily available even in C. Python implements an instruction to print a string or setup the **with** block. Being able to design an instruction set for a language makes the compiler simpler. And of course, generated bytecode can run on any platform, provided the latter has a virtual machine available.

But there are also some downsides. In practice, you can't ditch platform-specific code altogether, as you need some way to interact with the environment you run in. Interpreted languages are also significantly slower than compiled ones, as virtual machines have measurable overhead.

To see is to believe, so let's have a look at the bytecode of one popular interpreted language. You guessed it, Python. Consider a simple function that returns an absolute value of its sole argument:

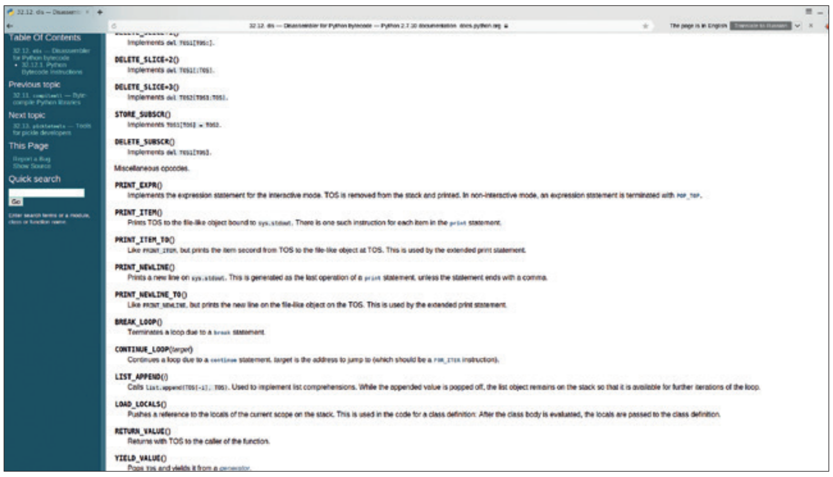
```
>>> def abs(x):
...     return -x if x < 0 else x
```

The Python Standard Library provides the **dis** module, which is a disassembler for Python bytecode. Note it is naturally implementation-specific, and if you use anything other than CPython, the command below may not work for you:

```
>>> dis.dis(abs)
 2          0 LOAD_FAST              0 (x)
            3 LOAD_CONST             1 (0)
            6 COMPARE_OP             0 (<)
            9 POP_JUMP_IF_FALSE     17
           12 LOAD_FAST              0 (x)
           15 UNARY_NEGATIVE
           16 RETURN_VALUE
      >>   17 LOAD_FAST              0 (x)
           20 RETURN_VALUE
```

abs() translates to nine bytecode instructions. Numbers in the first column denote lines of source code. CPython's virtual machine is stack-based, and it has no registers as real processors. This hurts performance a bit, but allows for a simpler design.

First, the function pushes the **x** value and **0** constant on to the stack. Then the **COMPARE_OP**


32.12. dis — Disassembler for Python bytecode — Python 2.7.10 documentation docs.python.org

Implements del TOS1[TOS:].

DELETE_SLICE+2()
Implements del TOS1[:TOS].

DELETE_SLICE+3()
Implements del TOS2[TOS1:TOS].

STORE_SUBSCR()
Implements TOS1[TOS] = TOS2.

DELETE_SUBSCR()
Implements del TOS1[TOS].

Miscellaneous opcodes.

PRINT_EXPR()
Implements the expression statement for the interactive mode. TOS is removed from the stack and printed. In non-interactive mode, an expression statement is terminated with POP_TOP.

PRINT_ITEM()
Prints TOS to the file-like object bound to sys.stdout. There is one such instruction for each item in the print statement.

PRINT_ITEM_TO()
Like PRINT_ITEM, but prints the item second from TOS to the file-like object at TOS. This is used by the extended print statement.

PRINT_NEWLINE()
Prints a new line on sys.stdout. This is generated as the last operation of a print statement, unless the statement ends with a comma.

PRINT_NEWLINE_TO()
Like PRINT_NEWLINE, but prints the new line on the file-like object on the TOS. This is used by the extended print statement.

BREAK_LOOP()
Terminates a loop due to a break statement.

CONTINUE_LOOP(target)
Continues a loop due to a continue statement. target is the address to jump to (which should be a FOR_ITER instruction).

LIST_APPEND(i)
Calls list.append(TOS[-i], TOS). Used to implement list comprehensions. While the appended value is popped off, the list object remains on the stack so that it is available for further iterations of the loop.

LOAD_LOCALS()
Pushes a reference to the locals of the current scope on the stack. This is used in the code for a class definition: After the class body is evaluated, the locals are passed to the class definition.

RETURN_VALUE()
Returns with TOS to the caller of the function.

YIELD_VALUE()
Pops TOS and yields it from a generator.

Figure 2: Python defines quite a few bytecode instructions. All of them are described in the **dis** reference manual.

> Ahead-of-time compilation can involve deeper and more time-consuming optimisation than just-in-time

Get to know ctypes

ctypes is a portable way to create and manipulate C language types from Python. It can call into shared libraries, and wrap Python functions so that libraries can call them back. **ctypes** is mainly useful to create bindings to C libraries. It isn't blazing fast, but requires zero C code. **ctypes** works on the ABI (Application Binary Interface) level, which is somewhat easier to break but doesn't involve any compilation steps. However, if you use it carelessly, you can crash the Python interpreter quite easily.

instruction pops both, compares them and pushes the result. **POP_JUMP_IF_FALSE** pops the result of comparison, and branches to instruction 17 (marked with a double arrow) if it is false. Here, the code again puts **x** on the stack and returns a value from the stack's top (ie, **x**) with **RETURN_VALUE**. Another branch works in a similar fashion. The documentation for the **dis** module lists all bytecodes known to CPython's VM. There are quite few of them, but still less than in a typical processor's instruction set.

Just in time

Conceptually, Just-In-Time compilation is like Ahead-Of-Time compilation, but there is one very important nuance. It happens "online", on the end-user device, often while the program is running. This poses some challenges, but if implemented properly, can also yield measurable benefits.

The main challenge is probably that the end-user device's processor is probably slower than that of the developer's machine or build farm, and is often battery-powered. Moreover, you don't carry a smartphone to build software on it, so the compilation process shouldn't be resource-intensive. This limits the amount of optimisation that the compiler can do, and the amount of code it compiles. JIT usually deals only with performance-critical application parts, and leaves the rest to the emulator. As a rule, JIT compiler also works on intermediate (bytecode) representation. It's simpler to translate than source code, and also offloads many things to the developer's machine, as in AOT. Sometimes the results of JIT are also cached on device for later re-use.

At the same time, JIT techniques enable targeted optimisation. The compiler knows exactly what CPU it runs on, and can potentially emit machine code for this particular processor. More importantly, the compiler knows how the program is being used, and can employ profile-guided optimisations. Say, if you barely use feature A, there's no point spending time and resources compiling it.

Perhaps the trickiest part is striking a balance between compilation costs and optimisation level. Again, there's no single solution. One way is to work on method or function level. The VM starts in interpreter mode and collects statistics on which methods are executed most often. Then it emits optimised machine code, so they could execute faster. Oracle's Java VM behaves this way. In fact, it's called HotSpot VM because it is all about detecting "hot spots" in your Java bytecode and optimising them properly.

Hot traces

Tracing JIT is an alternative approach. The idea is that programs spend most time in loops, or code that jumps to the same origin. These loops can span multiple methods (albeit they don't need to) and are dubbed "hot paths", as opposed to "hot spots" in method-level JIT. Internally, tracing the JIT compiler keeps a counter for each code location. Initially, the VM runs in monitor mode. It interprets bytecode and updates the counter each time a specific location is visited. When the counter appears to be above a threshold, a hot path is detected, so the VM switches to record mode. Then it carefully records all effects of bytecode execution until the code returns to the starting point.

Now, the VM has a "trace" of the new hot path. Instructions that can diverge from it (like branching) are protected with guards that quickly check that the assumptions under which the trace was taken are still true. Then the trace is compiled to native code. Next time the VM encounters this hot path, it executes compiled trace instead.

Mozilla's original TraceMonkey JavaScript engine and the PyPy Python language implementation are both examples of tracing JIT. However, there is no ultimate answer to which JIT flavour is the clear winner. PyPy delivers impressive results, while TraceMonkey was later superseded with combined JIT techniques. Results naturally depend on the languages you compile, and the environment.

Do it yourself

As a roundup, let's build a small JIT compiler for mathematical expressions. To keep things simple, we won't support variables or functions: just plain values and arithmetic operations.

We won't start from scratch. The Numba project (see boxout) maintains the **llvmlite** *LLVM* binding, which focuses on JIT compilation, and we'll use it today. We aren't going to use a dedicated parser, though. Instead, we'll employ the **ast** module to peek into the syntax tree generated by the Python compiler.

```
import ast
expr_str = '2+2'
ast_mod = ast.parse(expr_str)
expr = ast_mod.body[0].value
```

We start with the expression string and parse it into AST. Python delivers the result as a module containing

Python JIT for real

Besides our toy example, there are some real JIT compilers targeting the Python language. As you already know, PyPy (**www.pypy.org**) sports tracing JIT. Benchmarks look promising, and it's mature enough to run the Flask or Django web frameworks. Pyston (**www.pyston.org**) is a method-level JIT from Dropbox. Just like our tiny example, it starts with AST and relies on *LLVM* to produce fast code. The project is currently in alpha stage, but it would be curious to watch it progress.

The problem with doing effective JIT for Python and similar languages is in their dynamic nature. Variables may change their types, and the interpreter dispatches this at runtime. Objects may have dynamic attributes, and the interpreter looks them up in a dictionary. This is bad for machine code, where each value is typed, and the meaning of `a + b` is totally different for `a` and `b` being integers or strings. Sometimes, the compiler is able to infer types and generate effective machine code, but often it's difficult or impossible, and compiled code ends up being wasted.

a single expression (check it with **ast.dump(ast_mod)**), and we unwrap it.

The next step is to generate an *LLVM* intermediate representation (IR). The **llvmlite.ir** module provides all relevant functionality:

```
from llvmlite import ir
def create_ir_builder():
  fnty = ir.FunctionType(ir.DoubleType(), ())
  module = ir.Module(name=__file__)
  func = ir.Function(module, fnty, name="_main")

  block = func.append_basic_block()
  builder = ir.IRBuilder(block)
  return module, builder
```

Here, we create an IR module and define the **_main** function inside it. The function takes no arguments and returns double. Note how *LLVM* relies on types for its operations. **create_ir_builder()** returns a module and an IR builder we'll use later to emit IR instructions.

```
def emit_ir_for_ast(builder, node):
  if isinstance(node, ast.BinOp):
    left_ir = emit_ir_for_ast(builder, node.left)
    right_ir = emit_ir_for_ast(builder, node.right)
    if isinstance(node.op, ast.Add):
      return builder.fadd(left_ir, right_ir)
    # other operations follow
  elif isinstance(node, ast.Num):
    return ir.Constant(ir.DoubleType(), float(node.n))

prog, builder = create_ir_builder()
result = emit_ir_for_ast(builder, expr)
builder.ret(result)
```

This fragment walks AST in descent-recursive manner. It converts any number encountered to a double floating-point constant, and generates IR instructions for binary operations. **ast.Add** represents addition, and **fadd** is floating point addition in *LLVM* IR. Finally, we return the result of the top-level expression from **_main**. Operator precedence is handled automatically in the Python parser.

Then, the program calls into the **llvmlite.binding** layer to compile the IR into machine code. It's rather long and we won't show the details here; refer to comments in sources available at **ww.linuxvoice.com**. The **binding** functions accept IR source code as a string (you get it as **str(prog)**).

Finally, we use **ctypes** (see boxout) to call into the machine code we just generated:

```
from ctypes import CFUNCTYPE, c_double
func_ptr = engine.get_function_address("_main")
_main = CFUNCTYPE(c_double)(func_ptr)
print("_main() = %f" % _main())
```

ExecutionEngine.get_function_address() returns the pointer to the **_main()** function we just compiled.

To try this code yourself, you'll need to obtain **llvmlite** first. This could be tricky, so I suggest you use the *Miniconda* installer (**http://conda.pydata.org/miniconda.html**). It keeps everything in your home directory along with system-wide Python, and is great for trying new stuff. Download the installer script from the link above and simply run **conda install llvmlite** when done.

What does the **_main()** function look like at machine code level? Disassembly (**TargetMachine.emit_assembly()**) gives the answer. And there's another surprise: it's just three instructions long. *LLVM* is an optimising compiler, and as it detects that all operands in the expression are constant, it evaluates it compile-time. This is called "constant folding", and that's why optimised *LLVM* execution time (blue bars) doesn't depend on expression complexity. To make comparison fair, we can disable optimisations in *LLVM*. However, this doesn't change the results drastically.

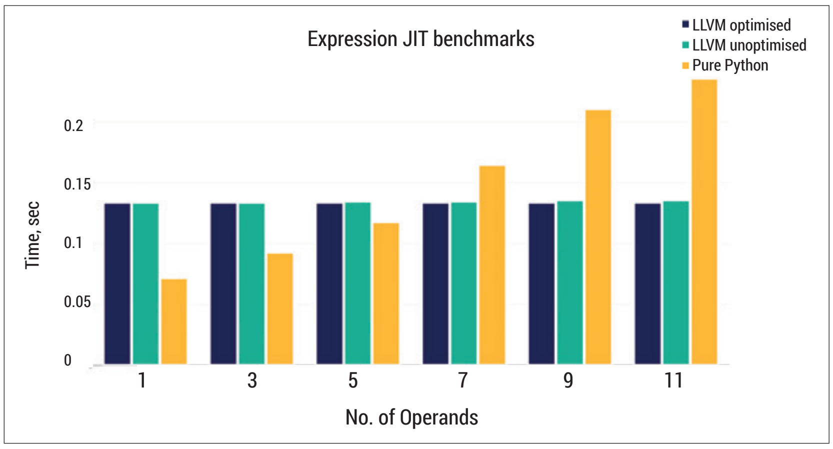


Figure 3: Execution time for varying expression sizes. Compiled code spends most of the time in **ctypes**, that's why the unoptimised version is only marginally slower.

Command of the month: pycc

Python's JIT is fun, but wouldn't it be nice to simply compile your script into static binary, as we do in C or C++? Well, it probably would, but **pycc** won't help you there. Instead, **pycc** compiles your Python functions into shared libraries (**.so**) that you can use in a language of your liking. **pycc** is also part of Numba (see boxout), and it uses the same *LLVM* machinery that the **@jit** decorator does.

Usage is straightforward: you tell **pycc** which Python sources to compile, and get a **.so** object. Note that any function you want compiled should be explicitly exported with **numba.export()**, specifying both arguments and the return type:

```
import numba
def add(x, y):
  return x + y
export('add i4(i4, i4)')(add)
```

Here, **i4** means 32-bit integer. Alternatively, you may ask **pycc** to output *LLVM* bytecode with **pycc --llvm**.

Numba advertises **pycc** as an experimental feature, and at the time of writing it had some known issues. Nevertheless, this tool looks rather promising.

/DEV/RANDOM/

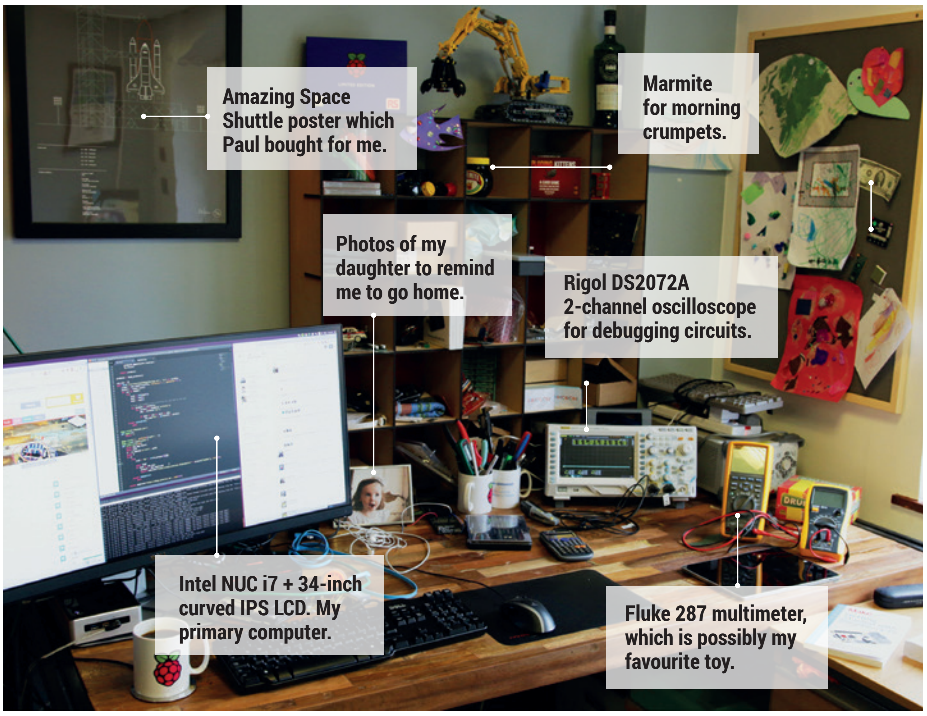
Final thoughts, musings and reflections

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!

Recently, the Linux Foundation promoted a new animation series "A World without Linux" (**http://goo.gl/H8UrrT**). The intention seems to be to remind people that they use Linux all the time – though, since only Linux types are going to know about it in the first place, I guess they are hoping that it will get tweeted or instagrammed or whatever it is you do on Facebook these days apart from dodge taxes. Tip: Omit the word 'Linux' from the link if you want non-Linux people to look at it.

It is a very reasonable thing to do, but I am not sure that having watched a few episodes everyone in the world is going to rush to download a distro. I went to a friend of a friend's birthday a few weeks back, and not wanting to go empty handed, I took him an Ubuntu T-shirt. When he opened it, I had to explain what it was – he uses Ubuntu because it is free and it works and he can do all the stuff he wants to on it. He probably never noticed the logo before.

People *do* use Linux every day, but they don't know, and, I suggest, they don't care. That's fine with me. For people who do care, it is more interesting to see the people who do embrace Linux and open source doing cool stuff – Netflix recently updated its GitHub repository and is doing a major push on its open source software (**http://goo.gl/it5wUH**). Why? Because more people using it (and finding bugs and fixing things) makes it better. The average viewer may be very slightly interested to know that without Linux their show wouldn't exist, but convincing non-open-source coders is probably more beneficial. And they might appreciate T-shirts more.


Amazing Space Shuttle poster which Paul bought for me.
Marmite for morning crumpets.
Photos of my daughter to remind me to go home.
Rigol DS2072A 2-channel oscilloscope for debugging circuits.
Intel NUC i7 + 34-inch curved IPS LCD. My primary computer.
Fluke 287 multimeter, which is possibly my favourite toy.

MY LINUX SETUP JON WILLIAMSON

Ninja-coder, product designer and co-founder of Pimoroni.

Q What version of Linux are you currently using?

A Ubuntu 15.04 (though I've just upgraded to 15.10 at home). All of our Pimoroni infrastructure is running on Ubuntu Server.

Q And what desktop are you using at the moment?

A Unity – it's great with a little tweaking. Matches my workflow well.

Q What was the first Linux setup you ever used?

A Yikes, that was a long time ago! It was a painful night spent installing Slackware in... err 2001? It took hours and wasn't a pleasant experience. I ended up moving to Debian (then naturally on to Ubuntu).

Q What Free Software/open source can't you live without?

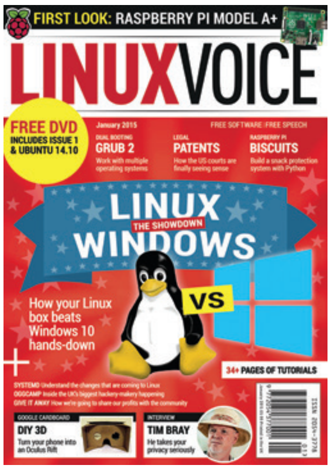
A All of them! It's amazing to be able to install something like **wkhtmltopdf** through your package manager and produce a PDF from a webpage then email it to someone all without leaving the terminal. Lots of great tools is where the power lies.

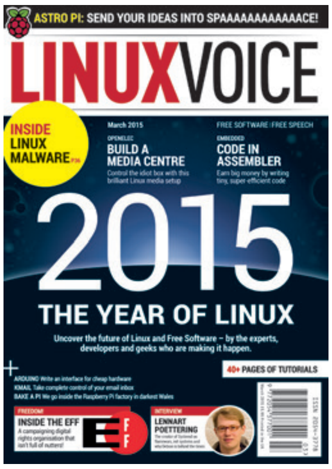
Q What do other people love but you can't get on with?

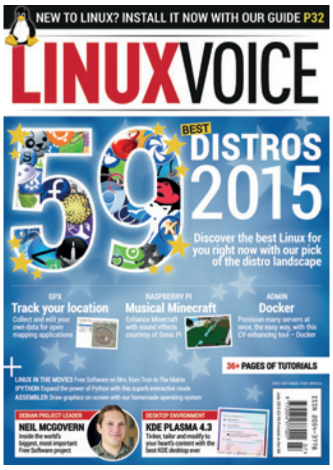
A *Vim/Emacs*. I'm a *Sublime Text 3* user and I while I wish it were open source I wouldn't change it for anything.

LINUXVOICE

This is what we've done in the last 12 issues. Subscribe to the next 12 from just £38.


FIRST LOOK: RASPBERRY PI MODEL A+
LINUXVOICE
LINUX THE SHOWDOWN WINDOWS
How your Linux box beats Windows 10 hands-down





FIREFOX: DISCOVER ITS HIDDEN FEATURES
LINUXVOICE
FREEDOM!
Switch to 100% Free Software today

Every subscription includes access to every PDF, ePub and audio edition we've ever published.

shop.linuxvoice.com

PIMORONI

http://pimoroni.com http://pimoroni.de

Join the pirates!
http://pimoroni.com/invest

For all things Pi <3

#CapitalAtRisk #InvestAware

We also need to tell you: Investments of this nature carry risks as well as potential rewards. Please #InvestAware as your capital is at risk.

Approved as a financial promotion by Crowdcube Ventures Ltd, which is authorised & regulated by the Financial Conduct Authority (No. 572026).