LINUX VOICE

The magazine that gives back to the Free Software community

May 2014

SCARY
KERNEL
Write your first module for the Linux kernel

RASPBERRY PI
ARCADE!
Build an arcade machine and live your 80s dreams

YE OLDE CODE
HOPPER
Learn from the creator of the first compiler

114 PAGES OF NEURAL ENHANCEMENT!

FREE SOFTWARE NEEDS YOU

Linux, Android, JavaScript – jump into coding now and make the world a better place

FREE SOFTWARE / FREE SPEECH

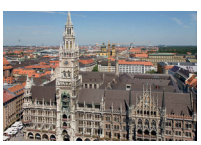
IT'S EASIER TO ASK FORGIVENESS THAN IT IS TO GET PERMISSION

+

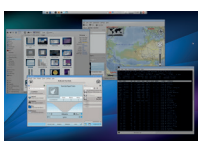
34+ PAGES OF TUTORIALS

UEFI Booting's Brave New World
TEXT EDITORS The Editor Wars rage on...
SOLYDXK The thinking geek's rolling release

ADOPTION
MUNICH
Inside the biggest Free Software migration yet



DESKTOP
KDE 4
Unleash the power of the tweaker's desktop



ISSN 2054-3778
May 2014 £5.99 Printed in the UK

Linux is awesome

The May 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 publicaton, 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 (you again).

THE LINUX VOICE TEAM

Editor Graham Morrison
graham@linuxvoice.com

Deputy editor Andrew Gregory
andrew@linuxvoice.com

Technical editor Ben Everard
ben@linuxvoice.com

Editor at large Mike Saunders
mike@linuxvoice.com

Creative director Stacey Black
stacey@linuxvoice.com

Malign puppetmaster Nick Veitch
nick@linuxvoice.com

Editorial contributors:
Mark Crutch, Liam Dawe, Juliet Kemp, John Lane, Vincent Mealing, Simon Phipps, Jonathan Roberts, Mayank Sharma, Valentine Sinitsyn

GRAHAM MORRISON

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

There's a lot of renewed emphasis on learning to code, and a great deal of noise being made about transforming the way computing is taught. This is unequivocally wonderful because to many of us, the only way to tackle such subjects is through Linux and Free Software. And it's working. At the recent OCR Raspberry Jamboree in Manchester, the many educators in attendance were talking in terms we've used to describe open source for years; collaboration, transparency, openness and being inclusive. But there was one term they used that I think we as a community often forget, and it's a word we try to put at the heart of this magazine; put simply, it's *fun*. Learning, for me, has always been much more effective when I'm enjoying myself. It's the fun element that has kept me sustained, and sees me through those twilit hours trying to figure out how PulseAudio works. Linux and Free Software is a serious business, a sober and essential cornerstone of computing. But that doesn't mean we can't enjoy every minute of it.

Graham Morrison
Editor, Linux Voice

SUBSCRIBE ON PAGE 34

What's hot in LV#002

ANDREW GREGORY

I'm already working on an easy solution to this month's coding challenge: prepare yourselves for the Gregorian fractal! **p96**

BEN EVERARD

KDE has always been a tough proposition. But our tutorial on getting the most from the KDE has made me try it again. **p86**

MIKE SAUNDERS

Discovering that Steve Ballmer was parachuted into Munich, and his subsequent hilarious encounter with its mayor. **p52**

CONTENTS

May LV002

We are all in the gutter, but some of us are looking at the stars

SUBSCRIBE ON PAGE 34

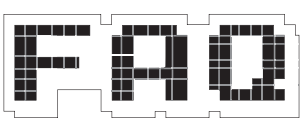
Get coding now!

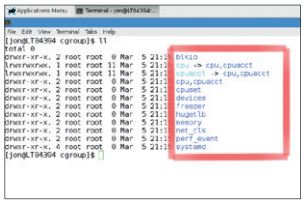
Android, JavaScript, Python – get stuck into programming.

MUNICH

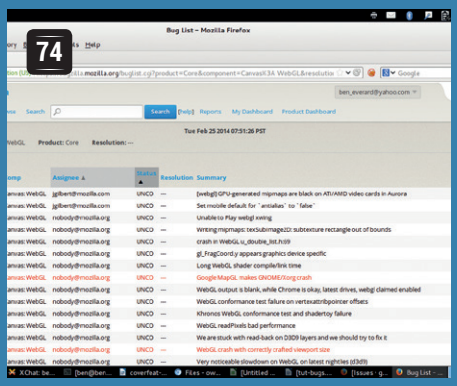
Find out what's behind the Bavarians' decade-long migration to Free Software

REGULARS



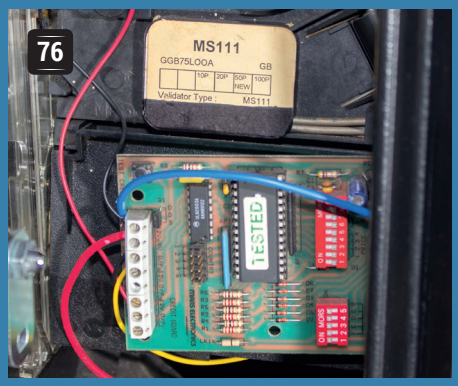


TUTORIALS



Bug reports: help make free software better

If you do it right, getting bugs fixed can be an important part of free software development.

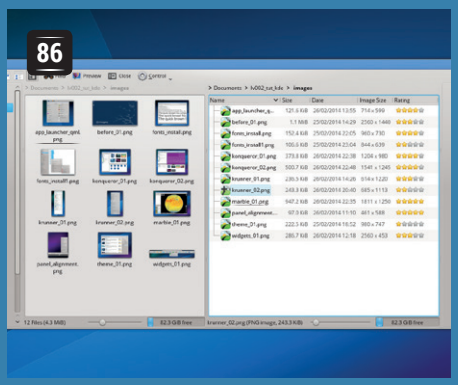


Raspberry Pi & MAME: build an arcade machine

Play the games of your youth* without squandering all your pocket money. *Graham's youth

Old code: Grace Hopper and UNIVAC

How to program on a machine that weighs more than a double-decker bus.

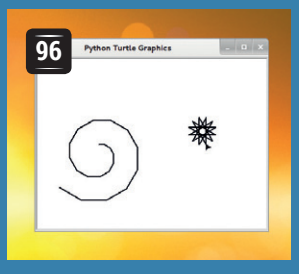


KDE: configure the hell out of your desktop

The defaults in KDE are an affront to taste and decency – so fix it and make your life better.

UEFI: The new world order of booting

Boot Linux without Grub or a BIOS.



Python: Draw patterns with recursion

Write better code with fractals.

100 Code 101: Secure key exchange

Lasers, maths and cryptography.

102 Linux: Write your first kernel module

It's not just for angry sweary men.

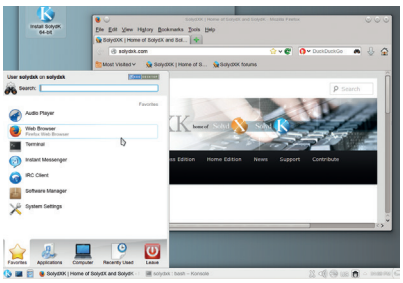
REVIEWS

20 Dell XPS 13 Looking for a new Linux machine, but don't want a heap of plastic rubbish? Step right this way!

22 Retrode Dust off your collection of games cartridges – the Retrode has arrived to imbue them with new life.

23 Krita 2.8 Now Windows types can use KDE's excellent image editor. That's great, but is there anything new in 2.8 for us?

24 PyBorg Your dream of controlling an underground robot city has just come one step closer to becoming reality.



25 SolydXK Based on Debian? Yawn. A third way between timed and rolling releases? Now we're interested…

26 Books Computing in the UK, then and now, plus a small dose of Arduino and a huge heap of Raspberry Pi.

NEWSANALYSIS

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

Opinion

Steering where you look

The seductive power of boundary conditions can ruin your business and community.

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

When I learned to drive, my instructor told me "you steer where you look" -- in other words, wherever you focus your attention becomes your destination, so keep your eyes on the road ahead and don't worry about the stores at the roadside (or even too much about the kerb and the parked vehicles).

The same principle seems applicable in other contexts. We're moving away from a hierarchical, post-industrial society and evolving into a meshed society of peers, interacting in variable roles on their own terms. That's challenging established institutions, but sadly they have frequently "steered where they looked" and made the wrong choices.

The big media lobby -- especially the movie and music industries, but also book, software and games publishers -- is right to be concerned about systematic infringement of their copyrights by commercial-scale criminals. But they have let their eyes wander.

They have looked so long in anger upon those crooks, invested so much time and money in frustrating them, that they've become fixated on copyright enforcement and forgotten to keep pace with the expectations of their customers.

They've let the market run away from them and failed to build new businesses around their fans and friends. Instead, fixating on copyright infringement, they alienate the very people who should be their best bet for the future by treating them as criminals. They may not go as far as chasing everyone with lawsuits, but the unskippable admonishments on DVDs and their like shout loud and clear: "We may have some of your money but we still don't trust you."

Open entrepreneurship

Open source projects are another case, perhaps a little more subtle. Entrepreneurs see the word "free" and assume there's a commons there to exploit for profit. But open source only has a commons once a community gathers -- it doesn't magically arrive the instant you apply an OSI-approved licence. Moreover, the commons exists only by the collective agreement of its participants to set aside certain rights so that collaboration becomes possible. Entrepreneurs tend to be so focussed on leveraging a free network effect that they overlook the actual mechanism that makes it happen.

An open source community is an example of a group of people choosing to synchronise a fragment of their mutual interests, each at their own expense, for the benefit of all involved including themselves. While there may sometimes be a non-profit organisation for administrative reasons, an open source community is inherently neither a non-profit or a for-profit organisation; profit is an orthogonal concept.

The process by which this distraction from core values happens is subtle, and undoing the error is hard. It probably happens incrementally as communities pass through the scales described by Dunbar Numbers [the theoretical number of people we are capable of forming direct relationships with without our brains melting, proposed by Robin Dunbar to be between 100 and 230]. They eventually mistake their boundary conditions for their core values.

At first a community is small enough for everyone to have a set of direct trust relationships. As the community grows through the Dunbar limit, it becomes necessary to define the norms for the community, to make it clear to newcomers what the values of the community are.

As growth continues, those norms become rules and the frequency with which they are enforced increases. Gradually, communicating the rules to outsiders becomes a common community function and the ability to do so becomes a community skill. Over time, application and explanation of those rules becomes so important to the community that they overshadow the original core values.

Influence, not control

There's no canned solution for this; it takes a brave executive to step away from the weapons and chart a course for influencing the meshed society instead of attempting to control it. The news that Getty Images is now allowing free embedding of their entire catalogue for non-commercial use is just such a bold, visionary move. I hope we will see much much more of that and many fewer anti-fan lawsuits and copyright-assignment-based open source projects.

"The media lobby alienate the people who should be their best bet by treating them as criminals."

CATCHUP Summarised: the eight biggest news stories from the last month

1 Linux has won. No, really.

According to technology analysts Gartner, Android has now overtaken iOS as the leading operating system for tablets. And this is massively significant – the same date suggests that in around 115m tablets were sold worldwide in 2012, compared with 200m in 2013, so in contrast to PC sales, tablets are growing quickly, and Linux is taking the majority of this very large pie. Android is estimated to have 62% of the tablet market, with iOS on 36% and Microsoft lagging behind them both with on just 2% market share.

2 Microsoft to follow strategy espoused by Linux Voice readers

We've only been in print for two issues, but already Linux Voice is having an effect at the top table of tech. We asked listeners to the Linux Voice podcast what advice they would give to the new CEO, Satya Nadella, and among the responses was the suggestion that the company should give Windows 8 away for free, to preserve the market for the lucrative Microsoft Office package. Lo and behold, it has come to pass, as the soothsayers of LV foresaw.

3 Mt GOX, the world's biggest Bitcoin exchange, has collapsed

The rabbit hole gets deeper for the most widely used cryptocurrency.

4 Broadcom releases the source code for Raspberry Pi graphics stack

Much as we love the Raspberry Pi, it isn't as free as in speech as it could be, largely because Eben Upton is a pragmatist and doesn't particularly want to make things easier for the Pi's profit-driven competitors. However, we're glad to see that Broadcom, maker of the Pi's graphics hardware, has released the source code for the Pi's graphics chipset under a BSD licence. Well done Broadcom!

www.raspberrypi.org/archives/6299

5 Google Chromium browser ported to run on Mir graphics server

The great schism at the heart of Linux graphics continues, as Ubuntu continues to go it alone with its Mir display server. It has now managed to get Chrome running on the server, which is its intended replacement for X. Meanwhile, most distros have elected to use Wayland as the replacement for X. X is such an old, ingrained standard that it is taking time to replace it, but the day will come, and either Wayland or Mir will be the new norm.

6 Linux Containers reaches its 1.0 release

Linux Containers have been bubbling away under the radar for a while now – discerning sysadmins can read our take on them on page 62 – and now they have reached their first major version number. Linux Containers can be thought of as being a bit like VirtualBox, except that rather that being completely virtualised, they use the existing kernel and just create their own userspace, making them handy for anyone who wants to be able to deploy lots of Linux instances easily.

7 Canonical unveils manufacturers for Ubuntu for Phones

Like buses, you wait ages for a phone manufacturer to team up with Canonical to bring Ubuntu for Phones to market, then two turn up at once. Spanish firm BQ (which also operates in Latin America) and Chinese firm Meizu have both signed up to produce handsets loaded with Canonical's swish mobile operating system. We've always liked Ubuntu for Phones, and we're delighted that there will soon be a physical product in the shops.

8 Calligra 2.8 released

KDE's office suite, Calligra, has reached its latest stable release. You can read all about our impressions of Krita, the suite's image editor, on page 23, but there's much more to the suite then that. There's now support for comments in Author and Words, the ability to open hyperlinks in the Kexi database app, and the usual bugfixes. We yield to no-one in our love of LibreOffice, but if you're running a KDE system you might want to give the Calligra suite a try as an alternative.

www.calligra-suite.org

DISTROHOPPER

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

Emmabuntüs

Linux for humanitarians.

Different situations need different distros. For many users it's important to having a solid base system that's around 700MB, making it easy to download. This is usually backed up by a great set of repositories that combine to give you access to almost all the software you need. Other times, it's really important to have everything you need installed from the start. Perhaps it's going to be installed somewhere without a good internet connection, or perhaps the users won't want to install software themselves. Emmabuntüs fits into this second category.

There's so much choice in Emmabuntüs, it can take some time just to get to know the applications.

For recycled hardware

It's designed for older computers for use by children. It has bucket loads of software in its 3.3GB download. Much of it will be familiar to most Linux users, but there's also a few unusual choices such as OOo4kids. This, you probably won't be surprised to learn, is a version of OpenOffice designed for children (**http://wiki.ooo4kids.org**).

Basically, it just has a simplified interface to make it a bit more friendly.

Xfce is a good choice for a desktop given the intended audience, and we like that it asks the user if they want a dock at the bottom, and if they want to install non-free software (Flash, etc). If you're looking at a distro for older computers to use in a school or community group, Emmabuntüs is definitely worth a look. Even if you decide to go with something else, it's sure to help you find a useful bit of software or two.

Picore

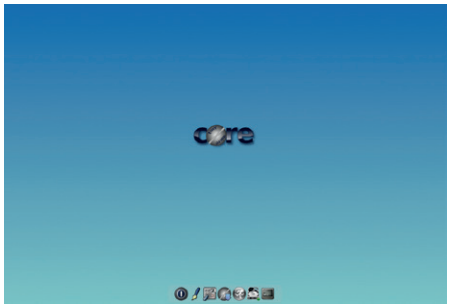
Possibly the fastest Linux distro on the Raspberry Pi.

TinyCore Linux is a technical marvel. It's a full Linux distro, including a graphical desktop, and it's only about 15MB. However, it's not the size of your distro that matters, it's what you do with it.

The small size of TinyCore allows it to boot in a slightly unusual way. Instead of loading files from disk each time something is needed, the entire system loads itself into RAM when the system boots. This results in an OS that's startlingly fast even on modest hardware. Modest hardware like, for example, the Raspberry Pi, and the TinyCore team have now released a build for the Pi called PiCore.

Compared to any of the other Linuxes for the Raspberry Pi, PiCore is startlingly quick. There's almost no lag when opening applications even on a non-overclocked Pi. It's a refreshing change from Raspbian, which can leave the user waiting at times.

It's hard to recommend PiCore to people new to Linux because its architecture does add a little complexity, and the RAM storage makes it a little too easy to lose all your files when you reboot if you're not careful.



There's not much software included by default in Picore, but plenty more can be downloaded from the internet.

However, experienced Linuxers looking for a little more snap on their (low-powered machine) should seriously consider it. After all, it's only 24MB to download and try.

Bodhi 3

Seek and ye shall find Enlightenment.

It's impossible to separate Bodhi and Enlightenment (a window manager). In fact, it's one of the only popular distros that's set up purely for Enlightenment, and so any new user is going to notice this desktop far more than the underlying distro.

Enlightenment has some real strong points. It's impressively fast for a desktop with so many graphical effects. We like how it looks, though some people may find the default setup a little gaudy with shimmers and glows everywhere. The biggest let down is the lack of Enlightenment-specific software. While both GTK and Qt have a large stash of programs that will fit in with the look and feel of a desktop, there's very little built specifically for Enlightenment, so most software looks out of place.

Underneath this, Bodhi is a solid distro based on Ubuntu LTS. Unfortunately, the Bodhi team have dropped support for ARM chips in this release. They found it simply too much work for too little gain, which is a disappointing, though understandable, decision. If you want to run Enlightenment, Bodhi represents the best choice for most users. This also means that the Bodhi live CD is the best way to try the desktop.

Beauty is in the eye of the installer, and plenty of people like the eye-candy of Enlightenment.

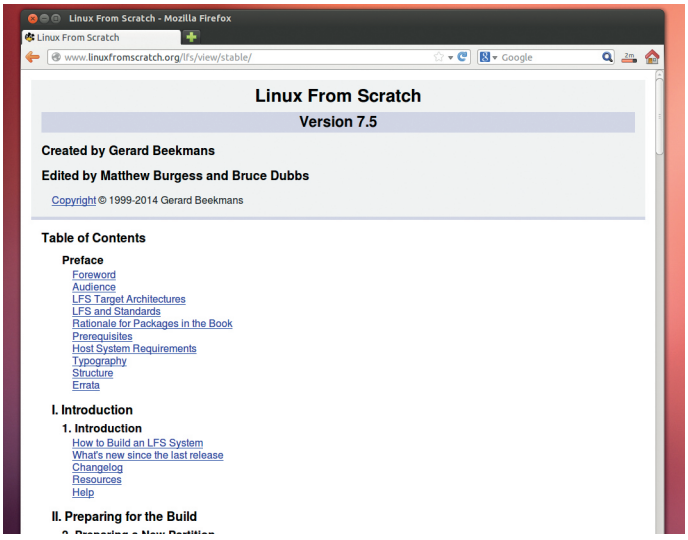
Linux From Scratch

Going beyond off-the-shelf Linux distros.

When most distros are released, you can go to the project's home page and download an ISO. Alternatively, you may be able to update a system that's already running. Neither of these is possible with Linux From Scratch (LFS) because, well, it's a book. You can read it online in HTML form, or download a PDF.

LFS isn't really a distro in the usual sense of the word. Instead it's a guide for creating a system that doesn't need a distro. After all, a distro is really just a collection of software, and that software is perfectly accessible on the upstream project's websites. So, instead of downloading a single image with all the things you need, you download them separately, and this book is a guide for combining everything into a working system. The start of March saw the release of version 7.5 of this book, which guides the user through the most recent versions of the upstream projects.

Different people have different takes on LFS, but the two most common opinions are either that it's the best way to learn about the internals of a Linux system, or that it's just a way to waste days compiling hundreds of pieces of software. Here at Linux Voice, we see the appeal of a system such as LFS, and it certainly has its place. However, most users wanting a distro they have control over will be better served by the likes of Gentoo and Arch. That said, if you ever want to know about what goes on underneath the various distros, LFS is an excellent place to start.



It's the ultimate geek quest, but compiling a full system takes time, especially on older hardware.

SystemD vs Upstart vs sysV init

When you turn your computer on, the bootloader loads the kernel into memory, then starts the init system. Traditionally, this init system has been a series of scripts that start everything running, and enables you to stop and start servers once the system is fully booted up.

These scripts (often called systemV init) kept Unix systems booting long before Linux was invented. However, two newer systems have been created: systemD and Upstart. These both offer more advanced features than the older scripts. Loosely speaking, Ubuntu and most derivatives use Upstart, while most other distros use systemD and a few still use systemV.

Debian, a slow mover in the distro world, had clung onto systemV up until the start of 2014. However, the technical committee was asked to vote on what direction Debian should go. In simple terms, systemD offered more functionality, while Upstart supports Debian flavours with other kernels (the distro can use the FreeBSD kernel, or GNU's Hurd). One sticking point was the fact that Canonical (which develops Upstart) requires all code contributors to sign a Contributor Licence Agreement (CLA) giving Canonical control over their contribution, including the power to build non-free software based on it.

In the end, systemD won after the committee chairman (Bdale Garbee) gave the casting vote. Following this, Mark Shuttleworth announced that Ubuntu would drop Upstart (in 14.10) leaving systemD as the init method on almost all distributions of Linux.

Not everyone is happy about this. Some users feel that systemD goes against the Unix spirit by being monolithic and logging in non-human-readable binary format. Since it doesn't play nicely with other Unix-like systems (such as the BSDs), anything that relies on systemD's features also won't work on these systems. There is some truth to these complaints, but unless anything changes, it will soon become hard to find a distro that doesn't use systemD.

GAMING ON LINUX

The tastiest brain candy to relax those tired neurons

VIRTUAL INSANITY

Liam Dawe is the brains behind www.gamingonlinux.com, the home of Linux gaming on the interweb.

Virtual Reality is a hot topic in gaming right now, and Linux isn't being left out in the cold. A company called Oculus VR has created a headset named the Oculus Rift – sounds cool, right? It's a strange-looking device that looks like it should be in Star Trek.

The good thing about the Oculus Rift is that it has Linux drivers ready, so it fully supports our platform. Alongside the driver support from Oculus VR we also have Valve; the company behind Steam. Valve is pushing out a Virtual Reality API for game developers, which also supports Linux.

The Oculus Rift is a VR device that sits over your eyes strapping you into your game, and that's where my problem is. Gaming is supposed to be a social thing, isn't it? Being strapped into some device feels so lonely. Not only that, but it's quite restricting not to be able to see anything else.

The other problem I have is that a game is supposed to be a game – why do people now want to emulate real life to the point of strapping things over their eyes? This is why I'm hoping that it's a fad, like 3D televisions, 3D movies and the 3D mode on Nintendo's latest hand-held console, which is more of a pain than anything else. Oculus VR will need to solve issues like eye-strain and resolution before it's taken seriously.

So, to end on a question again for you lot: will Virtual Reality take-over our gaming lives, or will it fade away into oblivion? You decide!

Liam Dawe, gamingonlinux.com

Project Zomboid

Craft your perfect hideout, wait for all this to blow over.

Brains braaaains braaaiiins! The game devs at The Indie Stone have graced us not only with a major patch for the hit Project Zomboid, but have also introduced tons of new features, including the first version of their its persistent-multiplayer. So, now you can go raiding supermarkets for tins of food while your friends on another Linux PC stand watch for zombies outside with a baseball bat.

Anyone is free to run a server for it, as The Indie Stone is giving away the server tools (be warned though that it's pretty rough), though it is fairly easy to get running on a Linux box.

Surviving the zombie apocalypse has never been so isometric, grrr arrrgh!

The player models have also been overhauled, so they have more animations, and the game looks fantastic.

When complete, Project Zomboid will have a sandbox survival mode, a single-player story, split screen co-op and online multiplayer.

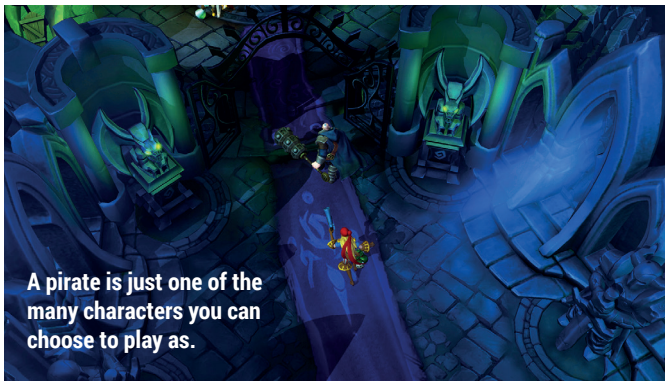
http://projectzomboid.com

Strife

A 'second generation' multiplayer online battle arena from S2 Games.

S2 Games shouldn't be a new name to people who have been following Linux gaming for a while – the company has just given out a whopping load of beta invite for its new multiplayer online battle arena (or MOBA for short) called Strife.

I was lucky enough to be invited and delighted to report that Strife is already fantastic. It aims to be a more minimalistic MOBA when compared to games like Dota2. For example, a lot of things are hidden from view to begin with, although you can pin different information to the screen to suit your playing style.



A pirate is just one of the many characters you can choose to play as.

Strife plays with the MOBA experience by introducing a pet system. Pets have their own unique abilities, and can be levelled up using food that you acquire after games. There's also a crafting system, so you can create your own items to use in-game and gain the advantage on your opponents.

This really is one to look out for, especially given S2 Games' history of great Linux support.

http://strife.com

Planetary Annihilation

That's no moon! No wait, it is a moon – and it's coming this way!

So, you have looked up at the moon and thought to yourself "I wonder what it would look like if it smashed into the earth"? Well you don't need to wonder about that anymore.

The developers of Planetary Annihilation have now included the ability to build massive space drives on moons for you to send them off-orbit and smash into other heavenly bodies.

They haven't stopped there either: they have also only just included their own version of a Minimap too, which is essentially the game running again in a small window on your screen, which uses up a lot of memory of course, but it does look crazy!

This is easily the most intense real-time strategy you can find on Linux, taking inspiration from classics like Total Annihilation and Supreme Commander.

If you are missing your fix of games like those or even Starcraft, we highly recommend checking this one out – just don't blame us= if you get addicted.

Planetary Annihilation was funded on Kickstarter to the tune of $2,229,344.

The game is still a little bit on the pricy side right now ($29) but they plan to drop the price once they hit the 'stable' version.

www.uberent.com/pa

Starbound

Ever seen a tree with big scary eye-balls for leaves? No? Play this then.

So, many, pixels! Starbound is sort of like Minecraft only in 2D and with far more to do. This game is an absolute gem, not only because one of the developers uses Linux, but because of just how fun it is.

You can go from one planet that has big scary eyeballs for leaves, to a planet that has pools of lava everywhere and pirates chasing you (no really, that happened to me once).

Nothing is more satisfying than taking down a massive UFO Boss that warps down little penguins (aww) with bazookas (eek!), though it does seem a little barbaric killing tiny pixellated penguins on a platform that has a penguin as its mascot!

http://playstarbound.com

The Swapper

You may or may not find talking rocks, it gets a little freaky.

Only recently released in the latest Humble Indie Bundle 11, The Swapper promises a very atmospheric puzzle-platformer experience.

When we read-up that everything in the game was made from real-life materials we didn't realise just how good that could actually look. From an escape-pod that looks like a can of beans, to freaky talking rocks that look like sponges, the visuals are completely original.

The game has some very unique mechanics too: you gain a gun that creates clones of yourself, which you can swap control with to solve puzzles.

http://store.steampowered.com/app/231160

ALSO RELEASED...

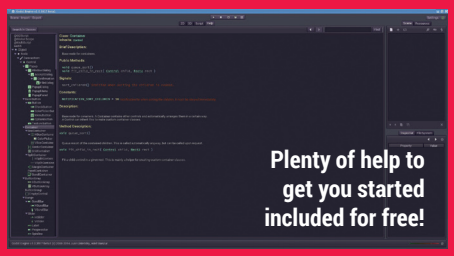
Mutant Gangland

Micromanegement got you down? Need some flesh pounding against metal? Enter Mutant Gangland, a very simple-looking turn-based strategy game that pits Mutants against Robots in close-qaurters combat.

The game includes a level editor for high replayability, and also throws in the source code for players to create mods with.

The team have recently released the first alpha version and we will be surprised if this doesn't pick up some fans.

http://mutantgangland.com



Godot Game Engine

Are you an aspiring game developer? Been looking to make a game, but you don't know where to begin? Godot has you covered!

This commercial-grade game-creation kit has been open-sourced under the MIT licence to give other engines a run for their money. It has quite literally everything you need in one package to begin creating.

www.godotengine.org/wp

SimpleScreenRecorder

My absolute favourite screen-recorder software, fully featured and extremely easy to use. I use this software myself to record games as I have found it to be the only software that can perfectly sync audio and video together.

If you need more to go on, it's also open-source and free, so I want to see more of you showing us your best play-throughs on the Linux Voice forum!

www.maartenbaert.be/simplescreenrecorder

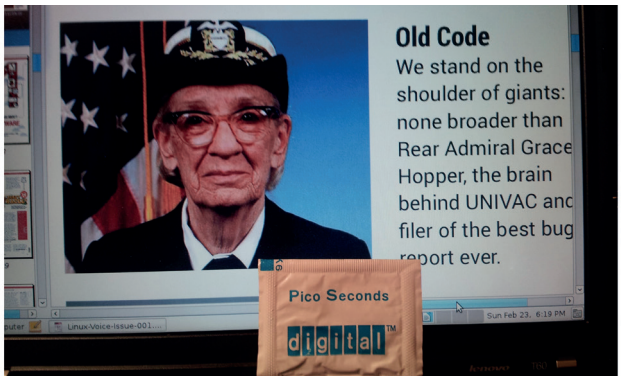
LINUX VOICE YOUR LETTERS

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

LINUX VOICE STAR LETTER

GRACE HOPPER

Congratulations on publishing the first of many great Linux Voice editions! I was excited to see a picture of Rear Admiral Grace Hopper for an upcoming article for next month. In my first job out of college (1981), I learned the basic "rules" of programming from a phenomenal designer/programmer, Gordon Vikse, who continues to teach me things to this day. He introduced me to the work of some of the greats in computing, such as Niklaus Wirth, C.A.R. Hoare, Donald Knuth and Grace Hopper. Years later, I worked as a civilian contractor at the Pentagon. One day while walking one of the literally 17 miles of hallways, I discovered a display on Grace Hopper, which greatly pleased me as it was a richly deserved honor. Shortly thereafter, I was even more pleased to be able to attend one of her talks (around 1990). At the time, she had been forced into retirement by the U.S. Navy and was working as a consultant for Digital Equipment Corporation (DEC). During the talk, she told the story of her nanosecond wires, then handed out picoseconds to the audience to demonstrate how much faster computers had become. The picoseconds were a packet of ground black pepper with the DEC logo and the words "Pico Seconds" printed on it. I only wish I could have had her sign mine. Her lesson that it is easier to ask forgiveness than permission is, in my opinion, embodied in the spirit of FOSS computing. I'm glad to see that Linux Voice is reminding us of the greatness that has existed, and continues to exist, in computing.
Paul Olson, Oklahoma, USA



It's only fair that we acknowledge, in Paul's words, "the greatness that has existed, and continues to exist" in computer science.

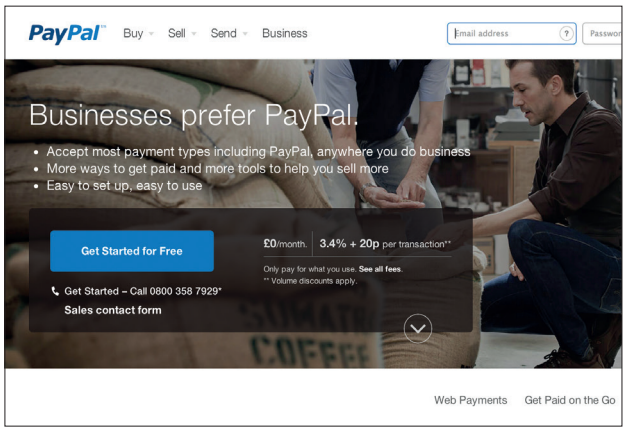
Andrew says: Paul, you've made my day. The Grace Hopper tutorial that we're running on page 80 isn't particularly Linuxy, but without Rear Admiral Hopper's work we might not have Linux at all.The debt that we owe the likes of Grace Hopper, Alan Turing (he's coming next issue), Donald Knuth *et al* is enormous.

PAYPONG

I've been an enthusiastic reader of your work for many years and have had a subscription to your previous magazine, for several years, bought for me by a family member; as a pensioner I need all the help I can get!

In trying to decide what to do about migrating to your new magazine, I discovered that it is impossible to do so without opening a PayPal account. I know that this is not a welcome situation, as far as my 'benefactor' is concerned. Is there any way around this obstacle? Whatever the outcome, good luck with your new venture.
Norman Kearey

Andrew says: We've spoken to our bank about a better online payment system, and they aren't interested in helping us until we have at least a year's accounts under our belt, so, distasteful as it is, we're stuck with PayPal and its exorbitant fees for now. We're not happy about it either.



As much as we dislike it, PayPal is the only way for us to accept payments for magazine subscriptions at the moment.

PUBLISHING

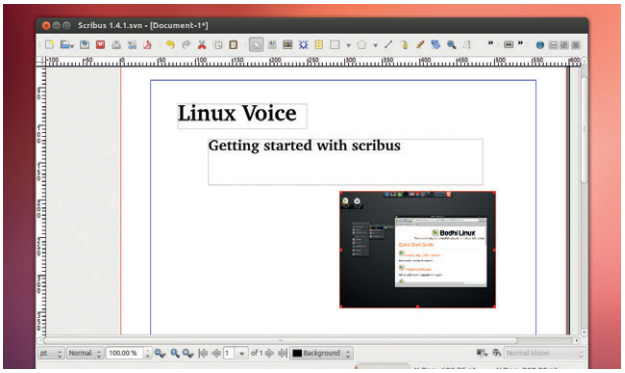
Thoroughly enjoying your first issue. What caught my eye at first glance was the news that a new release of Scribus is on the way. Great! I have no doubt that this program will be well able to compete with the extremely expensive equivalents in the Windows and the Mac platforms.

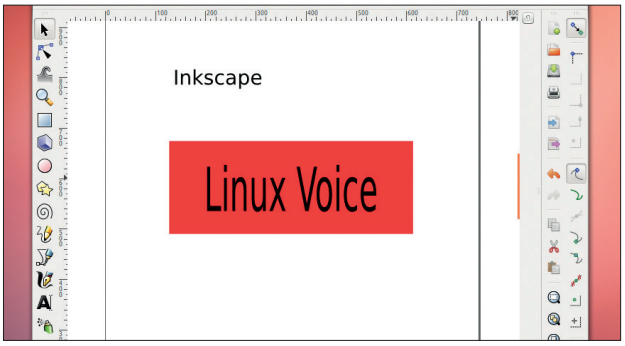
But there's a price to pay for this complexity. In earlier times Scribus was easy to use and gave me all the flexibility I needed. Now, unfortunately, I can't perform even the simplest DTP tasks. Everything seems to work differently and nothing can be done intuitively. There's a complete guide to Scribus in book form but it runs to 438 pages!

Which brings me to my main point: there's an excellent opportunity here for developers to construct a lower-level DTP program which is usable by ordinary human beings rather than high-powered professionals. Years ago there was 'Publisher', an elementary Windows DTP program which appealed to a different user than 'PageMaker'. So I would urge open source entrepreneurs to get their act together and fill the gap.

Or maybe I'm missing a simpler alternative in Linux ? Of course any word processor can design basic documents and Inkscape can produce good materials for publication without too much effort. But many people need something more versatile.

Maurice George, Ormskirk, Lancashire





Scribus (top) can be overkill for most tasks, so we'd stick with LibreOffice or Inkscape for simple jobs.

Mike says: I quite agree. The power to change, for example, the spacing between individual letters is awesome when you're doing high-end design work, but it's Scribus and its like are massive overkill for most people. Maybe all it would take is a version of Scribus with features taken out?

THE CODE AHEAD

Big fan of all your work. I've been listening to the TuxRadar... I mean Linux Voice podcast for a while and just got the issue 1 PDF. The content is great.

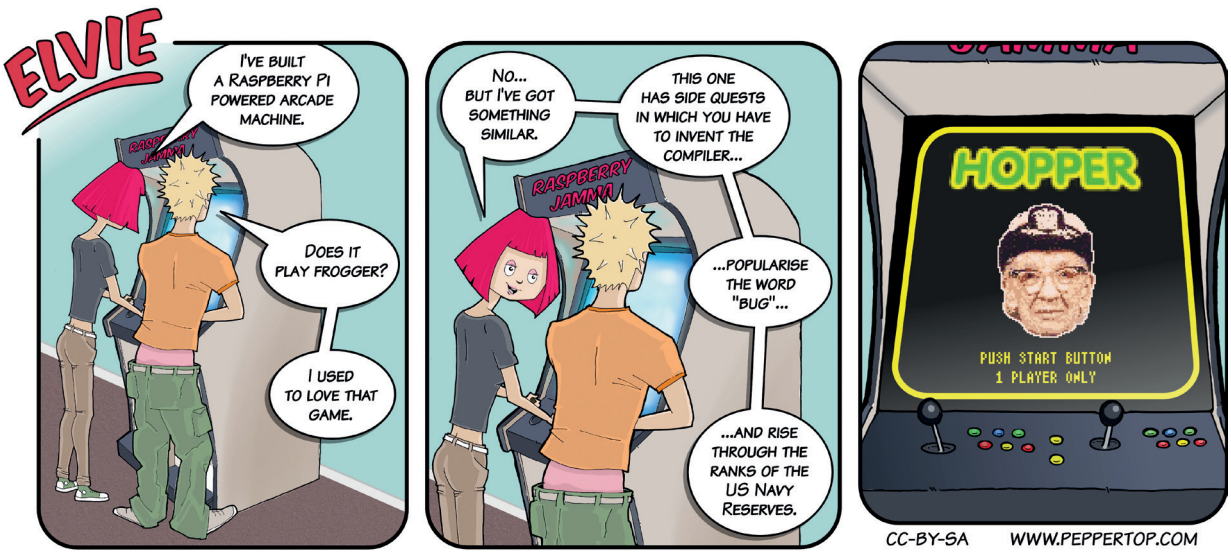
You might already be working on these but here's the two things I'd love to see:

- Have the table of contents and cover be made of clickable links to those stories. That will avoid "this story about bitcoin looks interesting, now let me scroll through 56 pages to get to it".
- An epub version.

Mark

Graham says: Thanks for the feedback Mark, we appreciate it. We know that we want to improve the digital offering from PDF-only; at the moment we're looking into a HTML 5 container app among other solutions, and epub might be one of those that we settle on. There's a rift between XML and HTML advocates at Linux Voice Mansions, and this many influence the formats that we choose. However, it's one of our principles that we should offer the content in whatever medium the readers want to buy, just as we offer the podcast in MP3, Ogg and Opus files.

Clickable links in the contents page though is something that we should have got right from day one, and if I can remember how to do it we'll have it in issue 2.

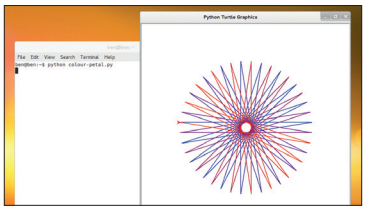


HEALTHY COMPETITION

Congratulations on an amazing first issue!
I could not believe when going through LV01, how much it is everything I have ever wanted from a Linux magazine:

- Amazing features.
- A section dedicated to gaming.
- Arch.
- A challenge after a tutorial.
- My name in it :-).
- Many more!

I have a print subscription and will be with you for the many, many years this magazine will no doubt be around for.
Craig Waites



In a happy coincidence there's another competition for you on p96.

Ben says: It's often easier to learn when you have a tangible incentive to do so. That's why we're offering exclusing Linux Voice T-shirts to the winners of our coding competitions – though of course, if you're learning something, you're already winning.

REDRUM REDRUM REDRUM REDRUM

I understand that you must be really busy these days, but you know what they say: "All work and no play makes Jack a dull boy..."

So I have made a Bash game called "Back in a minute" that may offer some escape. It's a simple text-based adventure game featuring four playable races, six enemies, turn-based fighting, eight hidden items and six different scenarios in the 270 sections of the world map. You can also create your own custom ASCII map for use in the game!

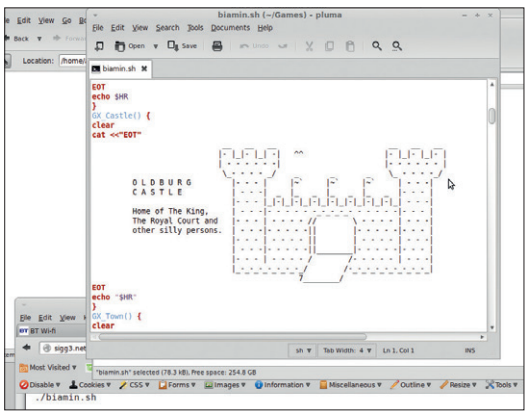
I'm currently doing an MA in political philosophy, so it is nice to "switch off" every once in a while and play around in Bash a little.

You can find the "Back in a minute" website at **http://sigg3.net/biamin** and browse the code directly at **https://gitorious.org/back-in-a-minute**

Thanks for providing me with GNU/Linux knowledge and tips for these last 7-8 years, and keep up the good work! Even though my studies and social life permit little time for LUGs and other F/OSS related events, listening to your podcasts makes up for it and also keeps me up to date.
Sigge, Norway

Graham says: Thanks Sigge, we'll show this to Ben as soon as we release him from the coding tower, where he's been devising this month's competition. And thanks the some light relief; LV Mansions is starting to look like the Overlook Hotel.

On second thoughts, let's not play Back In A Minute. 'Tis a silly game.



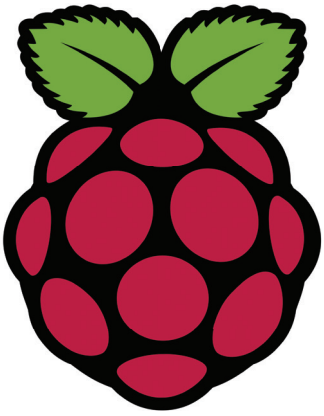
PIE TIME

First off, congrats on a great premier issue! Ah, most of the old band is back together again and sounding fine! Now if you just had a physicist on board, and perhaps a female computer language researcher, and... "Linux Voice" is perhaps the best start-up investment I have ever made.

With all the interest in the Raspberry Pi, here is a suggestion that I have been struggling with: setting up a true real-time version of Linux on the Pi. I have looked at Open Embedded, Preempt-RT, Xenomai and more, and they all seem promising but I haven't found a good complete setup yet. I am trying to sample the Pi's I/O pins deterministically at about 10~20 kHz to debounce and filter a 1kHz input, so evenly spaced sampling times without latency is important.

Congratulations again from the rural parts of New York State (yes, there is more to NY than NYC, like 500 km of farmland all the way to Canada) and I look forward to reading you for years to come!
Jim Cranston, New York, USA

Graham says: I love the idea of looking into running real-time stuff on the Raspberry Pi. From an audio perspective, it's quite easyto build a RT kernel for Jack, but I'll have to look into whether it works just as well with the Pi's I/O. Stay tuned!



The Raspberry Pi is like pulling an idea out of your brain and putting it onto a USB-powered PCB.

RANDOM ACTS

Given the revelations that the NSA tries to subvert the strength of encryption by weakening algorithms and by subtly altering the doping of the underlying silicon, we should be suspicious of the integrity of the random number generators. If the NSA is doing such things then it is likely that the Chinese are doing the same – the majority of electronics are now made in China or Taiwan.

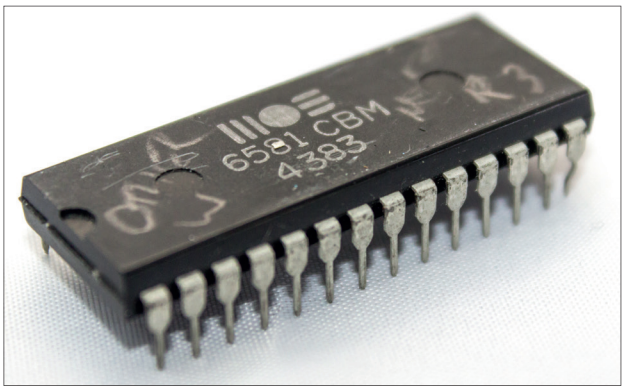
How do we know that our sources of random numbers are truly random? Can Linux Voice address this question? Is there a statistical test that can be applied to **/dev/random** and **/dev/urandom** to verify that they are continuing to provide statistically significant sequences of random numbers?
Andrew Shead.

Ben says: You're right to be concerned about random-number generation. It's a mathematical weak point in most security systems.

One test is to gather output from the random generator in Linux (**/dev/random**) to a file, then try to zip it. In theory, it's impossible for any lossless compression to shrink random data, so if the zipped file is smaller than the uncompressed file, you have a non-random random number generator. However, it's very hard to perform a comprehensive statistical check when you don't know what you're checking for. Imagine, for example, that every CPU is pre-programmed with a list of truly random numbers that it outputs when called. This list could also be sent to the NSA. However, any statistical analysis would show them to be random. Alternatively, the randomness generator could just cycle through non-random numbers under very specific circumstances.

The issue did raise its head last year when an online petition asked Linus to remove the Intel hardware random number generator from **/dev/random**. Essentially, the issue boils down to the fact that Linux generates a random number itself, then XORs the number with one generated by the hardware random number generator. This method of combination means that the outputted number will be as random as the most random input. A compromised hardware generator wouldn't make the system less secure.

Of course, there's also nothing to stop a malicious CPU from listening for software random number generators, and manipulating the output. Ultimately, there's very little defence against a hardware manufacturer that wants to spy on you, except, perhaps, open hardware design like the work of **opencores.org**.

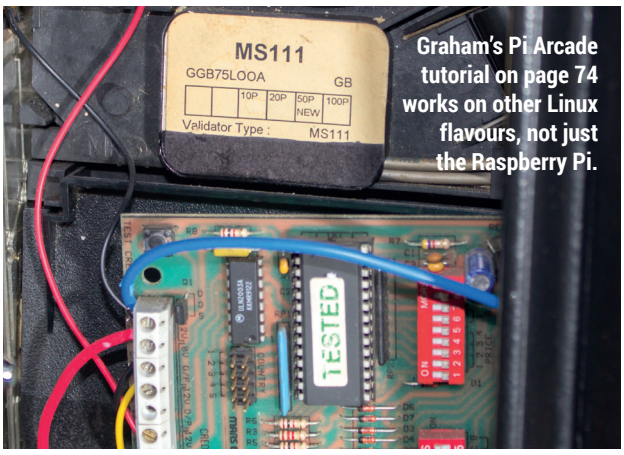


The SID chip, as used on the Commodore 64, is a part-analogue sound chip and also an excellent source of random numbers (noise), making it highly prized by synth geeks and encryption experts alike.

TASTY PIE

Could we see more Raspberry Pi stuff in the new magazine please – in particular, more projects would be good.
Matt Osman

Mike says: Of course, but we have to be careful not to alienate non-Pi users. Luckily most Pi tutorials apply just as well to generic Linux boxes – such as the arcade machine tutorial on page 74. Have fun!



Graham's Pi Arcade tutorial on page 74 works on other Linux flavours, not just the Raspberry Pi.

LUGS ON TOUR

OCR Raspberry Jamboree

Linux Voice heads to the land of dark satanic mills in search of education, innovation and Pi.

In the computing world, almost everything has an acronym associated with it, but OCR isn't one we hear very frequently. It stands for Oxford, Cambridge and RSA, though that tells you little. In fact, it's an exam board, and responsible for striking fear into the hearts of 16–18 year-olds across the UK. That should tell you a couple of things. Firstly, that the Raspberry Pi has been phenomenally successful at getting into education: the fact that this is the second year that a major exam board is running the event in Manchester says a lot about the impact of the Pi in schools. Secondly, it says that the event is going to have quite an academic feel to it.

The event is run in partnership with the Education Innovation Conference and Exhibition. Last year the two events were run in different parts of the Manchester Central Convention Complex, but this year they combined into the main hall. This was slightly disturbing given that the Jamboree charged £20 entrance, yet the EICE was free despite giving access to the same area, and the same stands.

Come together

Annoying though the price difference was, it did give the event a unified feel. The previous year, there was a Pi event and an education event. This year, it felt like there was just one. Alongside a myriad of stands for educational technology, you could see teachers learning Pi skills, and learning from the experts. While 2013's event was perhaps better for Raspberry Pi enthusiasts, 2014 seemed better for teachers, and that is more important in this case.

Pi demonstrations were provided by Simon Walters, Ben Nuttall, Clive Beale and many other learned and knowledgeable folk.

As well as the stands, there was a selection of talks to learn from.

That's not to say that the event wasn't good for enthusiasts. The Raspberry Pi foundation were out in force, with Carrie Anne Philbin, Clive Beale, Alex Bradbury and Ben Nutall wandering the floor and meeting people, though the Uptons didn't make it as they were in America at the time. The Linux Voice team enjoyed meeting up with fellow Raspberry Pi-ers such as RasPi TV, Peter Green (of Raspbian fame) and Jack Kelly (organiser of the Manchester Raspberry Jam, **www.mcrraspjam.org.uk**), and there were plenty more people that we missed (such as the excellent folks at MagPi) because we foolishly went on the Thursday rather than the Friday.

TELL US ABOUT YOUR LUG!

Chances are that you are already a member of a Linux User Group (LUG). LUGs are all over the world and each one has its own unique selling point, which draws its members to meet and discuss their favourite topic. We want to know more about your LUG or hackspace, so please write to us at **lugs@linuxvoice.com** and we might send one of our roving reporters to your next LUG meeting

South Wales LUG

Mark Einon writes from the land of our fathers.

The South Wales Linux User Group (SWLUG) has been in existence for more than 15 years, in that time supporting local users of the Linux operating system and other Free Software.

In April, SWLUG are adding to their usual long-running tradition of well-attended monthly meets in Cardiff and Carmarthen; this time organising a series of talks, starting on Saturday 26 April, 2.30–5pm at FoundersHub (**foundershub.co.uk**) in the centre of Cardiff.

Expect at least four free and awesome main talks on diverse Linux topics such as kernel internals, hardware accelerated Linux routing, the MIT Athena project, and Linux security monitoring. There's also time to fit in a few lightning talks and a keysigning party – so don't forget some sort of official ID (driver's licence, passport etc) for the keysigning, along with your key details. The party will mainly follow the course laid out in **www.cryptnet.net/fdp/crypto/keysigning_party/en/keysigning_party.html**.

If you can't make the afternoon talks in April, stalwart SWLUG members will still be partaking in the usual monthly pub-based meetings in both Cardiff (second Tuesday of the month at The City Arms, opposite the Millenium Stadium) and Carmarthen (third Tuesday of the month, at Yr Hen Dderwen on King Street), from 7.30pm or so where we're always happy to welcome anyone to join whatever their interest or experience level.

Lots more details, further updates and ways to join in all the South Wales-based Linuxy action you can handle may be found via the website at **www.swlug.org**. There you can also find out about our mailing list, IRC channel and Google+ community where you can get involved at any time. Look out for the Eventbrite link to register for talk attendance and we hope to see you there!

Key signing may seem like paranoia to some, but it's only sensible nowadays.

QCon London

Spend a beautiful spring day in London with some of the world's smartest coder geeks.

This was our first time at QCon London, and we can't believe we've missed an event like this for so long. But before you get too excited. It is expensive to attend. This is one of those professional events where you need to surreptitiously place a cost/benefit analysis chart where your boss can see it.

The benefits are obvious. Some of the best developers in the world, working on some of the most cutting edge-projects, such as Spotify and Netflix, all in the same place and talking about the challenges they're facing and how they're solving them. Each talk we saw was inventive and engaging. But our highlight was Wednesday's keynote, presented by Perl maestro Damian Conway. Somehow, he was able to shoehorn John Conway's (no relation) automata from Game of Life into a talk about how programming languages are structured, how insane they can be, and how he's invented a own 24th century programming language based on Klingon, complete with throat-mangling Klingon pronunciation.

QCon costs a chunky £2,295 to attend the conference plus three days of tutorials.

LIBREOFFICE 4.2
FAST, COMPATIBLE, INTEGRATED

LibreOffice 4.2 offers performance and interoperability improvements, that are particularly appealing for power and enterprise users.
You can read and write old and new Microsoft Office files, and import Microsoft Publisher, Microsoft Visio, Corel Draw and Apple Keynote documents. And you can do it faster than ever!

Test our Impress Remote Control for Android and iPhone/iPad
available on Google Play Store and iTunes Store

Support The Document Foundation
with a donation at http://donate.libreoffice.org

Download LibreOffice 4.2:
http://www.libreoffice.org/download/

LibreOffice
The Document Foundation

AND THANK YOU FOR USING LIBREOFFICE! www.libreoffice.org

LISTEN TO THE PODCAST

LINUXVOICE

WWW.LINUXVOICE.COM

LINUX VOICE

REVIEWS

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

Andrew Gregory
Is at long last planing an expedition to Bletchley Park to visit the old computers.

Don't it always seem to go, you don't know what you've got 'til it's gone. Yes, it does.

The thing that I had, which I don't have any more, is freedom over my computing environment. Through necessity I'm spending more and more time on my new Mac, and the limits are chafing like a tight sock.

There's little to no customisation over the GUI, for instance. Then there's the lack of a software centre – if you're used to installing whatever you want through Synaptic or with a simple apt-get, other operating systems seem extremely restrictive in comparison. And the worst thing is that before it lets me update, Apple wants to know my shoe size, inside leg measurement and mother's maiden name. Not cool at all.

Apple does make fantastic hardware, and until recently it's been impossible to find similar quality in a standard laptop. Praise be then, for the Dell XPS 13 – it's built with Linux in mind in conjunction with Ubuntu, and has raised the bar for your next Linux box.

andrew@linuxvoice.com

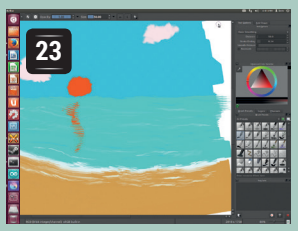
On test this issue...

Dell XPS 13

Wireless problems fixed, this Ubuntu-powered laptop is bringing Linux (and 6-hour battery life) to the High Street.

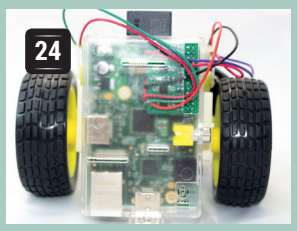
Retrode

Unlock the fun that's currently imprisoned in your old games cartridges. Emulation has never been so much fun.



Krita 2.8

Finger painting goes digital: KDE's image editor gets touchscreen support.



PiBorg

Get into robotics with this cheap and cheerful kit powered by the RPi.

SolydXK

Thesis: regular updates. Antithesis: rolling releases. Synthesis: turn to p25...

BOOKS AND GROUP TEST

Trees are good. They provide shelter, for woodland animals, many of which taste good when casseroled. They can also be mashed up into paper, which makes excellent medium for the written word. This month we're mostly reading about the Raspberry Pi, but Ben also got his hands on *A Computer Called Leo*, which recounts the early adoption of computers by Lyons' Tea shops way back in the 1950s, and tells how the Brits lost the post-war computing initiative to our American colonists despite being jolly good chaps. And in the Group Test, it's text editors.

Dell XPS 13 i7-4650

Graham Morrison is happy to discover the wireless problems that blighted last year's model have gone.

DATA

Web
www.apt-net.co.uk
Developer
Dell
Price
£999 + VAT

There's a lot to like about the XPS 13. Thanks to machined aluminium and carbon fibre, it's a tough, small, powerful, rigid and stylish device that's proved itself in combat. Both last year's model and the new one beneath our collective fingertips feature Intel's powerful Core i7 architecture, with an incredible 1920x1080 13-inch screen, bright backlit keyboard (something essential for nocturnal rambling) and a backpack-friendly form factor.

This XPS13 is just 316mm wide, 205mm deep and 6–18mm high (it's a wedge), and weighs 1.4kg It's slightly lighter than the new 13-inch Apple Macbook Pro but the 2cm less depth makes a considerable difference visually, with no less control from the (still larger than average) trackpad.

Nice specs

The new model has had a specification update, primarily to accommodate Intel's new Haswell CPUs as well as the inclusion of a touchscreen. With the CPU architecture update comes Intel's widely respected Intel HD GPU power. There are two variants of the XPS 13, the first with Intel HD Graphics 4400 and an i7 4500U CPU running at 3GHz, and the second with Intel HD Graphics 5000, i7 4650U at 3.3GHz and a 256GB mSATA SSD, which is the model we're looking at here. This is a powerful netbook, capable of some serious processing and great graphics performance, and we'd highly recommend going with the latter model for the £100 extra it costs.

It's still significant that this is a Dell machine that comes with Ubuntu 12.04 (LTS) installed, thanks to what Dell calls Project Sputnik. This noble endeavour attempts to offer more from Dell than simply buying a laptop with a specification suitable for compiling things. The idea started with Stephen O'Grady of RedMonk, who suggested there may be a market for developers in the emerging cloud markets. The end result was some custom-developed drivers, mostly for the Cypress touchpad, and bespoke cloud management software, profile management and storage that comes with the Developer Edition. These are great features, but we're more interested in a major manufacturer offering a Linux laptop along with its great recovery tools and support, things which don't usually apply to hardware you buy and then install Linux on top of. Still, it's a pity the project hasn't bumped the officially supported version of Ubuntu yet, but that's probably a business/support issue rather than anything to do with the way the XPS works with other version of Ubuntu or any other Linux distro.

The big problem for us with the previous model was wireless: it didn't work. For some reason, the version we'd been sent didn't maintain a connection with any

At low-CPU, low screen brightness and low keyboard LED settings, whilst web browsing and typing, the new XPS 13's battery lasted an exceptional six hours.

There are two charge friendly USB 3 ports, a combination mic/headphone jack and a mini-display port. An SD card slot would have been handy.

access point for more than a couple of minutes. This wasn't an isolated incident either, as there were many threads and comments online with Ubuntu XPS 13 users experiencing the same problems. The problem was with the Centrino Advanced-N 6235 wireless chipset and the iwlwifi driver it was using. Despite the Linux emphasis, it seemed a wireless hardware revision had slipped under the testing dragnet. Dell sent out a Killer Wireless-N 1202 as a replacement. But we're pleased so say the new version doesn't suffer connectivity problem at all. Wireless is now provided by Intel's 7260 chipset, giving strong, reliable 'N' compatible connectivity. Lifting the lid to resume brought us wireless connectivity in around eight seconds, which is only a few moments after you've entered your password to unlock the screen.

Our system was supplied by Apt-net Ltd, a company that's going to provide the XPS 13 in the UK at no extra charge, free delivery and with a choice of distributions. To that end, ours came with Fedora 20, Ubuntu 13.10, Ubuntu 12.04, OpenSuse and LinuxMint 16 helpfully installed on a 32GB USB stick. Every distribution worked well – much better than before some of the patches had made it to the kernel. We're sure with a little tweaking your favourite distro could feel as well integrated with the hardware as Ubuntu 12.04 does with its custom driver pack.

Key

We really like the keyboard. We've typed tens of thousands of words on it. Compared to our benchmark (which is a modern Macbook Pro, sorry!), it's got a softer, more cushioned final third of travel, which we'd imagine is easier on RSI-battered wrists. The horizontal layout is a little wider than normal, but we soon got used to it, and the LEDs behind the keys have variable levels of brightness. The font used on the keys is a little too Sci-Fi for our tastes, but the dimmer settings make the keys perfectly legible even in complete darkness. The granularity of control of the screen brightness was perfect, and brightness settings were great in a dark room, and sharp in sunlight.

The screen itself is gorgeous. As per the original model, we feel that the full HD resolution is a good compromise between the amount you can get on the screen and the clarity of the text. The problem with ultra-high resolutions and Linux is that there's no decent system for scaling whilst keeping the fonts at a reasonable size (in the same way Apple does with its Retina displays), so until this problem can be solved, 1980x1080 is the ideal number of pixels for us. The only slight hitch was some backlight bleed in the lower left of the display. More annoyingly, our model also suffered from a whining sound, emitted most loudly when the keyboard backlight was low. The pitch changed depending on the keyboard brightness, hinting at some regulation interference. Dell's support thread has lots of people with the same issue, regardless of operating system, and Dell is reportedly still looking into the issue. It's worth checking whether the problem has been resolved before you make an order.

> **"Lifting the lid to resume brought us wireless connectivity in around eight seconds."**

Overall, this is a fantastic piece of hardware. It's expensive, yes, but it's designed for people who travel, want to use Linux and still be as productive as they are with a desktop. It's also cheaper than the equivalent Apple units with a similar form factor – many of which are used by Free Software geeks at conferences. The new XPS 13 is a solid upgrade to a powerful machine with excellent Linux credentials.

LINUX VOICE VERDICT

About the most powerful netbook you can buy. Light, bright, packed full of features and it comes with Linux.

★★★★½

Touchscreen

The most intriguing addition for this model is the inclusion of a touch screen. This is likely to be because many Windows 8-based laptops are now featuring touch capabilities to make some sense of Microsoft's latest user interface, but it's a welcome addition for a Linux laptop, especially when so many desktops seem to want to be prodded.

Having used a Nexus 10 with a keyboard for some time, we can see the advantages of having both a laptop and touch screen, but we're not sure this yet translates into extra usability on the XPS 13. Firstly, the laptop itself is too light to be prodded much. The whole unit moves when you touch the screen, although thankfully there's too much friction on the screen's hinge for the angle to change. In Ubuntu 12.04, touching the screen does help when it comes to moving windows out of the way, or selecting text. We'd say it's slightly more efficient than with the touchpad or mouse for specific small jobs. But without the same gesture framework as Android or Ubuntu Mobile, swiping down from the screen edges, for example, or touch gestures to rotate and zoom, or the on-screen contention for the mouse pointer which either represents the location of the mouse or your touch, the touchscreen is an added bonus, rather than a reason to buy this laptop in itself.

Ubuntu 12.04 includes drivers to make the touchscreen work out of the box.

Retrode

Emulation-nut **Mike Saunders** is in heaven. And for once, it has nothing to do with Weißbier…

DATA

Web
www.retrode.org
Developer
Matthias Hullin
Price
€64.90

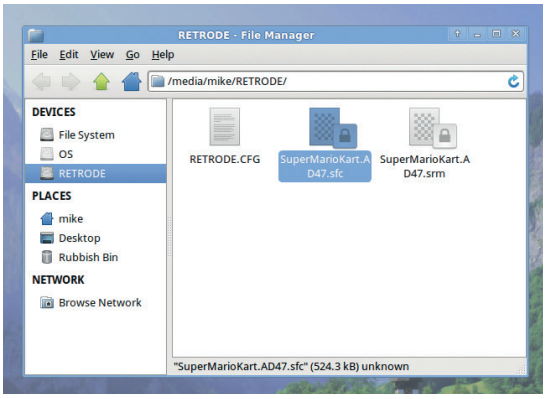
What a fantastic idea this is. We love emulators, and spend way too much time playing classic games from the 80s and 90s. But there's always one problem: the legality of ROM files (the file that contains the data from a read-only memory chip – more or less a clone of a game cartridge). Some people argue that if you already own the physical version of a game, there's nothing wrong with playing a ROM file in an emulator – but then, where do you get the ROM file from? Chances are it has come off a website somewhere, and generated by someone else, so the legal questions remain.

The Retrode avoids all of these complications by enabling you to play your original Super NES and Sega Mega Drive (aka Genesis) cartridges in an emulator on your PC. And the way it does that it is beautifully simple. It doesn't require special data transmission software or custom emulators or anything like that – it just works out of the box.

On the top of the Retrode are two cartridge slots for the aforementioned consoles. Plug in a cartridge, connect the Retrode to your PC via the included USB cable, and you'll see a new removable USB drive appear. Open this drive and *voilà*: the contents of the cartridge are available as a ROM file (eg for SNES games they appear as **.sfc** files), ready to play in an emulator such as ZSNES. You're no longer playing a random ROM file from an unknown source, but the exact code in the physical cartridge that you own.

"It doesn't require special data transmission software or custom emulators – it just works."

Shiny and black, the Retrode reminds us very much of the Sega Master System II.



Plug in your game cart and its contents appear as a ROM image on a removable drive – simple as that.

Rainbow Road revisited

One of the most common annoyances that people have with emulators is having to use different controllers to the original machines. You can get adapters to connect original Super NES and Mega Drive joypads to USB ports, but the Retrode is ahead of the game here and includes four joypad ports: two Super NES and two Mega Drive. These appear to the system as standard USB game controllers, so they will work with any emulator.

And it gets better: if the game has battery-backup save states, in many cases you can access them too. We plugged in our copy of *Super Mario Kart* that we bought in 1992, and a **.srm** file containing our lap time records appeared on the drive, which loaded seamlessly into ZSNES. If you have some saved games that mean the world to you, and you're worried about the in-cart batteries dying, you need this. Now.

The Retrode worked with all the Mega Drive (5) and SNES (8) cartridges we tested, although there are compatibility notes on the website, so it's worth checking those before buying the device. You can even get adapters to play Nintendo 64, Game Boy and Master System games. We're beaming with delight at the prospect of getting our old Wave Race 64 records off the cartridge…

LINUX VOICE VERDICT

A brilliant idea, executed well. Not cheap for casual gamers, but for hard-core emulation fans it's bliss.

★★★★½

Krita 2.8

Ben Everard decamps to the South of France, pours himself a coffee, cuts off his ear and searches for his artistic side.

Image editing software comes in many different forms. The majority we come across, are for touching up photographs. Krita, though, is different. It's designed for digital drawing, and creating new images rather than touching up existing ones. Think of it as digital painting software and you'll be pretty close. It does have some filters that could be used to touch up photos, but you'd almost always be better off with other software for this.

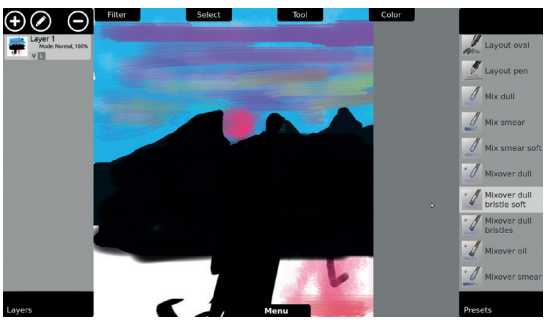
Digital drawing covers quite a wide range of tasks, and Krita is surprisingly capable of meeting most challenges. It's simple and intuitive enough for most beginners to get started with quickly. The range of brush presets has been improved in 2.8, and using just these and a bit of clicking, most people can start to create something loosely resembling art. If you possess some basic artistic talents, you should be able to make something looking good quite quickly.

Yet despite how easy it is to get started, Krita is still complex and powerful enough to suit the needs of many professional digital artists. The dockers interfaces make it easy to show or hide functionality so you have access to the tools you need, but don't overcrowd the window. For serious users, various levels of commercial support are available through **www.kritastudio.com**.

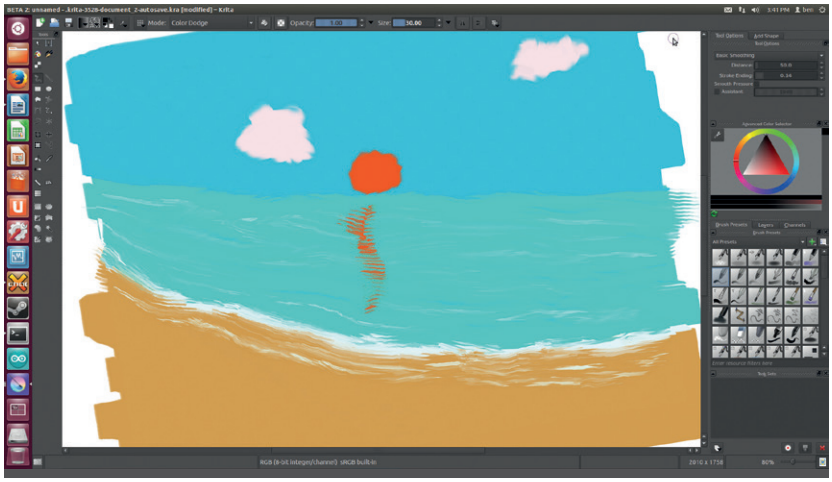
Free software on Windows

2.8 is an important release for the project and will probably see a large uptake in users for two reasons: it's the first stable version for Windows, and it's been greenlit by the community to come to Steam (**www.steamcommunity.com/sharedfiles/filedetails/?id=225403385**). While this is exciting news for the project, neither of these things really affects how the software performs for us Linux users. But still, welcome to the party, everyone else!

Krita now comes with Krita sketch. This is the a new user interface that's been designed for touchscreens. It was developed in partnership with Intel, presumably as a way to show off laptops that can convert into tablets (see the demo video at **www.youtube.com/watch?v=qiSGuCzRZFk**). This looks pretty, but it's almost useless unless you own a very specific type of device, especially as there's no Android support.



Krita Sketch is useful for posing with transformable touchscreen laptops, but not much else.



It's a sunset over an ocean, in case you were wondering.

There is some good news for Linux users. The wrap-around mode makes it much easier to create tiling images, which is great if you're working on game art. OpenGL support has also been improved, so some of the rendering can now be offloaded to free up the CPU. The developers claim that support for graphics tablets has been improved, though we don't have a device to test this on. Along with the usual bugfixes, this adds up to a useful, though not overly exciting release.

We can forgive Krita 2.8 for being focused on Windows. After all, it's the first stable Windows version. If this makes it easier for KO GmbH (the company that supports development of Krita) to fund development, then we all win. There's nothing wrong with version 2.8, but, unless you need wrap-around mode, or happen to own a laptop that transforms into a tablet, there's little reason to rush out to download the latest version.

> **"Krita is powerful enough to suit the needs of many professional digital artists."**

DATA

Web
www.krita.org
Developer
Krita Foundation
Price
Free under GPL or from €225 per person per year

LINUX VOICE VERDICT

A solid release for Linux users, and an exciting release for Windows users.

★★★★★

PiBorg PiCy

Ben Everard loses hours of his life to this little kit that sets your Raspberry Pi free to roam around the kitchen.

DATA

Web
www.piborg.org
Developer
PiBorg
Price
£29.99 (not including Raspberry Pi)

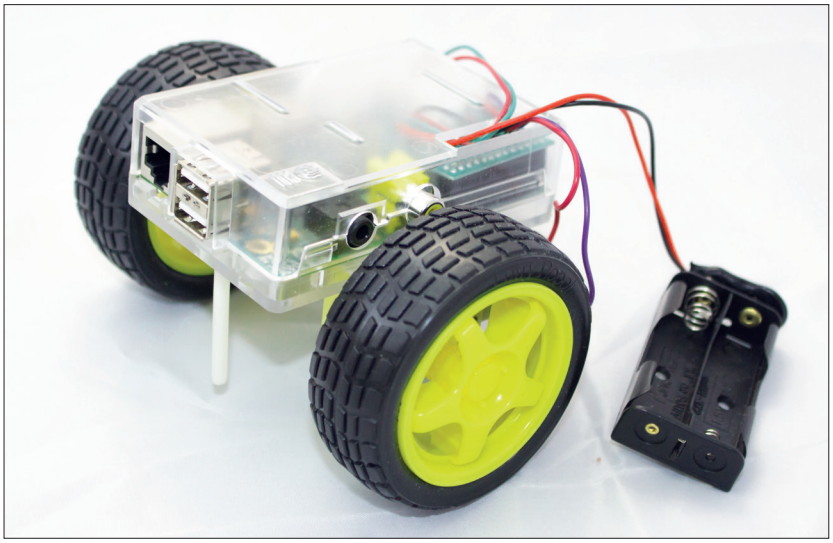
The Raspberry Pi is a great base for robotics projects of all shapes and sizes, and the PiCy is about as small as they come. It's just a Raspberry Pi case with a pair of motors and wheels.

Assembling the kit requires a little soldering, though none of it is difficult or fiddly (the kit is also available pre-soldered for an extra £7). Everything else can be assembled with no tools at all. The case snaps over the Pi, the motors and battery case are glued on, and the wheels are just pushed into place. It took us about 30 minutes to build ours, and that included time to take step-by-step photos.

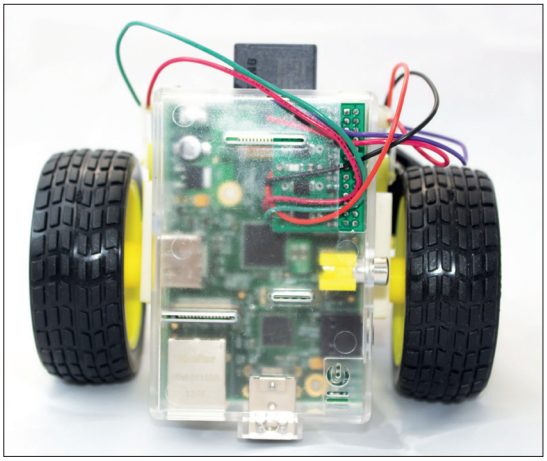
The only electronics are a small circuit board to power the motors. This board (called a PicoBorg) is just a group of Field Effect Transistors (FETs) that are used to control higher-power devices such as motors.

Once assembled, controlling it is just a matter of using the GPIO pins. There's example code on the PiBorg website to do this with Python, though it could be done in other languages if you prefer. Getting this running was as easy as assembling it. Most other motor control boards for the Pi use special libraries to control the hardware. The fact that this works on pure GPIO output makes it a great first kit for anyone who wants to learn more about controlling things with the Pi. However, this does give you less control over the motors – you can only turn each one on or off, so there's no speed control and no reverse.

"Considering how easy it is to program, the PiCy is a great introduction to robotics."



Though not strictly necessary, a portable USB power supply will let you run your creation untethered, and a Wi-Fi dongle will enable you to communicate with it once it's free.



WThe PicoBorg (the electronics at the heart of the kit) is available by itself for £6.99 including UK shipping.

The essence of robotics is compromise, and in order to make it easy to get started, the kit sacrifices some things. The glued-together frame doesn't feel particularly strong, so it's unlikely to cope well with rough handling. That said, the first thing to break will almost certainly be the double-sided tape, which would be trivial to repair.

My First Robot

The two video connectors on the Pi are inaccessible when it's strapped into the PiCy, so you couldn't use this as a base for anything with a HDMI screen, but then it's not likely to be big enough for that anyway.

To get up and running, you'll need the PiCy kit, a Raspberry Pi and the usual paraphernalia (SD card, power supply, etc). The main base for the chassis is a ModMyPi case, which is excellent at protecting the Pi, but doesn't make a very expandable basis on which to build. There aren't any holes or anything else to mount additional hardware on. The best bet for expanding it is either gluing parts on or drilling into the case. On the plus side, there are two unused protected GPIOs on the PicoBorg (including one with PWM) to add more motors or other peripherals to your robot. At this price, and considering how easy it is to assemble and program, the PiCy is a great introduction to robotics, and the parts it's made from can easily be recycled into to more advanced projects as your skills develop.

LINUX VOICE VERDICT

Possibly the easiest way that we've seen to to build a robot based on the Raspberry Pi.

★★★★½

SolydXK 201401

Does the world really need yet another Debian-based distro? Well, the answer is yes, as **Mike Saunders** discovers.

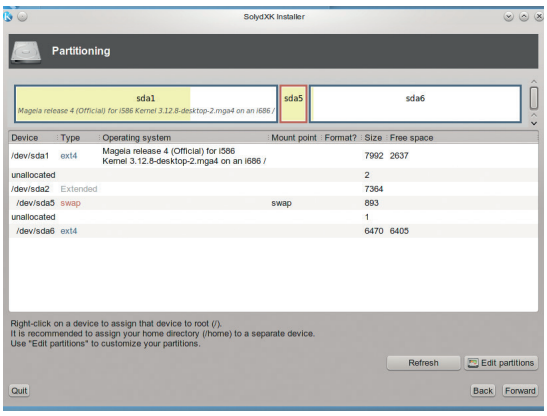
SolydXK is an unassuming distro that could very easily have missed your radar. After all, how many other desktop-oriented Debian spin-offs are doing the rounds? We've lost count now. But after two readers mentioned SolydXK in last issue's letters page, we had to take it for a spin. And we're glad we did, because its release model could become the norm for desktop distros in the future.

Right now, there are two main ways that distro makers approach releases: some, like Ubuntu and Fedora, create one or two big releases each year. Everyone waits patiently for several months and then does the big upgrade to get all the hottest new software. It works fairly well, but sometimes it's a drag having to wait. Then there's the rolling model, as used prominently by Gentoo and Arch, where you get new software as soon as it's released by the original developers. This lets you live on the bleeding edge, but things can occasionally break due to a lack of testing and integration.

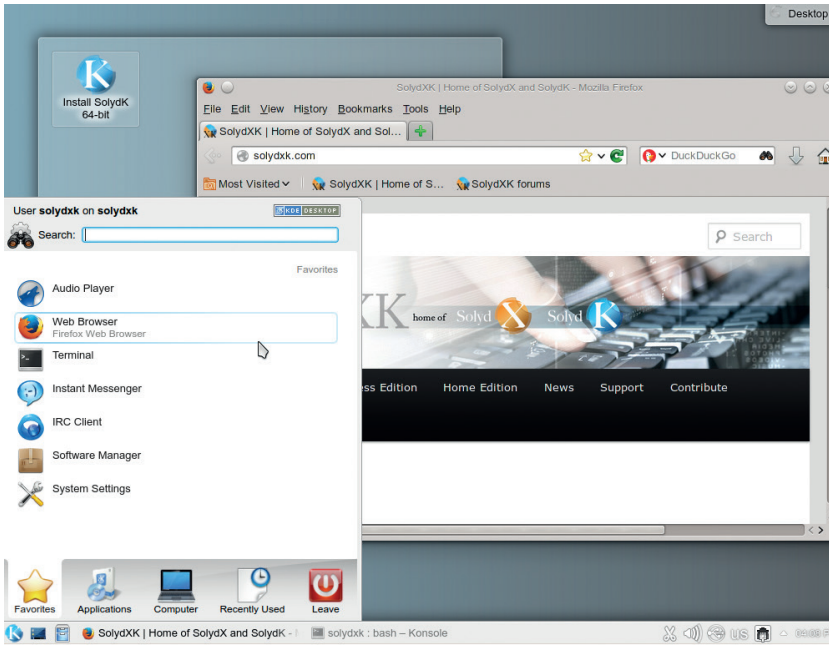
SolydXK is a half-way house; you only have to install it once, and then every three months you'll receive Update Packs to get the latest software. So there are no fanfare-surrounded major releases where everyone rushes to upgrade – you just get the newest programs in well-tested bundles. This is a great approach, and hope to see it adopted by more desktop Linux distributions.

What's in the box?

Two versions are available: KDE and Xfce (we tested the former here). In use it's a fairly vanilla Debian testing-based distro with a stock KDE 4.12 installation, supported by LibreOffice 4.1, Firefox 26 and various other desktop apps. VLC is included together with a bunch of media codecs, so the distro can handle most formats from the get-go. Along with Synaptic, Linux Mint's attractive Software Manager is included, and the installer is also based on Mint's.



SolydXK's Mint-based installer is quick, easy to use and versatile enough for power users.



Some people will like the distro's use of a plain KDE setup, but we'd like to see a bit more individuality shine through – or you could follow page 84's tutorial and tweak it yourself.

DATA

Web
www.solydxk.com
Developer
The SolydXK team
Price
Free (various open source licences)

There are a few rough edges: the presentation and theming isn't very slick, and some work could be done to tidy up the menus (having both Package Manager and Package Installer in the System menu could confuse new users, for instance). But the SolydXK team has demonstrated real commitment to the project, with installation videos on the website, a growing community, and annual reports explaining how things are progressing.

> **"The SolydXK team has demonstrated real commitment to the project."**

Time will tell if the Upgrade Packs work as intended and the goal of "you don't have to reinstall, ever" can be achieved. But it's certainly a distribution to keep an eye on, and if it gathers enough interest it could make some of the others (with their traditional timed-interval release cycles) start to look rather old-hat.

LINUX VOICE VERDICT

A promising distro with a fresh approach to updates – it just needs more personality of its own.

★★★½☆

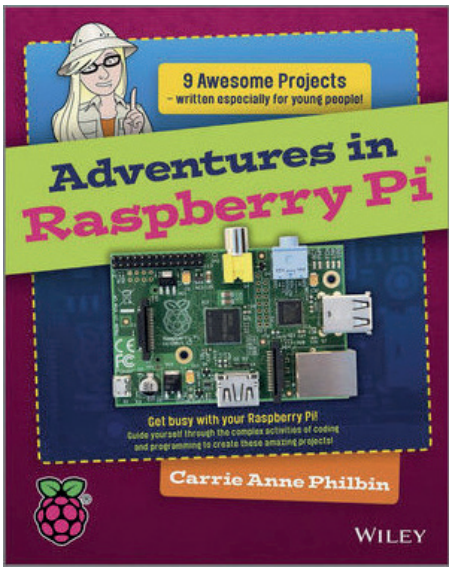
Adventures in Raspberry Pi & Raspberry Pi USer Guide

Andrew Gregory spends some rainy days with the world's favourite Linux machine.

The Raspberry Pi came about because Eben Upton, the PiFather, decided that something needed to be done to improve the quality of students applying to study on his university computer science courses. So far, so sensible, and you can read more about the effect that the raspberry Pi is having on education in our interview with the Pi Foundation's education team on page 46. But the Pi has also found an audience among the massed legions of home tinkerers, shed solderers, robot builders, beer brewers and people who generally have an interest in home hackery.

The two books we're looking at here have these two readerships in mind. Adventures in Raspberry Pi, by Carrie Anne Philbin, is aimed squarely at children, teachers and parents looking for a way into computer science. Raspberry Pi User Guide, with its oh-so-serious graph paper on the cover (a brilliant idea) is targeted at the older hacker, at adults who have some experience of programming, and who want to transfer their existing knowledge to this relatively new device.

It's not surprising then that Adventures is the more accessible of the two. The book is divided into nine adventures, beginning with installing Raspbian with Noobs, an introduction to the command line, the Scratch programming language, drawing with a Python turtle (just like we're doing this issue with our Python tutorial on **p96**). The final project is building a jukebox with the Pi, using a speaker, a breadboard an LCD display and the Pi's GPIO pins. This is the sort of thing that a supportive parent would be likely to find were they to simply Google for Raspberry Pi projects, and for most people it's just too challenging. Adventures in Raspberry Pi slowly builds up the difficulty, constantly challenging the newcomer to learn more and improve on their skills, and that's what makes it such an excellent book for teachers, kids, parents or just non-technical humans who want to learn a bit.



If we had to sum this book up in a word, that word would be: authoritative.

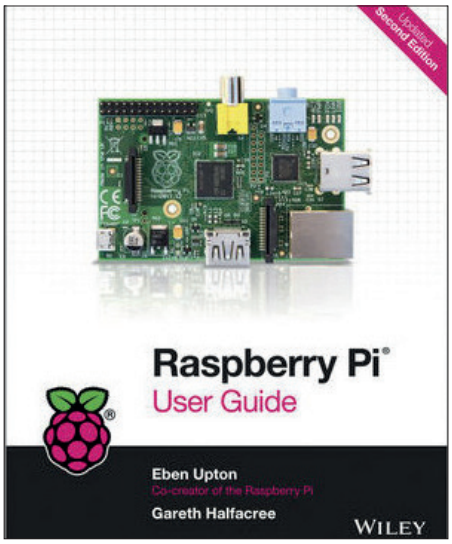
Raspberry Pi User Guide

In contrast, Raspberry Pi User guide is very much preaching to the converted. It's a geek fest, and almost seems designed to deter casual readers. There's a discussion on the merits of x86 vs ARM chips for instance, and sections on relatively advanced and stuff like network configuration and HDMI display modes.

Whereas Adventures presents a smooth learning curve, User Guide feels like it's had facts packed into it rather than projects, diminishing its usefulness as a learning tool. The chapter on Scratch, for example, the programming language aimed at kids, comes after the chapter on using the Pi as a web server. This doesn't make sense to us.

While not that great as a structured programme of learning, User Guide is a great reference book. There's (luxury!) a nice index, and its pages speak of authority. Even though it's not especially weighty at 300 pages – OK, so it's not light reading, but computer tomes often come in at around the 800 mark – it feels like a definitive, official work. There's a confidence to the writing that you can feel as a reader. This feels like a definitive work.

Which of these books work for you depends on you. We'd recommend them both in a heartbeat. You won't like both of these, but you will definitely like one or the other, depending on where you are in your Linux journey.



If we had to sum this book up in a word, that word would be: accessible.

LINUX VOICE VERDICT

ADVENTURES IN RASPBERRY PI

Author Carrie Anne Philbin
Publisher Wiley
ISBN 978-1-118-75125-1
Price £14.99/US $24.99/CAN $29.99

Brilliantly thought out, this is the perfect guide to catalyse computing curiosity.

★★★★★

RASPBERRY PI USER GUIDE

Authors Gareth Halfacree & Eben Upton
Publisher Wiley
ISBN 978-1-118-79548-4
Price £17.99/US $19.99/CAN $23.99

Contains many facts, all of them true. The definitive reference guide.

★★★★½

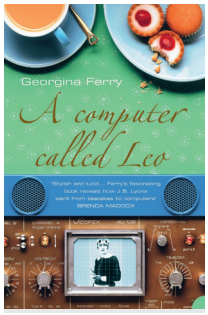
A Computer Called LEO

Ben Everard discovers that tea shops and computers go together.

The story of how Silicon Valley grew to dominate the computing world has been told many times, but it's not the only tale of how computers developed. While IBM were still building adding machines, a British company famed for its teashops built its own computer and ran the world's first office computing jobs. They ran payroll, management accounting jobs and others long before anybody else realised such things were possible.

In *A Computer Called Leo*, Brenda Maddox charts the history of this very British company and how it started the office computer revolution. It's not a technical exploration of how they worked, but a gentle history of what ultimately became a side-note in technology.

It is, perhaps, an important story of how plucky British boffins created some of the best technology in the world, but ultimately failed in the face of international competition. You can read this as a cautionary tale or an entertaining story, just don't expect a tech-heavy geek fest. It's perfect for reading over a cream tea on a sunny afternoon with the tune of Rule Britannia gently wafting through the air.



It's just like *Downton Abbey*, just with computers and forty years into the future.

LINUX VOICE VERDICT

Author Brenda Maddox
Publisher Harper Collins
ISBN 978-1-84115-186-1
Price £11.99

Computers, tea and cakes. Chuck in a pint of real ale and you've got the perfect way to spend an afternoon.

★★★★½

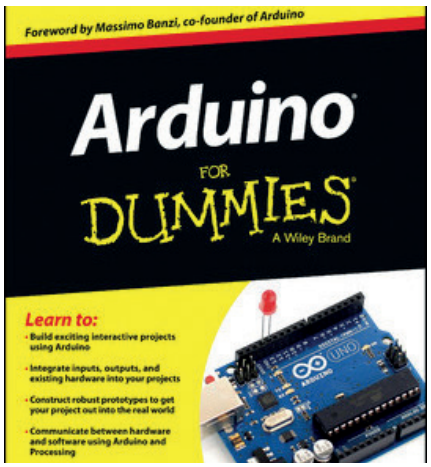
Arduino for dummies

Ben Everard ignores this book's cover, then enjoys the contents.

The Arduino is probably the easiest way to get started in interactive electronics. That said, anything that involves programming or electronics is guaranteed to scare some people. Anything that involves both programming and electronics (as the Arduino does) is going to scare quite a few people.

This reviewer has always steered clear of 'For Dummies' titles because they seem to confuse not knowing about a subject with stupidity. This book is clearly for people who don't know about the Arduino, dummy or not. A dummy who happened to know about the Arduino wouldn't have much use for the book and is probably a very dangerous person to have around (never let them plug anything into your power sockets).

Title aside, this book is a gentle intro to the Arduino starting from the assumption that the reader knows nothing about electronics or programming and building up from there. Despite this, it covers a wide range of hardware and projects.



Ben isn't a dummy, but he enjoyed this book.

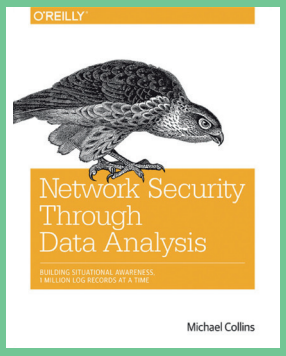
LINUX VOICE VERDICT

Author John Nussy
Publisher John Wiley and sons
ISBN 978-1-118-44637-9
Price £16.99

One of the gentlest introductions to the Arduino, if you can stomach the title.

★★★★½

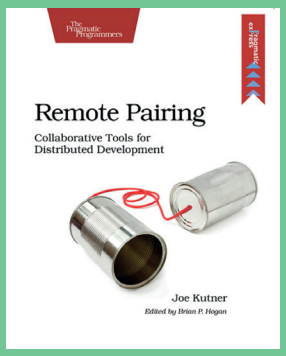
ALSO RELEASED...



Situational awareness is the key phrase for network analysis

Network Security Through Data Analysis

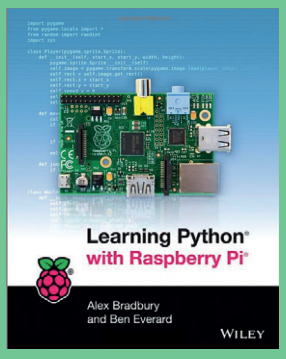
This book looks like the networking security equivalent to Crystal Maze. It gives you the tools to snoop on your own network traffic in an attempt to head off the bad guys before they even know they've been found.



Tin cans and a piece of string? You were lucky. All we had were yoghurt pots.

Remote Pairing

Feeling lonely? Paired programming is the latest thing to help isolated programmers develop more of a social life, whilst learning to interact with real human beings. Actually, it's not. It's a cool new methodology that's proving two heads are better than one.



Python and Pi: a match made in heaven.

Learning Python with Raspberry Pi

This book looks so wonderful, so learned and so entertaining we've already enlisted the help of top book reviewer, Brander Eve, to take a look at this title for us. Early reports indicate it's everything we hoped it would be, but we'll wait on Brander's final verdict next month.

LINUX VOICE

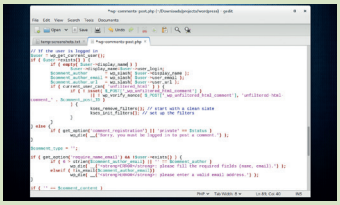
ADVANCED TEXT EDITORS

GROUP TEST

Mayank Sharma tests five supercharged text editors that can crunch more than just words.

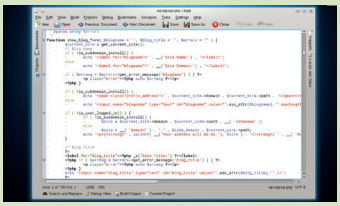
On Test

Gedit



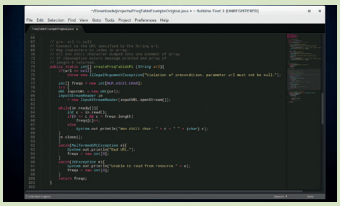
URL http://projects.gnome.org/gedit/
Version 3.10
Licence GPL
Is Gnome's default text editor up to the challenge?

Kate



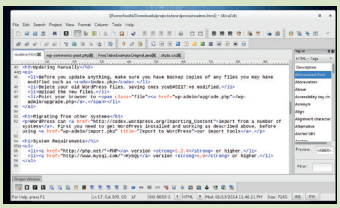
URL www.kate-editor.org
Version 3.11
Licence LGPL/GPL
Will Kate challenge fate?

Sublime



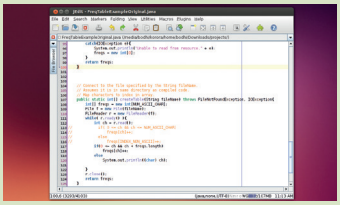
URL www.sublimetext.com
Version 2.0.2
Licence Proprietary
Proprietary software in the land of free with the heart of gold.

UltraEdit



URL www.ultraedit.com
Version 4.1.0.4
Licence Proprietary
Does it do enough to justify its price?

jEdit



URL www.jedit.org
Version 5.1.0
Licence GPL
Will the Java-based editor spoil the party for the rest?

Advanced Text Editors

The right editor can be passport to a better Linux.

If you've been using Linux long, you know that whether you want to edit an app's configuration file, hack together a shell script, or write/review bits of code, the likes of LibreOffice just won't cut it. Although the words mean almost the same thing, you don't need a word processor for these tasks; you need a text editor.

In this group test we'll be looking at five humble text editors that are more than capable of heavy-lifting texting duties. They can highlight syntax and auto-indent code just as effortlessly as they can spellcheck documents. You can use them to record macros and manage code snippets just as easily as you can copy/paste plain text.

Some simple text editors even exceed their design goals thanks to plugins that infuse them with capabilities to rival text-centric apps from other genres. They can take on the duties of a source code editor and even an Integrated Development Environment.

Two of most popular and powerful plain text editors are Emacs and Vim. However, we didn't include them in this group test for a couple of reasons. Firstly, if you are using either, congratulations: you don't need to switch. Secondly, both of these have a steep learning curve, especially to the GUI-oriented desktop generation who have access to alternatives that are much more inviting.

> **"Some simple text editors even exceed their design goals thanks to plugins."**

THE CRUCIAL CRITERIA

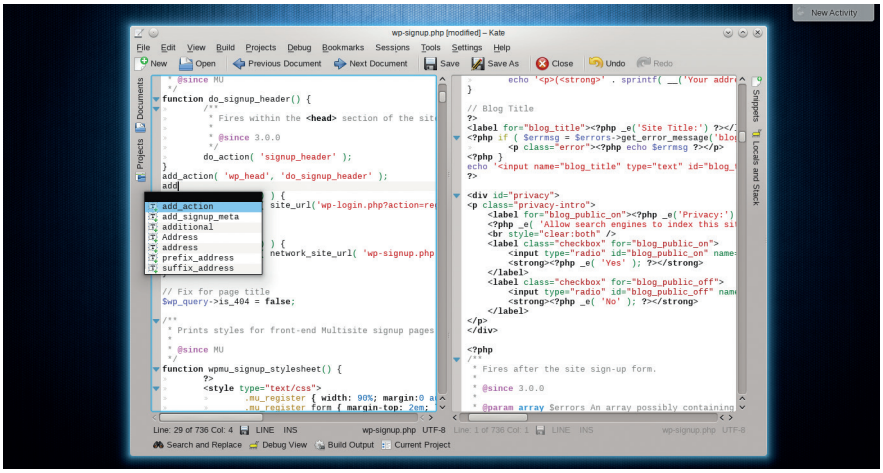
All the tools, except Gedit and jEdit, were installed on Fedora and Ubuntu via their recommended installation method. The former already shipped with the default Gnome desktop and the latter stubbornly refused to install on Fedora. Since these are relatively simple apps, they have no esoteric dependencies, the only exception being jEdit, which requires Oracle Java.

Thanks to the continued efforts of both Gnome and KDE, all editors look great and function properly irrespective of the desktop environment they are running on. That not only rules it out as an evaluation criterion, it also means that you are no longer bound by the tools that ship with your favourite desktop environment.

In addition to their geekier functionality, we also tested all our candidates for general-purpose text editing. However, they are not designed to mimic all the functionality of a modern-day word processor and weren't evaluated as such.

Programming language support

They're called code editors for a reason.



Kate can double up as a very versatile and capable integrated development editor.

UltraEdit does syntax highlighting, can fold code and has project management capabilities. There's also a function list, which is supposed to list all the functions in the source file, but it didn't work for any of our test code files. UltraEdit also supports HTML5, and has a HTML toolbar with which you can add commonly-used HTML tags.

Even Gnome's default text editor, Gedit, has several code-oriented features such as bracket matching, automatic indentation, and will also highlight syntax for various programming languages including C, C++, Java, HTML, XML, Python, Perl, and many others.

If you're looking for more programming assistance, look at Sublime and Kate. Sublime supports several programming languages and (as well as the popular ones) is able to highlight syntax for C#, D, Dylan, Erlang, Groovy, Haskell, Lisp, Lua, MATLAB, OCaml, R, and even SQL. If that isn't enough for you, you can download add-ons to support even more languages.

Furthermore, its syntax highlighting ability offers several customisable options. The app will also match braces, to ensure they are all properly rounded off, and the auto-complete function in Sublime works with variables created by the user.

Just like Komodo IDE, sublime also displays a scrollable preview of the full source code, which is really handy for navigating long code files and lets you jump between different parts of the file.

One of the best features of Sublime is its ability to run code for certain languages like C++, Python, Ruby, etc from within the editor itself, assuming of course you have the compiler and other build system tools installed on your computer. This helps save time and eliminates the need to switch out to the command line.

You can also enable the build system in Kate with plugins. Furthermore, you can add a simple front-end to the GDB debugger. Kate will work with Git, Subversion and Mercurial version control systems, and also provides some functionality for project management.

It does all this in addition to highlighting syntax for over 180 languages, along with other assistance like bracket matching, auto-completion and auto-indentation. It also supports code folding and can even collapse functions within a program.

The only disappointment is jEdit, which bills itself as a programmer's text editor, but it struggled with other basic functions such as code folding and wouldn't even suggest or complete functions.

VERDICT

Gedit	★★★☆☆
Kate	★★★★★
Sublime	★★★★★
UltraEdit	★★★☆☆
jEdit	★☆☆☆☆

Keyboard control

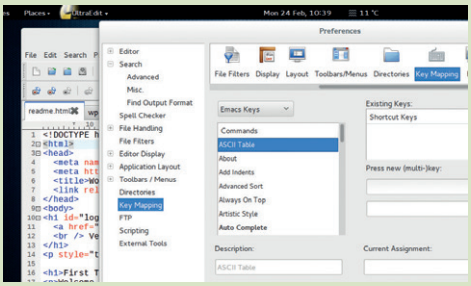
Flex those fingers!

Users of an advanced text editor expect to control and operate it exclusively via the keyboard. Furthermore, some apps even allow their users to further customise the key bindings for the shortcuts.

You can easily work with Gedit using its extensive keyboard shortcut keys. There are keys for working with and editing files as well as invoke tools for common tasks such as spellchecking a document. You can access a list of default shortcut keys from within the app, but there's no graphical way to customise them. Similarly, to customise the keybindings in Sublime, you need to make modifications in its XML keymap files. Sublime has been criticised for its lack of a graphical interface to define keyboard shortcuts, but long-term users have defended the current file-based mechanism, which gives them more control.

UltraEdit is proud of its "everything is customisable" motto, which it extend to keyboard shortcuts. You can define custom hotkeys for navigating the menus and also define your own multi-key key-mappings for accessing its plethora of functions.

In addition to its fully customisable keyboard shortcuts, jEdit also has pre-defined keymaps for Emacs. Kate is equally impressive in this respect. It has an easily accessible window to customise the key bindings. You can change the default keys, as well as define alternate ones. Furthermore, Kate also has a Vi mode which will let users operate Kate using Vi keys.



Both UltraEdit and Kate have options to print a reference guide for all their keyboard shortcuts.

VERDICT

Gedit	★★☆☆☆
Kate	★★★★★
Sublime	★★★☆☆
UltraEdit	★★★★☆
jEdit	★★★★★

Snippets and macros

Because time is money.

Macros help you cut down the time spent on editing and organising data by automating repetitive steps, while Snippets of code extend a similar functionality to programmers by creating reusable chunks of source code. Both have the ability to save you time.

The vanilla Gedit installation doesn't have either of these functionalities, but you can enable them via separate plugins. While the Snippets plugin ships with Gedit, you'll have to manually download and install the macro plugin (it's called **gedit-macropy** and is hosted on GitHub) before you can enable it from within Gedit.

Kate takes the same plugins route to enable the snippets feature. Once added, the plugin also adds a repository of snippets for PHP, Bash and Java. You can display the list of snippets in the sidebar for easier access. Right-click on a snippet to edit its contents as well as its shortcut key combination. However, very surprisingly, it doesn't support macros – despite repeated hails from users since 2002!

jEdit too has a plugin for enabling snippets. But it can record macros from user actions and you can also write them in the BeanShell scripting language (BeanShell supports scripted objects as simple method closures like those in Perl and JavaScript). jEdit also has a plugin that will download several macros from jEdit's website.

Sublime ships with inbuilt ability to create both snippets and macros, and ships with several snippets of frequently used functions for most popular programming languages.

Snippets in UltraEdit are called Smart Templates and just like with Sublime you can insert them based upon the kind of source file you're editing. To complement the Macro recording function, UltraEdit also has an integrated javascript-based scripting language to automate tasks. You can also download user-submitted macros and scripts from the editor's website.

VERDICT

Gedit	★★★☆☆
Kate	★☆☆☆☆
Sublime	★★★★★
UltraEdit	★★★★★
jEdit	★★★★★

Ease of use

Inviting or intimidating?

Unlike a bare-bones text editor, the text editors in this feature are brimming with features to accommodate a wide range of users – from document writers to programmers. Instead of stripping features from the apps, their developers are looking for avenues to add more functionality.

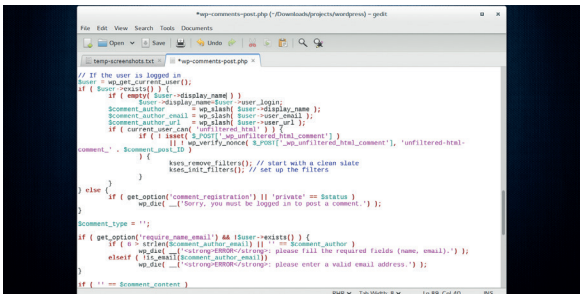
Although at first glance most apps in this group test have a very similar layout, upon closer inspection, you'll notice several usability differences. We have a weak spot for apps that expose their functionality and features by making judicious use of the user interface, instead of just overwhelming the user.

Gedit ★★★★☆

Gedit wears a very vanilla look. It has an easy interface with minimal menus and buttons. This is a two-edged sword though, as some users might fail to realise its true potential.

The app can open multiple files in tabs that can be rearranged and moved between windows. Users can optionally enable panels on the side and bottom for displaying a file browser and the output of a tool enabled by a plugin. The app will detect when an open file is modified by another application and offers to reload that file.

The UI has been given a major overhaul in the latest version of the app yet to make its way into Gnome. However it isn't yet stable, and while it maintains all features, several plugins that interact with the menu will need to be updated.

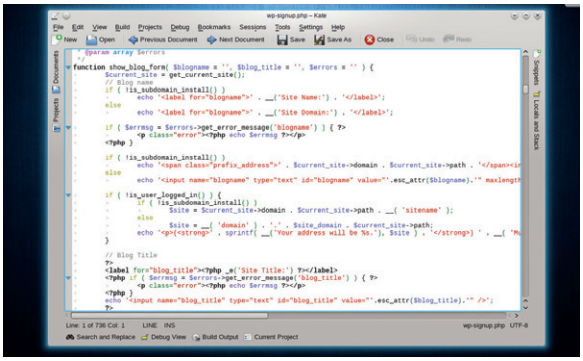


There's a fine balance between stuffing an app with features and exposing all of them to the user – Gedit keeps most of its features hidden.

Kate ★★★★★

Although a major part of its user interface resembles Gedit, Kate tucks in tabs at either side and its menus are much fuller. The app is approachable and invites users to explore other features.

Kate can transparently open and save files over all protocols supported by KDE's KIO including HTTP, FTP, SSH, SMB and WebDAV. You can use the app to work with multiple files at the same time. But unlike the traditional horizontal tab switching bar in most app, Kate has tabs on either side of the screen. The left sidebar will display an index of open files. Programmers who need to see different parts of the same file at the same time will also appreciate its ability to split the interface horizontally as well as vertically.

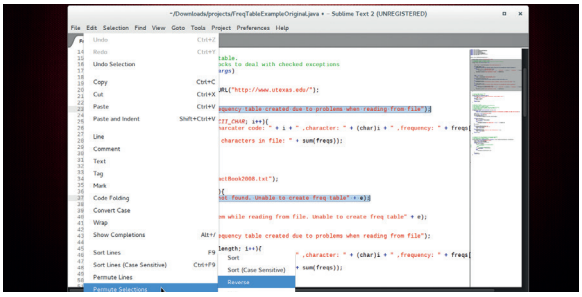


KDE's philosophy of exposing functionality to the user applies to Kate, and works well in this context.

Sublime Text ★★★★★

Sublime lets you view up to four files at the same time in various arrangements. There's also a full-screen distraction free mode that just displays the file and the menu, for when you're in the zone.

The editor also has a minimap on the right, which is useful for navigating long files. The app ships with several snippets for popular functions in several programming languages, which makes it very usable for developers. Another neat editing feature, whether you are working with text documents or code, is the ability to swap and shuffle selections.



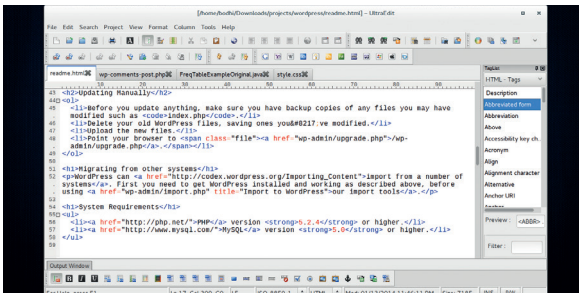
If you don't like Sublime's default 'Charcoal' appearance you can choose one of the 22 other themes it includes.

UltraEdit ★★★☆☆

UltraEdit's interface is loaded with several toolbars at the top and bottom of the interface. Along with the tabs to switch between documents, panes on either side and the gutter area, these leave little room for the editor window.

Web developers working with HTML files have lots of assistance at their fingertips. You can also access remote files via FTP and SFTP. Advanced features such as recording a macro and comparing files are also easily accessible.

Using the app's Preferences window you can tweak various aspects of the app, including the colour scheme and other features like syntax highlighting.

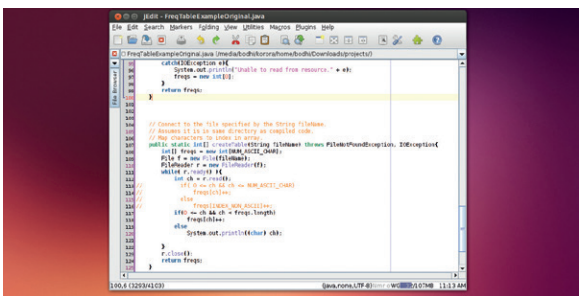


UltraEdit's UI is highly configurable – you can customise the layout of toolbars and menus just as easily as you can change many other aspects.

jEdit ★★★☆☆

In terms of usability, one of the first red-flags was jEdit's inability to install on RPM-based distros. Navigating the editor takes some getting used to, since its menus aren't in the same order as in other popular apps and some have names that won't be familiar to the average desktop user. However, the app include detailed inbuilt help, which will help ease the learning curve.

jEdit highlights the current line you are on and enables you to split windows in multiple viewing modes. You can easily install and manage plugins from within the app, and in addition to full macros, jEdit also lets you record quick temporary ones.



Thanks to its Java underpinnings, jEdit doesn't really feel at home on any desktop environment.

Availability and support

Where do you look for help?

There are several similarities between Gedit and Kate. Both apps take advantage of their respective parent project, Gnome and KDE, and are bundled with several mainstream distros. Yet both projects are cross-platform and have Windows and Mac OS X ports as well as native Linux versions.

Gedit is hosted on Gnome's web infrastructure and has a brief user guide, information about the various plugins, and the usual channels of getting in touch including a mailing list and IRC channel. You'll also find usage information on the websites of other Gnome-based distros such as Ubuntu. Similarly, Kate gets the benefit of KDE's resources and hosts detailed user information as well as a mailing list and IRC channel. You can access their respective user guides offline from within the app as well.

UltraEdit is also available for Windows and Mac OS X besides Linux, and has detailed user guides on getting started, though there's none included within the app. To assist users, UltraEdit hosts a database of frequently asked questions, a bunch of power tips that have detailed information about several specific features, and users can engage with one another other on forum boards. Additionally, paid users can also seek support from the developers via email.

Sublime supports the same number of platforms, however you don't need to buy a separate licence for each platform. The developer keeps users abreast with ongoing development via a blog and also participates actively in the hosted forums. The highlight of the project's support infrastructure is the freely available detailed tutorial and video course. Sublime is lovely.

Because it's written in Java, jEdit is available on several platforms. On its website you'll find a detailed user guide and links to documentation of some plugins. However, there are no avenues for users to engage with other users or the developer.

VERDICT

Gedit	★★★★☆
Kate	★★★★☆
Sublime	★★★★★
UltraEdit	★★★☆☆
jEdit	★★☆☆☆

Add-on and plugins

To flesh out new features.

Different users have different requirements, and a single lightweight app can only do as much. This is where plugins come into the picture. The apps rely on these small pluggable widgets to extend their feature set and be of use to even more number of users.

The one exception is UltraEdit. The app has no third-party plugins, but its developers do point out that third-party tools such as HtmlTidy are already installed with UltraEdit.

Gedit ships with a number of plugins installed, and you can download more with the **gedit-plugins** package. The project's website also points to several third-party plugins based on their compatibility with the Gedit versions.

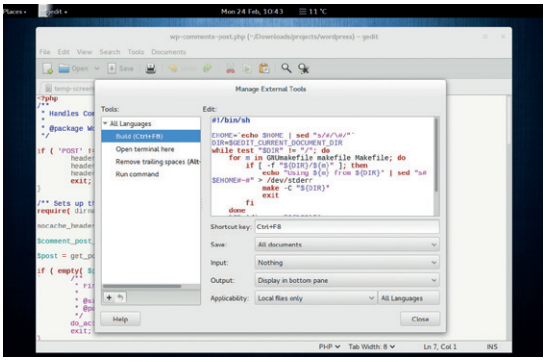
Three useful plugins for programmers are Code Comment, Terminal Plugin, which adds a terminal in the bottom panel, and the Session Saver. The Session Saver is really useful when you're working on a project with multiple files. You can open all the files in tabs, save your session and when you restore it with a single click it'll open all the files in the same tab order as you saved them.

Similarly, you can extend Kate by adding plugins using its built-in plugin manager. In addition to the impressive projects plugins, some others that will be of use to developers include an embedded terminal, ability to compile and debug code and execute SQL queries on databases.

Sublime plugins

Plugins for Sublime are written in Python, and the text editor includes a tool called Package Control, which is a little bit like **apt-get** in that it enables the user to find, install, upgrade and remove plugin packages. With plugins, you can bring the Git version control to Sublime, as well as the JSLint tool to improve JavaScript. The Sublime Linter plugin is a must have for coders and will point out any errors in your code.

jEdit boasts the most impressive plugin infrastructure. The app has over 200 plugins, which can be browsed in the dedicated site of their own. The website lists plugins under various categories such as File Management, Version Control, Text, etc. You'll find lots of plugins housed under each category.



You can run any external command or shell script in Gedit by defining it as an external tool.

Some of the best plugins are the Android plugin, which provides utilities to work on Android projects; the TomcatSwitch plugin, using which you can create and control an external Jakarta Tomcat server process; and the Vimulator plugin, for Vi-like capabilities. You can install these plugins using jEdit's using its plugin manager.

VERDICT

Gedit	★★★☆☆
Kate	★★★★☆
Sublime	★★★★☆
UltraEdit	★☆☆☆☆
jEdit	★★★★★

Plain ol' text editing

How good are they as just simple editors ?

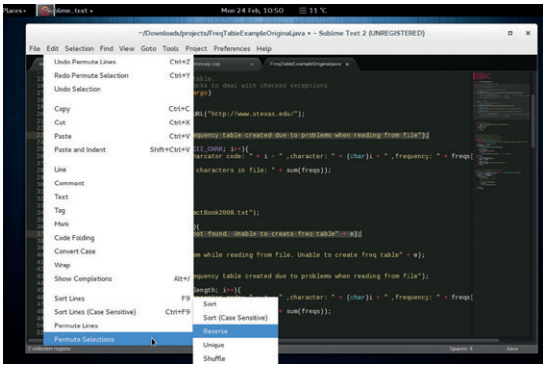
Despite all their powerful extra-curricular activities that might even displace full-blown apps across several genres, there will be times when you just need to use these text editing behemoths to read, write, or edit plain and simple text. While you can use all of them to enter text, we are evaluating them for access to common text-editing conveniences.

Gedit which is Gnome's default text editor, supports an undo and redo mechanism as well as search and replace. It can spellcheck documents in multiple languages and can also access and edit remote files using Gnome GVFS libraries.

You can spellcheck documents with Kate as well, which also lets you perform a Google search on any highlighted text. It's also got a line modification system which visually alerts users of lines which have modified and unsaved changes in a file. In addition, it enables users to set bookmarks within a file to ease navigation of lengthy documents.

Sublime has a wide selection of editing commands, such as indenting text and formatting paragraphs. Its auto-save feature helps prevent users from losing their work. Advanced users will appreciate the regex-based recursive find and replace feature, as well as the ability to select multiple non-contiguous spans of text and act on them collectively.

UltraEdit also enables the use of regular expressions for its search and replace feature and can edit remote files via FTP. One unique feature of jEdit is its support for an unlimited number of clipboard which it calls registers. You can copy snippets of text to these registers which are available across editing sessions.



Sublime Text offers some unique features to manipulate text.

VERDICT

Gedit	★★★★☆
Kate	★★★★★
Sublime	★★★★★
UltraEdit	★★★★☆
jEdit	★★★★☆

OUR VERDICT

Advanced text editors

All the editors in this feature are good enough to replace your existing text editor for editing text files and tweaking configuration files. In fact, chances are they'll even double up as your IDE. These apps are chock full of bells and whistles, and their developers aren't thinking of stripping features, but adding more and more and more.

At the tail end of this test we have jEdit. Not only does it insist on using the proprietary Oracle Java Runtime Environment, it failed to install on our Fedora machine, and the developer doesn't actively engage with its users.

UltraEdit does little better. This commercial proprietary tool focuses on web developers, and doesn't offer anything to non-developer power users that makes it worth recommending over free software alternatives.

On the third podium position we have Gedit. There's nothing inherently wrong with Gnome's default editor, but despite all its positive aspects, it's simply outclassed by Sublime and Kate. Out of the box, Kate is a more versatile editor than Gedit, and outscores Gnome's default editor even after taking their respective plugin systems into consideration.

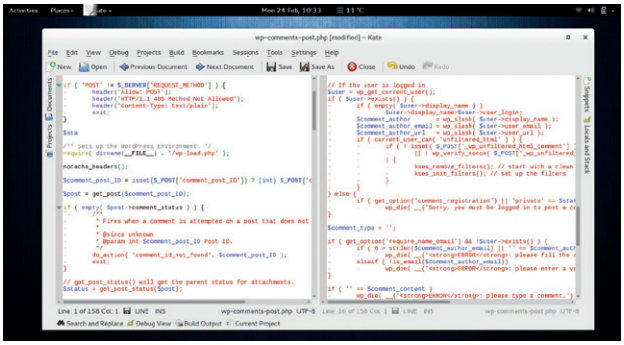
Both Sublime and Kate are equally good. They performed equally well in most of our tests. Whatever ground it lost to Sublime for not supporting macros, it gained for its keyboard friendliness and its ease of use in defining custom keybindings.

Kate's success can be drawn from the fact that it offers the maximum number of features with minimal learning curve. Just fire it up and use it as a simple text editor, or easily edit configuration file with syntax highlighting, or even use it to collaborate and work on a complex programming project thanks to its project management capabilities.

> **"Kate offers the maximum features with the minimum learning curve."**

We aren't pitching Kate to replace a full-blown integrated development environment such as [insert your favourite specialised tool here]. But it's an ideal all-rounder and a perfect stepping stone to a specialised tool.

Kate is designed for moments when you need something that's quick to respond, doesn't overwhelm you with its interface and is just as useful as something that might otherwise be overkill.



Despite being a KDE app, Kate looks good across many desktops.

1st Kate

Licence LGPL/GPL **Version** 3.11

www.kate-editor.org
The ultimate mild-mannered text editor with super powers. Kate is one of the best apps to come out of the KDE project.

2nd Sublime Text

Licence Proprietary **Version** 2.0.2

www.sublimetext.com
A professionally done text editor that's worth every penny – easy to use, full of features and it looks great.

3rd Gedit

Licence GPL **Version** 3.10

http://projects.gnome.org/gedit
Gets it done from Gnome. It's a wonderful text editor and does an admirable job, but the competition here is too great.

4th UltraEdit

Licence Proprietary **Version** 4.1.0.4

www.ultraedit.com
Focuses on bundling conveniences for web developers without offering anything special for general users.

5th jEdit

Licence GPL **Version** 5.1.0

www.jedit.org
A lack of support, lack of working on Fedora and a lack of looking nice relegate jEdit to the bottom slot.

YOU MAY ALSO WISH TO TRY...

The default text editor that ships with your distro will also be able to assist you with some advanced tasks. There's KDE's KWrite and Raspbian's Nano, for instance. KWrite inherits some of Kate's features thanks to KDE's katepart component, and Nano has sprung back into limelight thanks to its availability for Raspberry Pi.

If you wish to follow the steps of Linux gurus, you could always try the revered text editors Emacs and Vim. First time users who want to get a taste for the power of Vim might want to consider gVim, which exposes Vim's power via a graphical interface.

Besides jEdit and Kate, there are other editors that mimic the usability of veteran advanced editors like Emacs and Vim, such as the JED editor and Joe's Own Editor, both of which have an emulation mode for Emacs. On the other hand, if you are looking for lightweight code editors check out Bluefish and Geany. They exist to fill the niche between text editors and full-fledged integrated development platforms.

SUBSCRIBE

shop.linuxvoice.com

Not all Linux magazines are the same

Introducing Linux Voice, the magazine that:

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

12-month subs prices

UK – £55
Europe – £85
US/Canada – £95
ROW – £99

7-month subs prices

UK – £38
Europe – £53
US/Canada – £55
ROW – £60

DIGITAL SUBSCRIPTION ONLY £38

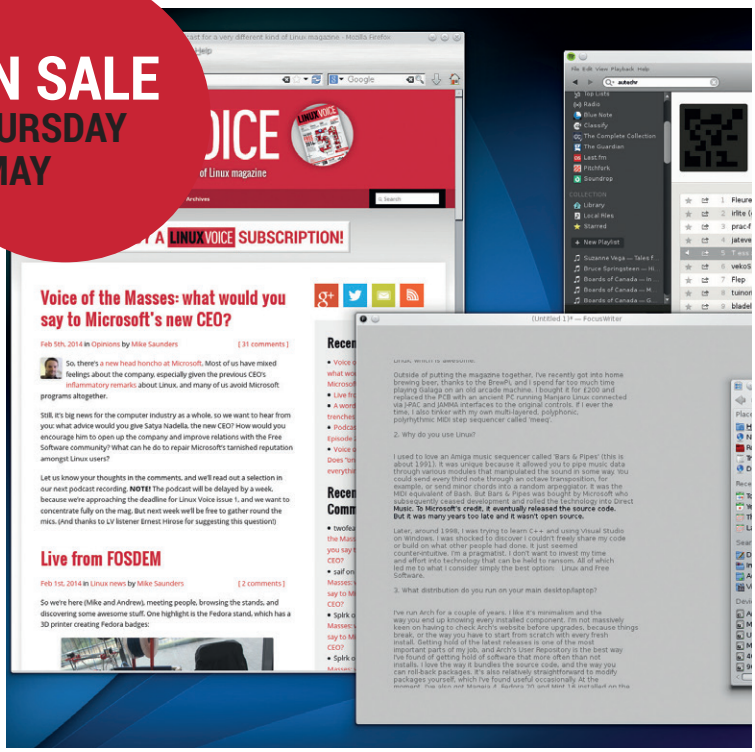
Each month Linux Voice includes 114 pages of in-depth tutorials, features, interviews and reviews, all written by the most experienced journalists in the business.

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

NEXT MONTH IN

LINUX VOICE

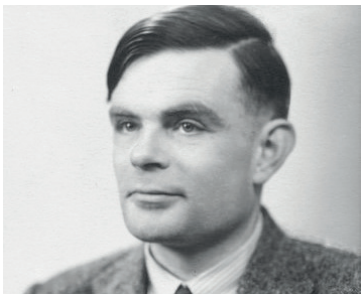
ON SALE THURSDAY 1 MAY



GUI VS COMMAND LINE

The command line will always have its proponents, but you can't beat a nicely polished slice of graphical user interface for getting things done. Or can you?

EVEN MORE AWESOME!



Old Code

Arise Sir Alan Turing, genius coder, computer inventor, Nazi code cracker and 2hr 46min marathon runner. What a man.

Shopping

How hard is it to get a Windows refund? Are there any decent machines out there with Linux installed? Find out before you waste your money

It's full of stars

Use humble Python to identify comets in images taken by the awesome-sounding SOHO (Solar and Heliospheric) satellite. Astronomy for the win.

LINUX VOICE IS BROUGHT TO YOU BY

Editor Graham Morrison
graham@linuxvoice.com
Deputy editor Andrew Gregory
andrew@linuxvoice.com
Technical editor Ben Everard
ben@linuxvoice.com
Editor at large Mike Saunders
mike@linuxvoice.com
Creative director Stacey Black
stacey@linuxvoice.com

Editorial consultant Nick Veitch
nick@linuxvoice.com

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

Printed in the UK by
Acorn Web Offset Ltd

Disclaimer We accept no liabiliy 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

ISSN 2054-3778

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

CODE FOR ALL

Programming isn't just the domain of übergeeks – in the Year of Code, everyone can get involved. Follow our guides and create programs for the desktop, mobile phone and command line.

It wasn't long ago that programmers were viewed by the general public as strange wizards who spent hours alone every day, tapping incomprehensible gobbledygook into black boxes on the screen. But in recent years, the perception of programming as a hobby (and profession) has changed enormously. Today it's cool to hack the Raspberry Pi. Today it's trendy to write software for iOS and Android. Today you can tell someone at the pub that you enjoy programming, and not be looked at like you've just started speaking Tagalog.

We're all hackers on the Linux Voice team, so we support every effort to inspire people to code. It's not all boring, complicated and alien like some people claim; coding can be fun, stimulating and useful for developing future skills.

With this in mind, we wanted to make this issue's cover feature all about coding. But not just as a generic introduction to a language or platform – no, we wanted to show you how to do useful things. So over the next nine pages we have three projects explaining how to make real-world desktop, mobile phone and command line applications. As you'll see, coding is for everyone, and you can code for any platform.

Desktop apps with Python

First step: building the required skills.

To kick off, we're going to use Python, because it's an excellent all-round language which combines highly readable code with oodles of advanced features. Python code tends to be self-explanatory, so it's a great way to dip your feet into programming. We'll start here with a quick overview of the language; if you're already familiar with Python, you may want to quickly skim over this and then turn over the page, where we start using it in our application.

A Python primer

Python is installed by default in most major distributions, and it's an interpreted language, so you don't need to compile your code before running it. In the following examples, save the source code in plain text format as **test.py**. Then run it like so:

```
python test.py
```

Let's start with a very simple program:

```
name = "Linux Voice"
print name + " is the best Linux mag"
```

This demonstrates two aspects of the language: variables and output. In the first line, we create a new variable (like a storage space) called **name** – in Python, you don't need to explicitly state the type of a variable when you create it. In the second line, we print the contents of the variable to the screen, along with another string of characters. Dead easy, right?

Let's look at numeric variables:

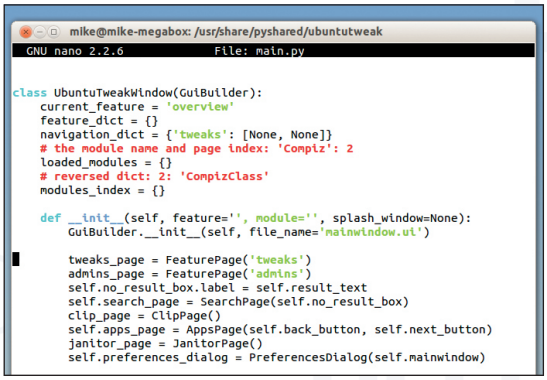
```
x = 1
while x < 10:
    print "x is", x
    x = x + 1
print "Finished!"
```

Here we declare the **x** variable to contain 1 at the start, and then begin a loop. While the contents of **x** are less than 10, we print the contents, and add one to **x**. It's important to note the indentation here: in Python, you use tabs to say which lines belong to a chunk of code. Here we say that the **print** and **x = x + 1** lines belong in the **while** loop, because they're indented. If you removed the tab from the **x = x + 1** line, the contents of **x** would never be incremented in the loop, so the loop would go on forever.

When the loop has finished, the program continues, so no indentation is required. (If you have loops inside loops, you will have multiple levels of indentation.) Next, let's move on to input and comparisons:

```
x = input("Enter a number: ")
if x == 1:
    print "You entered 1"
else:
    print "That wasn't 1"
```

Here we have our **x** variable again, and we call Python's in-built **input** routine, which displays the specified text in quotes. Then we ask Python: if the contents of **x** are 1 after the input, print one thing, and if the contents are not 1, (shown in the code as the **else** statement) print something else. Why the **if x == 1** though – why the double equals signs? Well, it's just to make a very clear distinction from **x = 1** (a single equals sign), which stores a number or line of text within a variable.



Many good text editors include syntax highlighting, to make it easier to read code – here's Nano, for example.

Funky functions

Next, we'll dip our toes into functions. These are self-contained chunks of code that you can use to build bigger programs. You can re-use them with different parameters, to keep your code small and easy to understand. For instance:

```
def multiplier(a, b):
    print a, "multiplied by", b, "is: "
    print a * b
multiplier(6, 7)
multiplier(10, 20)
multiplier(211, 2352)
```

Here, we '**def**'ine a function called **multiplier**, which receives two numbers from the calling program, and stores them in the variables **a** and **b**. This function then prints out the result of multiplication (using the * operator). Note the tab indention again here, and also note that this function is defined at the start of the program, but it isn't executed immediately – Python starts execution in the non-indented part.

So we can call our **multiplier** function with different numbers, as shown in the three lines at the bottom of the code. This is a trivial example (you could just do the multiplications in the main code), but it shows how you can build up programs from self-contained units.

"Python is installed by default in most major Linux distributions."

So, that's the basics of Python covered – now turn the page and let's do something useful.

A kiosk-like web browser

Write your own locked-down, ultra secure web browser.

With our Python skills freshly prepped, let's make a real application. Here we're going to create a simple, and locked-down web browser than can only visit certain web pages and not escape onto the big, bad web. Why would you do this? Well, let's say you're setting up a web terminal for a school, shop or museum, and you want to restrict access to certain places. You could use a normal web browser, load it up with kiosk-like extensions and filtering proxies and hope that it's secure, but clever users may still be able to break out of the restrictions and cause trouble.

With our browser, we have just the bare essentials. And even if setting up a web kiosk isn't high on your list of things to do, it's well worth following this tutorial to see how a simple web browser is implemented.

The code

Here it is – a whole web browser, contained within 35 lines of Python. You probably don't want to type this out by hand, so grab it from **www.linuxvoice.com/code/microbrowser.py**. This browser is set up to view the Debian website at **www.debian.org** and nothing else, but you can of course change that.

```
import gtk, webkit

def goback(button):
        view.go_back()

def navrequest(thisview, frame, networkRequest):
        address = networkRequest.get_uri()
        if not "debian.org" in address:
                md = gtk.MessageDialog(win, gtk.
DIALOG_DESTROY_WITH_PARENT, gtk.MESSAGE_INFO, gtk.BUT-
TONS_CLOSE, "Not allowed to leave the site!")
                md.run()
                md.destroy()
                view.open("http://www.debian.org")

view = webkit.WebView()
view.connect("navigation-requested", navrequest)

sw = gtk.ScrolledWindow()
sw.add(view)

button = gtk.Button("Back")
button.connect("clicked", goback)

vbox = gtk.VBox()
vbox.pack_start(button, False, False, 0)
vbox.add(sw)

win = gtk.Window(gtk.WINDOW_TOPLEVEL)
win.set_size_request(800, 600)
win.connect("destroy", gtk.main_quit)
win.set_title("Linux Voice browser")
win.add(vbox)
win.show_all()

view.open("http://www.debian.org")
gtk.main()
```

Here it is: our funky mega skillo web browser, which we can restrict to wherever we want, written in just 35 lines of Python. How cool is that?

Now, we're not going to write the entire HTML, CSS and JavaScript rendering engine ourselves – that would take months of hard effort and fill many issues of the magazine. No, we'll leave that to WebKit, an extremely capable rendering engine used in several notable browsers such as Chrome/Chromium and Safari. (Well, Chrome now uses the Blink rendering engine, but this is based on WebKit.)

There's a great Python module to interface with WebKit, which we're using here: you'll find it in the **python-webkit** package in Debian and Ubuntu-based distros. You'll also need **python-gtk2** for the interface. So let's step through the code:

```
import gtk, webkit
```

This is simple enough – it just tells Python that we want to use the GTK module to provide the GUI widgets, and the WebKit module for the rendering engine. Then we have two functions:

```
def goback(button):
...
def navrequest(thisview, frame, networkRequest):
...
```

We'll come back to these in a moment, because first

we need to set up some code to use them. So execution of the program begins here:

```
view = webkit.WebView()
view.connect("navigation-requested", navrequest)
```

This is the heart of the program. We create a new object called **view** (an object is like a variable, but it can store and do a lot more), which is a WebKit WebView. Put simply, this is a plain web browser view – but with no buttons, no surrounding window, or anything like that. It's just a canvas to render web pages inside.

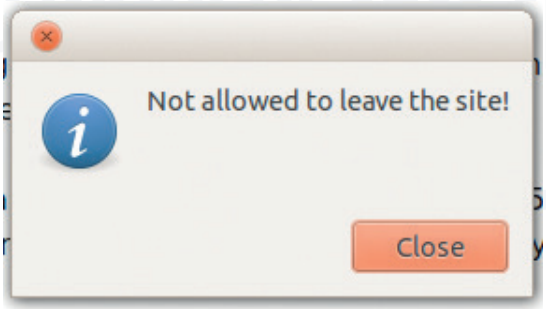
The second line of this snippet is crucially important, and demonstrates a callback function. We tell our web page view that if a navigation event occurs (ie the user clicks on a link), we should call the **navrequest** function as included at the start of the code. This function retrieves the address of the clicked link (**networkRequest.get_uri**) and checks to see if **debian.org** is contained in the address. If not, we show a dialog box and go back to the Debian home page. (Yes, this doesn't make it 100% impossible for people to escape **www.debian.org**, but you could narrow down the allowed links even further with regular expressions – that's beyond the scope of this guide though!)

So, we have a web browsing pane which checks links as they're clicked. Next are these lines:

```
sw = gtk.ScrolledWindow()
sw.add(view)
button = gtk.Button("Back")
button.connect("clicked", goback)
```

As mentioned, the WebKit view isn't attached to anything yet – it just exists in memory somewhere and that's it. Here we attach it to a GTK scrolling window so that users have scroll bars to move around in the page. We create a new **ScrolledWindow** and add the **view** object to it.

Then we create a new GTK button with the label **Back**, and also attach it to a callback function we wrote earlier: **goback**. So whenever the user clicks this button, the function **goback** is called. In that function, we tell the view to step back a page (**view.go_back()**) and return to the main code.



And here's what happens if users try to stray beyond the limits, as we defined in the **navrequest()** function.

Now we have to pack the button and WebKit view into a single space, for which we use a vertical box widget in GTK:

```
vbox = gtk.VBox()
vbox.pack_start(button, False, False, 0)
vbox.add(sw)
```

Then the following six lines, beginning with **win**, create a new application window, set its size, say what to do when the close button is clicked and set a title. They also add the vertical box widget to the application window and make sure that all of the widgets inside it are visible. And the last two lines are:

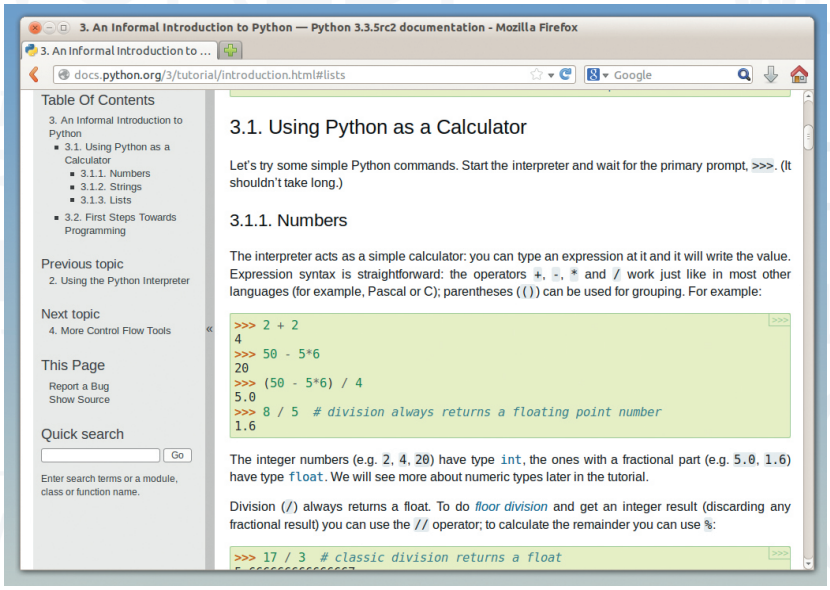
```
view.open("http://www.debian.org")
gtk.main()
```

Here we tell the WebKit view to open a specific page, and run GTK's main event loop (where it watches for button clicks and window operations – we leave it alone from here).

And that's it! We've crammed a lot in here, so if you have any questions, check out the websites for Python GTK (**www.pygtk.org**) and Python WebKit (**https://code.google.com/p/pywebkitgtk**). If you need further help, post on our forums at **http://forums.linuxvoice.com** and we'll do our best to answer.

"There's a great Python module to interface with WebKit, which we're using here."

Want to delve into more advanced Python topics? See **http://docs.python.org** for a wealth of tutorials.



Making Python apps directly executable

It's easy to run our browser from the command line by typing **python microbrowser.py**, but what if you want to make it directly executable – so you can run it by simply double-clicking on it? The trick is to add this to the top of the file:

```
#!/usr/bin/env python
```

Now make the file executable (eg **chmod +x microbrowser.py**) and run it (**./microbrowser.py**). This extra first line tells the operating system which interpreter should be used with the following code, so you don't need to specify Python manually at the command line. In your kiosk setup, you can easily trim down a desktop or window manager to the bare essentials, and add a single launcher pointing at **microbrowser.py** somewhere on your system.

Mobile Linux

Don't rely on the app store for software – create your own.

Over the last few years, Linux has taken over the mobile computing marketplace. Android is hugely popular, and there's also Amazon's Fire OS, Firefox OS, Sailfish OS, Tizen, Bada and soon there'll be Ubuntu Touch as well. Developing for mobile platforms isn't like desktop Linux, where the same code is just repackaged for different distros. This allows developers to make their apps fit the look and feel of the OS, but it also means that you have to spend a long time developing for the different devices.

Fortunately, it doesn't have to be this way. It is possible to maintain a single codebase for use across all the Linux-based mobile platforms (and a few non-Linux ones like iOS and Windows Phone as well). This way is Apache Cordova.

Apache Cordova is a framework that enables you to develop in HTML and JavaScript, then package it up for each different environment.

While it may once have been suitable only for displaying web pages, and perhaps a little interaction, JavaScript has grown in to a powerful programming environment. If you don't believe us, have a look at these examples and see for yourself:

- The Unreal Engine 3 has been ported to JavaScript and you can explore Epic Citadel in your browser: **www.unrealengine.com/html5**.
- Tearable cloth: **http://codepen.io/suffick/pen/KrAwx**.
- Freerider II: **www.freeriderhd.com/t/1016-layers**.

With the canvas HTML 5 element, you can make 2D JavaScript graphics as complex as you like, and with WebGL, you can even harness the power of the device's GPU to create accelerated 3D graphics. WebRTC can be used to set up a communication channel between two browsers, and Webaudio helps you add sound to your creations. We should point out that not all of these features are as yet possible with Cordova on all devices (even new ones), but it's only a matter of time.

The performance of JavaScript has long been considered a problem for web app development, but in recent years, this has improved dramatically. Now, well-written JavaScript should perform at about half the speed of natively compiled code. This means that it's still not quite there for some high-performance applications, but it should be fine for most cases.

The people at Mozilla know more than most about the power of HTML and JavaScript, and they believe in it so much that they built an entire phone operating system built around it.

Another great advantage of the HTML and JavaScript approach is that it makes it really easy to get started. You can create simple pages in point-and-click HTML editors, then progress onwards as you learn more about programming and the environment.

PhoneGap

Cordova is closely related to Adobe PhoneGap. In fact, they're so closely related that they're often mistaken for one another. Officially, PhoneGap is an implementation of Cordova, but it doesn't add much to the core release in the same way a distribution of Linux adds a lot of software around the kernel. For now we prefer to build on a framework released by the Apache Software Foundation than one developed by Adobe.

Don't forget to consider the size of your computer screen when selecting what type of device to emulate. Modern phones have a lot of pixels (often more than monitors).

First build

To get started, you'll need the appropriate SDK for the platform (or platforms) you're developing for, and the appropriate version of Cordova (a full list can be found at **cordova.apache.org/#download**). In this tutorial we're going to use Android, as it's the most popular mobile Linux, but you shouldn't have any problem transferring the work to a different platform such as Ubuntu Phone or Firefox OS. Sailfish OS should be able to run Android apps, but we haven't been able to test this particular app.

You'll need the SDK for every environment you're developing for. For Android, you can get it from **http://developer.android.com/sdk**. You'll get a ZIP file that you can extract. You need to add the **sdk/platform-tools** and **sdk/tools** directories from this ZIP to your path. In the author's environment, this was with the following command, though you'll have to change it depending on where you unzip the SDK:

```
export PATH=$PATH:/home/ben/Downloads/adt-bundle-linux-x86-20131030/sdk/platform-tools:/home/ben/Downloads/adt-bundle-linux-x86-20131030/sdk/tools
```

Cordova pugins

To access phone features, you'll need to use plugins. The standard ones that come as part of Cordova are: Accelerometer, Camera, Capture, Compass, Connection, Contact, Device, Events, File, Geolocation, Globalization, InAppBrowser, Media, Notifications, Splashscreen and Storage. There's also a good selection of plugins available at **http://plugreg.com**, should the standard ones not do everything you need.

You'll have to re-run this every time you restart your computer unless you add it to the **.bashrc** file in your home directory.

Cordova is a **node.js** application, and you'll need both **node.js** and **npm** for it to run. On Ubuntu and derivatives, this can be done with the following code. If you're using a different distro, check the available packages:

```
sudo apt-add-repository ppa:chris-lea/node.js
sudo apt-get update
sudo apt-get install npm nodejs and
npm config set registry http://registry.npmjs.org/
sudo npm install -g cordova
```

Cordova works with a specific directory structure. To create a new project directory and appropriate subdirectories run **cordova create myProject**, where **myProject** is the name of the new project. Now you can move into the new directory with **cd myProject**.

Projects start off without any platforms. You can add as many as you like provided you have the SDKs installed, and Cordova will manage the builds for you. We'll just add Android with:

```
cordova platform add android
```

And you can compile the example code (that each project is created with) using:

```
cordova build android
```

In order to test your app, you either need an Android device, or to use an emulator. To create a new emulated device, open the version of Eclipse that came bundled with the Android SDK. Go to File > New > Other > Android > Android Project From Existing Code and select the **platforms/android** folder from your project's directory. Then do to Window > Android Virtual Device Manager and create a new virtual device. Once this is set up, you can run the project from the command line with:

```
cordova emulate android
```

Some of these work far better with a physical device than an emulator, so if you have an Android phone or tablet, it'll be easier to follow the rest of this tutorial on that. The method for doing this varies depending on the version of Android you have. Visit **http://developer.android.com/tools/device.html** for more details. Once you've set up your phone, and connected it to your computer via the USB cable, you can load and run the app with:

```
cordova run android
```

In its basic state, this enables you to package up HTML and JavaScript for phones. In essence, it allows you to create off-line websites. This certainly has its uses, and many apps are nothing more than this.

Harness phone-specific features

However, to be a true phone app, it should have access to more of the phone's features, such as the GPS, accelerometer or filesystem. These aren't available through normal JavaScript, but Cordova allows plugins that expose certain features to its JavaScript API. Take a look at the boxout above for the standard plugins.

As an example, we're going to create a simple weather forecast app. It'll get the current location, then display a weather forecast for the area. We'll also add the ability to share it using social media, as apparently all good apps do this.

To start with, we need to get the HTML and JavaScript to grab a forecast based on latitude and longitude. Since everything is in the web technologies, it's far easier to test it out using web development tools than the Android-specific ones. Once everything's working properly in a browser, you can then transfer it to Cordova and check it in Android.

We'll start with a really simple HTML doc that just grabs a forecast for London. Change the **www/index.html** file so that it contains:

```
<!DOCTYPE html>
<html>
<head>
<title>Weather Forecast</title>
</head>
<body>
Forecast:
<br>
<iframe id="forecast_embed" type="text/html" frameborder="0" height="245px" width="245px" src="http://forecast.io/embed/#lat=51.5072&lon=0.1275&units=uk"> </iframe>
</body>
</html>
```

Save this as **index.html** in the **www** folder of your app. You could run this using Cordova on your phone or an emulator, but it's easier at this stage to open it in your normal web browser.

51.5072, 0.1275 are the coordinates we'll use (this is in London). This grabs an iframe with the current forecast from **forecast.io**. In order grab the forecast for the current location, all you need to do is create an iframe with the right latitude and longitude.

> **"Another advantage of HTML and JavaScript is that it makes it really easy to get started."**

Writing iframes in JavaScript is easy, since you can manipulate the HTML inside an element. All you need to do is create a ** ** that you can put the iframe inside. Now we'll add the ablity to switch the location between London and New York. First change the code between the **<body></body>** tags to:

```
Forecast:
<div id="location"></div>
<div id="forecast">Select Location</div>
<br>
 <button onclick="getForecast(51.5072, 0.1275)">London
Forecast</button>
 <br>
 <button onclick="getForecast(40.6700,73.9400)">New York
Forecast</button>
<br>
<div id="getlocalforecast"></div>
<div id="forecastDetails"></div>
```

This has two divs with different IDs that we'll use now, and some more that we'll use in a bit. We'll use JavaScript to update them to what we need them to be. The two buttons call the **getForecast(latitude, longitude)** function that we'll now define.

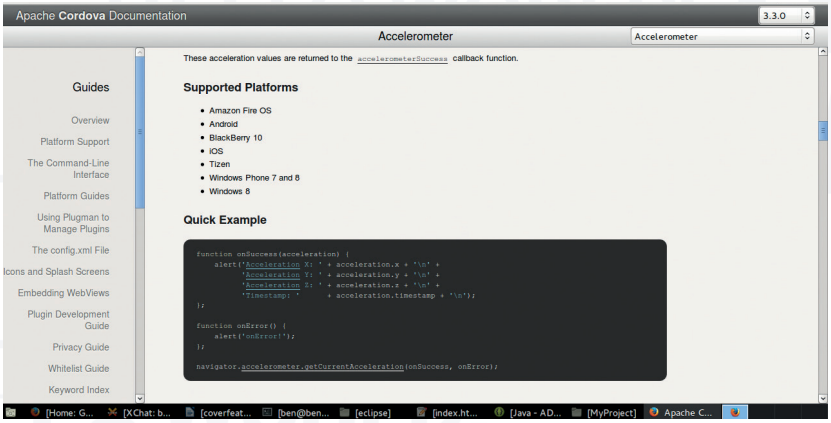
Add the following just before the **</head>** tag:

```
<script type="text/javascript">
function getForecast(latitude, longitude) {
   var element = document.getElementById('location');
   element.innerHTML = 'Latitude: ' + latitude  + '<br />' +
      'Longitude: ' + longitude + '<br />';
   window.iframeurl='http://forecast.io/embed/#lat=' + latitude +
      '&lon=' + longitude + '&units=uk';
    showSimple();
   }
function showSimple() {
   var weather = document.getElementById('forecast');
   weather.innerHTML = '<iframe id="forecast_embed"
type="text/html" frameborder="0"' +
      'height="245px" width="245px" src="' + window.iframeurl +
'"> </iframe>';
   }
</script>
```

This splits the execution up into two stages. The first sets the contents of ** **, and the variable **window.iframeurl**. By defining this variable as attached to **window**, it makes it available to all our functions, rather than just local to the current function. The same effect could have been achieved by using a global variable.

The second function sets the ** ** to be an iframe with the appropriate location. The reason we've split this up into two functions will become clear later. Again, you can test this out in a web browser and it should work fine.

The documentation at **http://cordova.apache.org/docs/en/3.3.0/** is a great place to find help. It has guides for all the standard plugins including comprehensive code samples.



Signing apps

Cordova can build a final version of your app using the **--release** option to the build command. However, this won't install on any phone until it's been signed. You can create a key for signing it yourself, so this isn't a restriction on distributing your software. There are details of how to do it here: **http://developer.android.com/tools/publishing/app-signing.html**.

You can distribute your app without an app store if you want. Just send the **.apk** file to people and (as long as they have sideloading enabled) they can install it themselves. Of course, you can put your app on the main Google Play store if. You'll find details about how to do this here: **http://developer.android.com/distribute/googleplay/publish/preparing.html**.

Google Play isn't the only Android app store though. If you open source your app, you may wish to add it to the F-Droid store. Take a look at **www.f-droid.org** for details.

Not just a website!

Now let's add the phone-specific stuff. Cordova uses plugins to add access to different features, so in order for our app to be able to access the location, we need to use the Geolocation plugin. This is done by running the following command in the root directory of the web app:

```
cordova plugin add org.apache.cordova.geolocation
```

This will add it to every platform you have registered as long as the plugin works on that platform.

In order to access the Cordova features, you need the **cordova.js** script, so add the following line just below **</title>**:

```
<script type="text/javascript" charset="utf-8" src="cordova.
js"></script>
```

With this in place, you can add the following functions inside your main **<script>** tag:

```
document.addEventListener("deviceready", onDeviceReady,
false);
function onDeviceReady() {
   localforecast = document.getElementById('getlocalforecast');
   localforecast.innerHTML = '<button onclick="getLocal()">Local
Forecast</button>';
   }
function getLocal() {
   navigator.geolocation.getCurrentPosition(onSuccess,
onError);
   }
   function onSuccess(position) {
   getForecast(position.coords.latitude, position.coords.
longitude);
   }
function onError(error) {
   alert('code: '    + error.code    + '\n' +
      'message: ' + error.message + '\n');
   }
```

The first line listens for the **deviceready** event. This tells it to run the function **onDeviceReady** once the

app is running properly. We've added this to stop people trying to get a local forecast too soon.

The function **getLocal** can just call **navigator.geolocation.getCurrentPosition()**. We've passed it two parameters: the first is the name of the function to call if it succeeds in getting the location; the second is the function to call if there's an error.

onSuccess passes the returned values on to **getForecast()**, while **onError()** displays the error message as a JavaScript alert.

With all this entered and saved, it's ready for its first proper test. To compile and run it, enter the following terminal commands in the app's root directory:

```
cordova build android
cordova run android
```

If you've got your phone attached, this will send it across and open it on your device, otherwise it'll start the emulator.

You've just created a phone app! It's quite limited, but accesses one of the phone's features. Since all good mobile applications have social features, we'll add this facility now. We won't make specific links to social media, but use the phone's features to share the forecast with other applications. The user can then pick how they want to share the forecast.

Engage Twitbook

As you may have guessed, this feature comes from another plugin, but this time it's one that's not part of the main Cordova release. You can add plugins straight from Git, so in a terminal in the app's root directory, enter:

```
cordova plugin add https://github.com/leecrossley/cordova-plugin-social-message.git
```

As with the previous plugin, this exposes more JavaScript functions that we can access. In this case, it's **socialmessage.send()**. Using this, you can interact with the other apps on the phone. Add the following function inside the **<script></script>** tags:

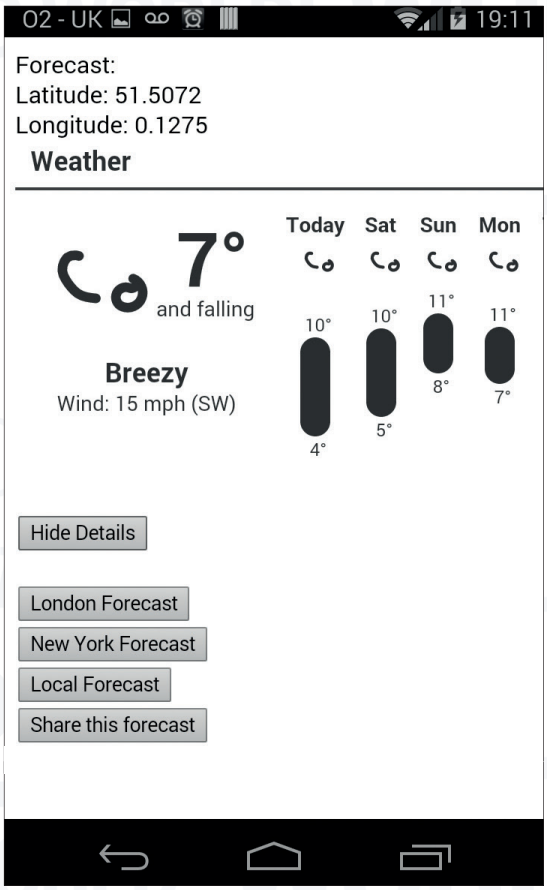
```
function share() {
  var message = {
    subject: "Weather Forecast",
    text: "Check out my local forecast",
    url: window.iframeurl
    }
  window.socialmessage.send(message);
  }
```

You'll also need a button in the body of the HTML to access it. However, you can't share the forecast until it's received, so the button should only appear once there's a forecast. The easiest way to do this is by adding the lines:

```
    var weatherDetails = document.getElementById('forecast Details');
    weatherDetails.innerHTML = '<button onclick="share()">' + 'Share this forecast</button>';
```

to the end of the **getForecast()** function.

We're almost done with our app now. The last little feature we'll add is the ability to show a simple or



The app in action. Unfortunately, that's the best weather we've had in months.

detailed forecast. Fortunately, **Forecast.io** does most of the hard work on this. The only thing we have to do is change the size of the iframe.

You'll need to adjust the **showSimple()** function and add **showDetails()** as per the following:

```
function showDetails() {
  var weather = document.getElementById('forecast');
  weather.innerHTML = '<iframe id="forecast_embed" type="text/html" +
    'frameborder="0" height="245px" width="500px" src="' + window.iframeurl +
    '"> </iframe><br><button onclick="showSimple()">Hide Details</button>';
  }
function showSimple() {
  var weather = document.getElementById('forecast');
  weather.innerHTML = '<iframe id="forecast_embed" type="text/html" +
    'frameborder="0" height="245px" width="245px" src="' + window.iframeurl +
    '"> </iframe><br><button onclick="showDetails()">Show Details</button>';
  }
```

> **"Cordova uses plugins to access different features – we need the Geolocation plugin."**

Of course, it still looks a bit plain, and you could add many more options, but this isn't a tutorial on creating the perfect weather forecasting app, it's a tutorial on getting started with mobile Linux development. It's up to you to decide what to do with it now.

Programming the command line

Automate everything, then sit back and relax as your computer takes care of itself.

So far we've talked about programming in terms of making new software. However, programming can also be a way of linking together existing software to automate tasks. In this way, you don't create anything that you didn't have access to before, but you make it much easier to use. Let's take a really quick example. Suppose you're a writer, and you save all your work in ODT files. These files are scattered about your home directory (because most writers aren't organised enough to keep their files in one place), and you want to do a full backup of all your writing.

There are many ways you could do this. One of the easiest is to create a simple program that searches for all the files and copies them to a remote computer.

Bash is the shell that most Linuxes use, and while many users know it only as a command line environment, it's also a programming language in its own right. We can use it to link up a series of Linux commands to execute based on the information that other commands provide. In this example, we'll use the command:

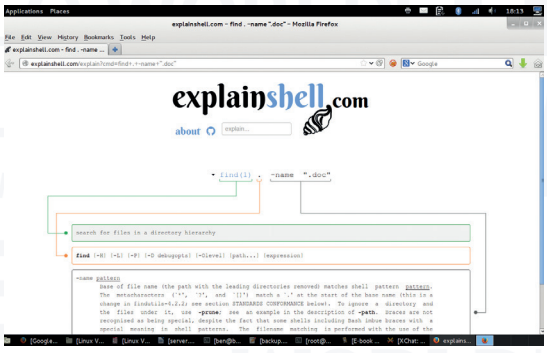
```
find /home/ben -name "*.odt"
```

To find all the required files. Not surprisingly, the **find** is command for finding things, and is far more powerful than this command shows. Using other options, you can find files based on the time they were created, the time they were last modified, and a whole host of other things. See the man page (type **man find** in a terminal) for more details.

We'll then copy all the files into a backup folder (which could be on an external drive). The bash code to do all this is:

```
#!/bin/bash
find /home/ben -name "*.odt" | while read f;
do
  cp -f "$f" /home/ben/backup
done
```

You'll need to change **/home/ben** to the location of your home directory.



Explain Shell (**www.explainshell.com**) is a tool for linking bash commands to their help text.

Lets take a look at this in detail. The first line is called a shebang, and it tells the computer that this is a Bash script and that it should be executed with the command **/bin/bash**.

The second line does two things. First, it executes the **find** command, which we explained above; then it pipes the output of the command into a **while** loop. Piping (which is done using the character **|**) is an essential feature of Bash programming, and it can also be done on the command line. It just tells the system to take the output of one command and feed it into the next. As another example, if you're using a terminal and you're in a directory with loads and loads of files, it sometimes doesn't work very well if you just run **ls** to list them (the filenames can go off the top of the screen). Instead, you can pipe the output into a text viewer such as **less** with:

```
ls | less
```

This allows you to scroll up and down through the list of files. You can also do it for other commands that produce a lot of output.

Digression over

Back to our backup script though. In this case, the program outputs the result of the **find** command into **while read f**. This slightly cryptic statement starts a loop for every line in the output and tells it to store the line in the variable **f**. In other words, everything between **do** and **done** is executed once for every line in the output of the find command, that is:

```
cp -f "$f" /ben/backup
```

The **$f** tells Bash to insert the line output from the **find** command here. It's in quote marks because otherwise filenames with spaces in them will cause problems.

This is a really simple example, but it shows how you can build up scripts in Bash. The two main ways of combining commands are piping output, and running loops over multiple lines. With these two techniques, you can combine all the command line tools in Linux into your own powerful scripts.

Before running it, you have to make the backup directory with:

```
mkdir ~/backup
```

If you save this program as **backup.sh**, you can run it from the command line with:

```
bash backup.sh
```

As long as you are in the same directory that you saved the file. Alternatively, if you make it executable with the command:

```
chmod a+x backup.sh
```

you can run it with:

```
./backup.sh
```

Sometimes, it's not enough just to send the output of one command straight into another. Sometimes

you need to make a decision based on the output that's being processed. For example, what if you didn't want to copy all files straight into the backup directory? What if you wanted to sort them and put different files in different places?

In the next example, we'll find all LibreOffice Writer and Calc files (ODT and ODS respectively) and all MS Office Word and Excel files (DOC/DOCX and XLS/XLSX respectively), and split them into word processor and spreadsheet folders.

This can be done with the following:

```
#!/bin/bash
find /home/ben \( -name "*.odt" -o -name "*.ods" -o -name
"*.doc" -o -name "*.docx" -o -name "*.xls" -name "*.xlsx" \) |
while read f;
do
  if [[ $f == *.odt ]] || [[ $f == *.doc ]] || [[ $f == *.docx ]]
  then
    cp -f "$f" /home/ben/backup/wordprocessor
  fi
  if [[ $f == *.ods ]] || [[ $f == *.xls ]] || [[ $f == *.xlsx ]]
  then
    cp -f "$f" /home/ben/backup/spreadsheet
  fi
done
```

The Bash **if** command allows you to execute a code block only if a particular condition is true. It's both hugely powerful and quite complex. Used like this (with **[[string1 == string2]]**) it matches filenames, and you can use asterisks in the same way you can at the command line, so ***.doc** matches any file that ends with **.doc**. The **||** is used to group multiple conditions together so that the code block is run if any one of them is true.

Running automatically

Writing scripts like this can really simplify general tasks like backing up data, but wouldn't it be great if you could automate running them as well?

Almost all versions of Linux (and, for that matter, Unix in general) come with a tool called **crontab**. The name doesn't give much away, but it's for scheduling tasks to run at certain times (it's named after Chronos, the Greek god of time). There are only really two options that you need to know: **-e** and **-l**. The first is used to edit scheduled commands, and the second is used to list them.

So, to set up our script to back up commands, run:

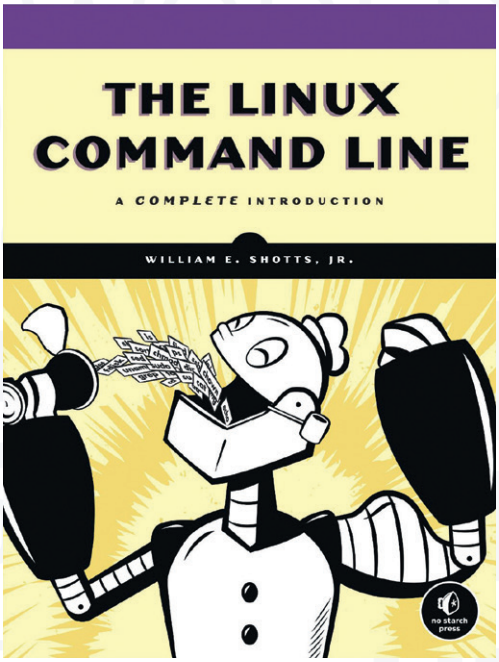
```
crontab -e
```

This will start a text editor (usually either Vim or Nano). If blank lines at the bottom of the file are displayed as **~**, then it's probably opened in Vim. This is a powerful editor, but it can be quite confusing if you haven't used it before. If you want to switch to something easier, exit Vim by pressing Escape, **:**, Q and **!**. They you can tell the system to use Nano instead by running:

```
export EDITOR=nano
crontab -e
```

Books

Bash scripting is incredibly, but can also get quite technical. If you're interested in taking it further, there are loads of good books on the subject. *The Linux Command Line* and *The Advanced Bash Scripting* guide are both excellent choices. The latter is a bit more technical than the former. What's more, the both have e-book versions that are free as in zero cost and free as in speech. You can get them from **www.linuxcommand.org/tlcl.php** and **www.tldp.org/LDP/abs/html/index.html** respectively.



If you prefer your books in paper form, see *The Linux Command Line* website for purchasing options.

You should now see 'GNU nano' In the top-left corner. Depending on your distribution, you may find that you already have some scheduled tasks, you may have a blank file, or you may have some lines that start with a **#** (these are comments).

"Sometimes it's not enough just to send the output of one command into another."

To schedule tasks, you need to add a line to this file that tells it what to run and when. Schedules are broken up into five parts, each of which can be a number or an asterisk. For example:

```
0 2 * * * /home/ben/backup.sh
```

The five scheduling segments represent the minute of the hour, the hour of the day, the day of the month, the month of the year, and the day of the week. An asterisk means 'every'. The above line will run the backup script at 2.00am every day. If you were more cautious, you could run it every hour with:

```
0 * * * * /home/ben/backup.sh
```

Or you could run it twice a day with:

```
0 0,12 * * * /home/ben/backup.sh
```

EDUCATION EDUCATION EDUCATION

Teaching the world to code is a noble goal, but how is it going to work in practice?

Two years ago, when the Raspberry Pi launched, it was with the intention of improving IT education in the UK. Since then more powerful, better connected or cheaper boards have come onto the market, but the Pi retains its position as the white knight of ICT teaching.

Why? Because of the community of users that has grown up around it. To find out more we travelled west to Manchester, venue for the second annual Jamboree – a festival of educators, makers and messer-abouters focussed on highlighting how engaging the Pi can be. There, we met 75% of the Raspberry Pi Foundation's education team – Ben Nuttall, Clive Beale and Carrie Anne Philbin – to discuss IT teaching in the UK.

LV So, Raspberry Pi education team, we were saying earlier that the obvious place to start is with the UK government's Year of Code initiative, but that seems far too negative to begin with!

Carrie Anne Philbin: Yeah, but there's so much to say about it!

Clive Beale: I mean the main issue is how the media portray computing, which is a brilliant, creative, rigorous, hard, challenging, fun thing, and they just reduce to this 'code'. You start to hear things like 'We must learn to code' and 'You better learn code or you're a rubbish teacher.' Which of course is not the case at all; it's so much more than that. And so the teachers are running around now thinking the sky's falling because they think that by September 2014, if they can't code, then they've failed and they'll think their kids have failed, and it's a really bad message. There hasn't been a simple message to say this is not the case. You can get out into the playground with some chalk and make a maze and do some computing, to teach them how to use a computer. No politician has stepped in to say this is not the case and they haven't asked the teachers who would tell them this isn't the case. And that's the problem. Where are the teachers? Where are the people who are actually teaching?

LV Carrie Anne, you've only just left teaching to join the Raspberry Pi Foundation – you must have some insights into the difference between the reality of teaching in schools and how the latest government wheeze imagines it to be?

Carrie Anne: It feels like a lot longer than two months, it feels like an eternity! Teaching is so fast-paced, in that you're seeing the results of what you're doing in class straight away, whereas being out of the classroom for the past two months and working for the Foundation, I can't actually see the impact I'm having, but obviously people are talking to me and saying I am having an impact. It's just very different to what I was seeing before.

Clive always says there's a massive difference between teaching and being a teacher. There are a lot of people we talk to who run Raspberry Jams, workshops, Coder Dojos and that kind of thing, and they always say 'Oh, it's really easy, you just do this, this and this.' And that's great, but you've got kids coming to you who *want* to learn this stuff. Imagine a class where you've got a bunch of kids who aren't interested or engaged by this subject and actually teaching is much harder...

Clive: Maybe 15-year-olds on a Friday afternoon...

Carrie Anne: ...yes, a six-period day, end of the week...

Clive: ...Ofsted saying, 'Why haven't your kids developed after 20 minutes in your lesson? Why haven't they progressed?' It's very different.

So you have some idea of how things should be done because of your recent experience. Are we, in general as a society, doing the right things to foster the next generation?

Carrie Anne: I think so. I think what's been really nice about the Raspberry Pi community is that is gives back to the community. So there are experts, there are people who love what they do, who are reaching out to teachers and reaching out to children by running workshops and clubs and things. And it's actually that collaboration that produces the best materials and produces the best way to move forward with the new curriculum. I mean, the work I did as a teacher producing the Sonic Pi was a team of work. That was because I worked with – yes, an academic – but he was an expert. He wanted to develop a teaching environment that I could use with my students to teach tech-based programming in a fun and engaging way, that engaged both genders and engaged both low- and high-ability students. It's a tech-based programming language, which is important at Key Stage 3 [pupils between the ages of 11 and 14], where you need to not just be able to teach Scratch, you need to teach a tech-based language that's a nice bridge between Scratch and something like Python, which we can teach later on. So, yeah, I think we are moving in the right direction. It would be nicer if the powers that be...

"The community and third parties are doing more to push ICT education along."

Clive: It would be nice to see [what happened with Sonic Pi] as a microcosm of how these things actually happen. So if an academic has a brilliant idea and they're very good at what they do, they should feel that they can come to a teacher and say 'How can we make this useful in the classroom?', 'How can we get assessment in there?', which schools need, frankly, 'How do we make it robust?', 'How can we test it?'.

Isn't that a weird idea, to actually ask the people that teach how we should do that? It hasn't happened really. But, yes, as Carrie says, we're going in the right direction, certainly. The community and third parties are doing more to push it along.

From someone outside a little bit, it kind of looks like a community has sort of

Carrie Anne Philbin's book, Adventures In Raspbery Pi, is reviewed on page 26.

spontaneously developed around the Raspberry Pi. Has this suddenly mushroomed, or has it always been there and it's just become more obvious now?

Carrie Anne: I think it's always been there. I was a teacher, so when Raspberry Pi first came out I got one. I thought, this is brilliant! Someone's developed something for education. A Linux box that we can use in the classroom. You can mess about with it, it's cheap, it's brilliant!

And then I was like, right, so where are the resources to go with it? Ah, there aren't any. So where can I go and find some? The first obvious place was Raspberry Jams. There are people running events where they're doing stuff. So I thought I'll go along and speak to some people, and see what's available. And it was through this that I met people to work with, and they'd formed that themselves, the enthusiasts from throughout the community around the Raspberry Pi.

Clive: It's been a focus, hasn't it? There's been a lot of people sort of hanging around, saying 'Look, I like tinkering, I like messing, I like coding, I like making', and this thing appeared that was cheap and cheerful and fantastic to play around with. It was waiting to happen really.

LV Was there anything equivalent to Raspberry Jams before the Raspberry Pi came along to bring people and teachers together?

Carrie Anne: Well, teachers tend to generally get together through things like TeachMeets and through Twitter, and those kinds of chat tools. There are ways that you can get together, but that's more talking about teaching practice. Like the different ways you can use a sentence. It's very teachery, it wasn't specific to teaching computing.

Clive: I think that hardware-wise, the Raspberry Pi was in the right place and the right time.

Ben Nuttall: The Jams have done a really good job of bringing people together from all walks of life. The people, like myself, who were attending user groups and who are interested in tech and really passionate about it, have got a chance to share that interest with the wider community. There were families coming in, teachers coming in, and they were just sharing what they were doing and the skills they already had, and I was already programming in Python and things like that just on the desktop, and the Pi came along and it opened up this way of plugging into the real world and all the other things the Pi brings with it. Just being about to use those skills and pass them on, I got involved in education through that.

LV So you weren't a teacher before then?

Ben: No, I was a software developer.

Carrie Anne: This is what's great about the education team at Raspberry Pi. It's 50% ex-teachers and 50% software developers. We need people like Ben and Dave (Honess).

Ben: Yes, some people have ideas for things and think this could be an engaging exercise, but they may not know exactly how to deliver it, or how it's going to work. They might not know how exactly a teacher is supposed to produce something to use that, but they have an end goal and working with someone else can help achieve that.

Clive: Yeah, you're right, the real key is the mix. So you're getting teachers and engineers and developers and families. Before, they might have been on a Linux user group, they might have been a teacher group, and you're just bringing a bunch of different people together and that just (to use a horrible word!) synergises stuff.

(Everyone LOL)

Clive: Yes, I did it! I said synergises! I'm buying a copy of the magazine now.

Carrie Anne: There's something that comes about from getting all those different types of people together. It breeds this wonderful learning environment that you cannot reproduce. Like, Ben was running a Picamera workshop this morning. So that was run by people who run Jams who are from industry. And what was really nice is that there were teachers and there were people who had come for the Jamboree from industry that were helping the teachers do stuff. And there was this environment that was like, it's OK to not know something, it's OK to ask a question, it's OK to get it wrong and make mistakes. And that's really powerful, because sometimes teachers are afraid perhaps of saying they don't know.

"You're bringing a bunch of different people together and that just synergises stuff."

Ben: So at this workshop, we gave people an intro to building a real application around the camera. So it's not just 'Oh, there's a camera and you can take pictures'. It was 'OK, let's plug in a button, and attach that to the Pi and let's make that be the button for the camera'. And just a simple intro like that opens up a world of possibilities. Sometimes a lot of these things, such as Jam, just gives you a lot of

Clive Beale (left) and Carrie Anne Philbin were both teachers until recently, while Ben Nuttall comes from the world of software development.

inspiration. Or if you see something in a magazine or online, or on Twitter, and you think somebody's done that with the Pi, I'd really like to do that project in my garden or I'd like to do that myself and twist it and use some of the libraries they've used or, use some of the codebase they've used and take it in their own direction.

Clive: Like maybe it's not a button; maybe it's a sensor for when your parents walk into your bedroom, and it then tweets it as they walk in.

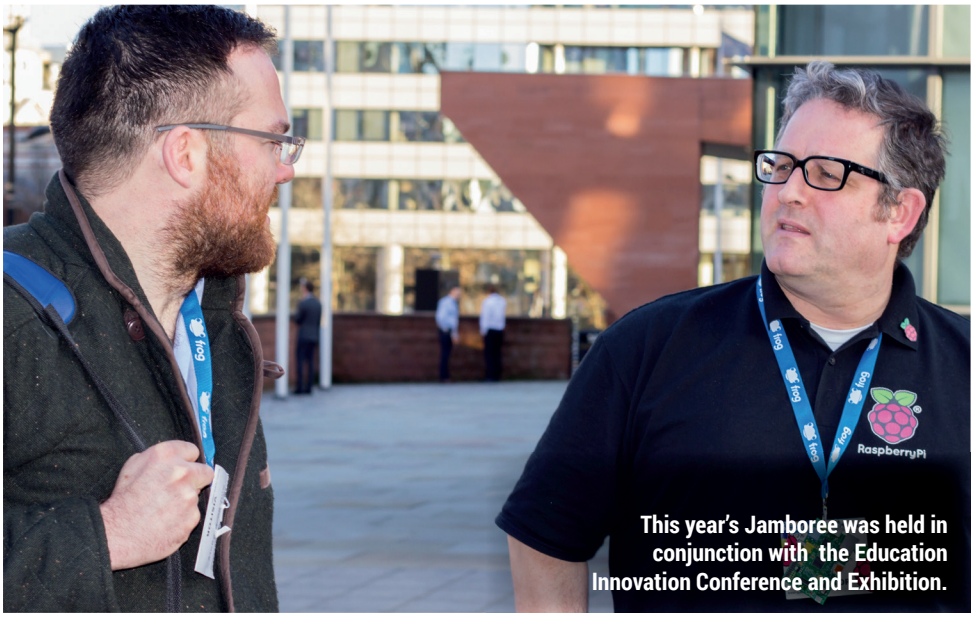
Ben: And everyone's got a different way of thinking. If you're in classroom of 30 kids and you show them how to make a button do this, each of them is thinking 'Oh I can make a such and such'. They'll all have a different idea. And some of them will just go straight home and make one. And some of them would need a lot more guidance.

Are there some kids that just don't get it at all?

Ben: I think there's something for everyone, but they might not find it straight away. If you delivered a term's worth of content for a class, with a good scope of different projects, I'd be surprised if there was one kid that wasn't interested in any of it or didn't find any of it engaging.

Clive: It's almost an antidote for kids not getting it. With teaching music, you'll have people that are level 5 or 6 while others can't read music. Because computing is creative and engaging, we don't all have to become master coders. With Scratch, it's a visual language and you'll find that quite often you'll get what are classed as low ability kids who just rip into that and do fantastic things because it's the first time they've been allowed to get ideas out of their heads and make something with it, whereas before, if they've had problems with writing and numeracy, they haven't been able to do that.

There are case studies with young boys who aren't very good at reading and writing but they start telling stories when you give them an environment where they can actually do these things. So it's not that they don't get it. There's something for everybody.



This year's Jamboree was held in conjunction with the Education Innovation Conference and Exhibition.

Pre-Raspberry Pi, in the dark ages of about five years ago, before the ICT revolution, what would those children be doing? Would they have responded to ICT in school at all?

Carrie Anne: In a classroom, you have a network of computers that are all on lockdown. You've got your network administrator and team of technicians, and they do a wonderful job and I certainly wouldn't slate them – I was a technician once. But we were living in a time where you had to lock down the internet, which I disagree with, I think it should be open. And all the computers, you can't execute any files on them, so you can't actually teach any programming on them. So that was a problem for me.

Is that changing?

Carrie Anne: I think it is

changing, and will change with the new curriculum. For me as a teacher, what was great when the Raspberry Pi came along is that I didn't actually need those computers around the outside of the classroom any more. I didn't need to seek anyone else's permission any more to do what I wanted to do. Here's my box full of Pis, let's just get them out. You can break it. And that's OK, you just flash it and start again.

Clive: In 1997, they put the C back in ICT [Information and Communications Technology] and suddenly it became this thing that you had to teach. The curriculum wasn't really that bad. A lot of people moan about it, but if you actually sat down and read it, it was quite flexible and did let you take control and make programming and coding interesting. But because resources are so important to schools, you just ended up doing the easy things.

But when you said about kids using Scratch and becoming motivated to do other things, that would never have happened before the Raspberry Pi came along.

Clive: Yes, it was more just following what the teacher said, 'And now we're going to write a letter to the cinema using Microsoft Office' or something. That's like giving someone a Ferrari and saying you've just got to drive in this room for half an hour. So you've given them this fantastic tool for exploration and creativity, and then you're telling them exactly what they have to do with it instead of letting them explore. Whereas with the computing thing, especially things like Scratch, just lets them think, 'OK, I can do a movie, I can do a little flip frame animation, or you know what, I can actually make a game'. And then suddenly they're doing stuff that they haven't had the opportunity to do.

Carrie Anne: That old ICT curriculum was about 12 years out of date. It was created and it wasn't updated.

Obviously quite a lot has changed in computing education according to the media over the last couple of years in the UK. If the curriculum has only just changed, what is it that has been driving improvement?

Carrie Anne: I think the teachers. They're the ones in the classroom who have to teach the curriculum. When I became a teacher, I was already working in a school and I kept putting off becoming a teacher because the curriculum bored me.

But then I realised that, when I actually got into the classroom, I was able to put my own spin on it. I think it started with the teachers –the people who are saying: "We want a new curriculum, we want to teach this". I think it started there, and then industry picked up on what was happening and they wanted to get more industry experts involved. And then the government got involved and it snowballed really.

Clive: Scratch was a big word-of-mouth thing. It was about mid-2000 when it came out, and suddenly you just found that any teacher worth their salt was using it for their ICT curriculum because it taught about control.

Carrie Anne: HTML as well. HTML has been on the curriculum for years. We've been teaching HTML in Notepad for years.

What are going to be the big things pushing it forward over the next few years?

Carrie Anne: I think more of the same really. It'll be teachers, it's always the teachers. They're the ones who come to the Jamboree and this kind of thing, and learn from people like Ben and that sort of collaboration. That's where it starts. The teachers see what can be done, and they start doing it, and there'll be more of that. And there are initiatives like code clubs, and the Master Teachers are great.

Clive: Teachers are meeting up more when before they may not have been getting together.

Ben: And as well as there being more content, I think there'll be convergence of a lot of this stuff. So, because the Raspberry Pi doesn't have any official resources right now, some people are going off and writing their own. I think they'll be a convergence of people pulling their ideas together and there'll be a more centralised system for that. And we'll be helping the community out with that.

You haven't mentioned government policy, or anything like that at all. Is that a negative thing or just by-the-by?

Clive: I was at the Westminster forum yesterday and they had a chap from the Department for Education there, and I couldn't resist it, so I got the mic and said: "You haven't really taken it seriously have you?" He turned round and said, we have *really* taken it seriously. This idea that we can just bring in a new program of study and say 'Oh, aren't we wonderful', because [Google Boss] Eric Schmidt has made a speech, and suddenly we've made it all better for you. But you haven't, you just seemed to have done something that a

lot of teachers are now scared of. So there's a lot of work to do, and we're really positive. But no, the government have not, in my personal opinion, given the money or the support or the thought behind this.

If you go to Jersey, there's such a great contrast with what's been announced for England. They're going to spend £6 million on the 100 thousand people who live on the island – that's the size of Cambridge. Jersey has the infrastructure, fibre to the door in every school, linking into businesses, a £2 million training budget for

"Every teacher needs more time off their timetable to develop their skills."

teachers... That's practically what the government pay for the whole of England with 53 million people.

So, has the government taken the scale of the task seriously: no. I think they've completing underestimated what's involved. They thought the teachers would just pick it up and have the time and resources, which we don't have the time for.

Carrie Anne: Maggie Philbin [presenter of television programmes *Tomorrow's World* and *Bang Goes The Theory*] has been leading a UK digital skills taskforce, and which I'm part of the team. We're looking at where the skill shortages are and what digital skills are needed. Because there's going to be a whole group of kids who are now 14 to 16 who are going to leave the education system who haven't had new programs like these. They were on the old program, so we're looking at who missed out and what we can do about that. So hopefully that report will inform government policy.

If there's one thing you could change about government policy, what would it be?

Clive: It really is to do with support for teachers. This idea that teachers – especially primary school teachers where you have to teach a range of subjects – would be able to suddenly go off, and teach themselves from the third-party resources just doesn't work.

Carrie Anne: Time. One of the biggest recommendations that I would say is time. Every teacher needs more time off their timetable to develop their skills, especially in an area like this that they perhaps think they're weak. Because it takes a while to set things up and start your learning. As a teacher, you get a 30-period timetable, you're teaching for about 22 lessons of that with about 7 free periods, but some of those you'll be covering for another lesson and some of those I need to plan my lesson and mark students' work. They need time.

Clive: Science is a good example. If you're a chemistry or biology teacher at secondary level, there's a scheme where you can re-skill to physics and they will give you free periods, a huge bursary, and they'll also take you off timetable one day a week to go off and go to other schools and retrain, and maybe pay for the cover.

This is the government doing this, and if you do this, this and this, you come out as an accredited physics teacher. So they took that seriously, but yet here's a brand-new subject and they're expecting people just to pick up and run with it, the preparation is completely inadequate.

Carrie Anne: But it's not just time to learn something, it's also time to go and meet industry people. Like go to a company and be in there and work and learn from them, and see what the world is like. Because some people, they went through education, went to university and became a teacher, so they've never left this school environment and they've got no idea of what the world of work is like.

Ben: I think there are some people in the current government who seem to think there should just be this package, and this is what you should deliver as your syllabus this year. Everyone is treated the same. Each teacher has their own class, and they're all different, with different ways of engaging their interest. It needs to be tailored, so the teacher needs to take that material, and perhaps look on the Raspberry Pi site in the next year and say, well this one looks quite suitable for my class, or this one might be a good one to do.

Clive: The government also does not understand that this is long term. So if you're going to start teaching five-year-olds about algorithms and a bit of code and Scratch, what happens further down the line? The secondary school teacher will be saying, I can't teach them Scratch anymore, which is what we do in Year 7 at the moment.

It's a long term challenge, and things will continue to change over the next several years until that pipeline becomes full. And they've said "Here's £2 million, do some training for September 2014". What's 2014 got to do with it? This is five, six, seven years down the line.

THE BIG SWITCH

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

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

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

But while 8,000 is an impressive number, it's not as impressive as 15,000. That's how many people the Munich city council has switched from Windows to Linux over the last decade. Migrating workers of Germany's third-largest city was no easy task and there were plenty of hurdles along the way, but by and large the project has been a storming success.

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

Humble beginnings

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

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

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

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

Doing the maths

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

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

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

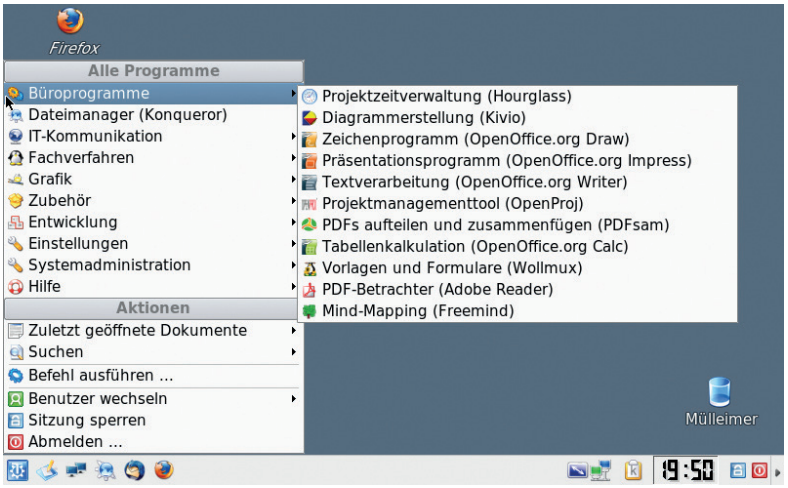
Peter Hofmann is the leader of the LiMux project, and explained its ups-and-downs from his office overlooking the Frauenkirche.

Ballmer marches in

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

What is the LiMux Client?

Put simply, it's a customised version of Kubuntu. We had a chance to explore it in Peter's office, and it's very much what you'd expect from an older Kubuntu release: a Start menu in the bottom-left, various office and productivity applications installed, and a generic theme. There's a bit of LiMux theming in the wallpaper, but otherwise it looks rather plain. A new version of the LiMux Client is due this year; it will be based on Kubuntu 12.04, an LTS (Long-Term Support) release. With this, LiMux users across the city will make the transition to KDE 4, and experience something rather more polished than the KDE 3 desktop they've been used to.



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

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

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

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

And it just got worse for Microsoft's boss. "Our mayor was preparing for a meeting with Steve Ballmer, and because English is not his native language, he asked his interpreter: 'What shall I say if I don't have the right words?' And the interpreter replied: 'Stay calm, think and say: What else can you offer?' Later on during the meeting, our mayor was quickly at the point where he had nothing to say to Ballmer, except for 'What else can you offer?' several times. Years later, he heard that Ballmer was deeply impressed by how hard he was in negotiations!"

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

Alea Jacta Est

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

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

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

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

From Debian to Kubuntu

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

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

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

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

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

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

Who's next?

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

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

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

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

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

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

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

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

Money talks

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

"Yes, it has, depending on the calculation. We did a calculation and we made it publicly available on our information system for the city council. We have the exact same parameters for staying with Windows as with the migration to the Linux platform. Based on those parameters, Linux has saved us €10m."

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

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

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

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

YOUNG REWIRED STATE

Look inside the philanthropic project that's fostering the next generation of coders, with **Mayank Sharma**.

"Open data teaches you that you can make the world a better place – not everything has to be closed up, and it's nice to share things like that, as it can be interpreted in many ways, for many things!" This sage advice doesn't come from a veteran open data advocate, but rather from 14-year-old George Streten. Streten is one of the hundreds of kids, all under 18, who have received neural enhancement at the Festival of Code event organised by Young Rewired State (YRS).

"We are about finding those young programmers and bringing them together at open events around the world introducing them to open data, and each other", explains Emma Mulqueeny, the founder of YRS.

At the events, the kids collaborate with their peers to build projects based on any of the various publicly available real-world open data sets. Mulqueeny is not just a vocal proponent of open data but also played a pivotal role in showcasing its potential to Government officials, which eventually led to the birth of the UK government's open data portal, **data.gov.uk**.

While she was conducting hack days for the government, Mulqueeny noticed that all the coders were older than 25 years of age. So in 2009, "a small group of us decided that we needed to bring the open government data revolution to the next generations", writes Mulqueeny on her blog.

So in August 2009, she along with a bunch of friends, organised a weekend event, christened Young Rewired State, at Google HQ in London with the intention of introducing open data to kids under the age of 18.

To her surprise, just three kids signed up for the free event! It took the organisers three months, and a huge credit card bill for hotels and trains, to find students from all over the UK to fill up the quota of 50 seats.

That experience gave Mulqueeny an unprecedented insight into the ICT education in schools around the UK. Students shared their frustrations of being let down by a curriculum that did not support technical skills or computer science and forced these enterprising students to teach themselves.

Rather than being disheartened, Mulqueeny resolved to give these self-taught kids, who had been programming in isolation, a platform on which they could interact with like-minded peers. By 2010 she had quit her job and founded Rewired State, a for-profit enterprise that organises hack days and has a network of more than 1,000 software developers and designers. YRS became a philanthropic arm of Rewired State, meaning that, unlike Rewired State, YRS is a non-profit social enterprise.

Festive season

YRS has been organising hackathons for kids ever since. From a weekend-long session in 2009 the event is now a week-long affair, and the number of participating kids has been gradually rising.

Young Rewired State has grown from 50 participants in 2009 to over 1,000 today.

While YRS is involved in various activities to engage with coding kids through the year, it brings young coders together from across the UK once a year in a meeting called the Festival Of Code. This stretches over a week, during which kids create all sorts of coding projects. All the apps are built around open data – from something as simple as a website to access data to apps that turn that data into meaningful information.

The event is held in the first full week of August of every calendar year. The event works by gathering kids at local centres all across the UK. At the centres, the kids are encouraged to pair up with their peers, although they are free to work on their own as well.

They can create anything they want, using any types of programs or equipment they've brought along. The only requirement is that the project must include at least one open data set. Mentors are available on-site at each local centre and even online at the YRS IRC channel. The mentors assist the kids with their projects.

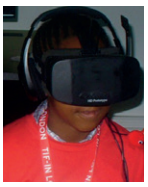
From Monday to Thursday the young programmers assemble at their centres and hack on their projects to create a functional prototype. On Friday, everyone from across all centres travels to and assembles at a central location in the UK to present their work to a panel of judges.

"YRS brings young coders together once a year in an event called the Festival Of Code."

Until 2013, the festival culminated in the Custard Factory in Birmingham. This year that venue has shifted, and the weekend will be held at the University of Plymouth.

Over the weekend, the young coders will present their projects to a panel of judges during the

Tyriah Taylor, 10



I kept asking my mum for a new 3DS and games for my Nintendo DSi and Wii. Eventually, my mum got annoyed and told me that I could make my own games. She started looking for computer clubs and that is when she found Young Rewired State.

I didn't think it was going to be very interesting. However, right from when I got there we had talks and tours of the venues. We had practice sessions for our presentations for the Festival Of Code, which helped build our confidence. Young Rewired State is one of the best experiences I have ever come across and it was great fun!

The first YRS event I went to was at the Rutherford Appleton Laboratory [in Oxford]. I learnt some Python, and we made a game in Game Maker called Food Fetcher. Last year I went to the Microsoft Campus in Reading and I learnt some HTML, CSS and PHP. We made a website called Top Tweets, which is a Twitter search engine, where you put in a word or hashtag and it returns the top 3 most retweeted and favourited tweets containing that word/hashtag.

We don't have any workshops at my school. We have Scratch, but I don't think anyone else knows how to use it, except me. I can't wait for YRS 2014 and I am hoping that some of my friends will be coming along as well.

George Straten, 14

I heard about Young Rewired State from a friend. I hadn't met him in person before; we'd met on Twitter. He invited me to write for his blog, and while chatting to him on one of the numerous Skype calls that we had, he mentioned it. I don't really know what I expected from the event to be fair. If I'm honest, I expected it to be a little dull, but throughout the week, my opinions couldn't have changed more dramatically... I really enjoyed the event!

We created a lost and found website, for the whole of the UK. I created a 30-second video advertisement for it too. At school, we never have any experiences like this. We don't ever have-IT based workshops, which is a huge shame, so it was different from anything that I'd ever experienced.

Young Rewired State's Festival of Code not only is an amazing opportunity, but an opportunity to meet people. I've met a huge community of amazing people, which I've stayed part of through various social media platforms, such as Twitter – without YRS, I would never have met such talented people. Once I'm over 18, the maximum age limit to attend as a participant, I hope to join Young Rewired State, and help other people like me to engage with others and share their talents.

Emma Mulqueeny, Young Rewired State's founder, has in the absence of government policy been preparing kids for the future since 2009.

preliminary heats and the semi-final rounds that will be held on Saturday, and the finale on the Sunday. Their friends and family members are welcome to attend and watch over the proceedings.

Going global

In five years, YRS has grown from a single weekend event with 50 young coders to a week-long event across the UK. Now Mulqueeny and the YRS team are reaching outside the UK in order to foster young programmers in other countries.

The YRS International events are held under the banner of the YRS Everywhere program. The idea with YRS Everywhere is to replicate the scale that has been tested in the UK. The events are currently restricted to a weekend, just like the first YRS event in 2009. It'll then be extended to a whole week and stretched to multiple centres across the new country.

The first YRS Everywhere event was held in 2013 in New York in collaboration with a number of networks such as Mozilla Hive and the Museum of the Moving Image in New York.

Just like the first-ever YRS event in 2009 in the UK, the event in New York also invited 50 kids for a two-day hackfest. During the event they worked with open data local to the US and came up with projects that were of local interest. The kids programmed under the guidance of local developers as well as YRS's worldwide mentor network.

The event in New York was followed by a similar two-day hackfest in Berlin, again with the help of well-known local networks such as SAP and the Open Knowledge Foundation. Just before the close of the year, the team went back to the USA, this time for an event in San Francisco. In 2014, the network is planning a YRS Everywhere event in Asia in Singapore.

The YRS Everywhere events follow the same pattern as the Festival of Code. On the first day the participants form groups and choose the open data set that they will work on. They work on their projects till the afternoon of the next day. In the evenings the hacks are presented to the panel of judges.

You don't have to be an ace coder to participate in a YRS event. In fact, according to the YRS website, the

These kids aren't just learning a new skill; they're sharing, and learning from each other.

Planning a YRS event

Organising an event that runs simultaneously all across the country and involves over a thousand kids is no small feat. But Emma Mulqueeny and her small team manage to pull it with relative ease. So we asked her: What does it take to organise a YRS event?

"Crikey, a lifetime!", exclaimed Mulqueeny. "As with running any hack weekend, the practicalities are about venue, Wi-Fi and power," she explains. Once that's taken care they focus on "who and how to recruit" which applies to both the participating kids and their mentors. Next up is selecting the panel of judges and the prizes. An important aspect of any YRS event is access to local open data. Mulqueeny says it "is critical, as we try to make this about local challenges."

"For the Festival of Code, centrally we deal with all of the logistics and fundraising, but centres act as our outposts," she continues, pointing to her website, which details the requirements for a centre. They also have a team that check the credentials of the registered mentors. "We do not provide training for them, but we connect mentors from previous events with new mentors so that they can ask any questions they have."

Beyond their main event, Mulqueeny says that the YRS Everywhere events are relatively simple to setup. "The organisers apply and we check their credentials: have they run hack weekends before, do they have access to a developer network versed in Open Data to act as mentors, and so on."

If the organisers meet these criteria, Mulqueeny and her team guide them to find the young participants, based on their experience, and support them remotely. If required, the organisers can pay them to put together a team to help run the event.

youngest participant in the 2013 edition of Festival of Code was a mere five years old!

If you have basic programming skills, you are good enough to be a part of YRS. The group accepts coders of all levels; however, it helps if you at least have a very basic understanding of HTML. There are no lessons in programming dispensed at the Festival of Code event. Instead, the event helps young people learn from and teach each other.

Low barrier to entry

According to an FAQ on its website "The only thing we ask is that when you come along you get involved and use your skills to the best of your ability, and we encourage you to be adventurous in your learning."

To encourage more participation YRS has introduced the Google Assemblies program in partnership with Google. The idea behind the program is to introduce kids to YRS and coding.

During a Google Assembly, a YRS participant gives an assembly presentation at their school. They are assisted by YRS mentors to prepare for the presentation using slide decks, videos and YRS and Google swag.

With these assemblies, YRS intends to connect with young students via one of their peers who can share their experience about the event and encourage participation with their enthusiasm. They are hopeful that the association with Google will help demonstrate how important their talents are to a multinational marquee brand that everyone can connect with.

Pitch in!

According to Thom Brooks of YRS, organising the 2014 edition of the Festival of Code will cost them around £250,000. That's quite a sum to raise, and being a non-profit organisation they rely on sponsorship from individuals and institutions.

If you appreciate the work of Young Rewired State and wish to donate, you can email Emma Mulqueeny at **emma@rewiredstate.org** for the official channels of donation. The idea behind all YRS events, including the Festival of Code, isn't to make any profit. All the funds received are used towards organising the festival itself as well as for the fund to support families who can't spare the monetary resources required to cover the associated travel costs.

Besides direct monetary help there are several other ways you can help the initiative. If you have a large room or a conference hall that you can spare for a week, you can sign up with YRS as a Festival of Code Centre. You'll also have to assign an individual who will be the centre lead. When you register as a centre, YRS will work with you to make sure you're all set up well in advance. You'll be listed on their website and participants will be assigned to you based on their geographic proximity.

You can also register your centre as a Hyperlocal Centre. Such a centre allows participants to continue working on their prototypes even after the Festival of Code event has come to an end. The Hyperlocal program is designed to ensure that the prototypes designed during the week complete the journey to a finished product.

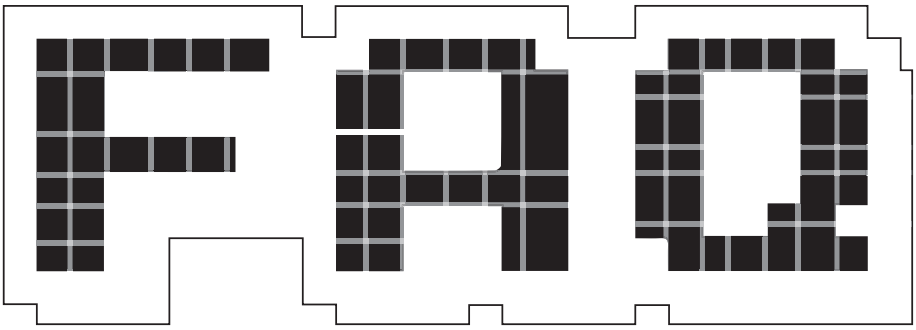
Young Rewired State isn't about coding for the sake of it: the emphasis is on using open data to improve participants local areas.

The Hyperlocal centres will run sessions where the kids can come back to work on their projects under the guidance of their mentors. The YRS team will help prepare the centre to run sessions as per the schedule that suits the centre.

If you have programming skills but are over 18 and can't participate in the events as a coder, you can lend your skills and expertise and volunteer as a mentor. YRS has a thriving mentor community. Traditionally mentors for the YRS events have been drawn from the Rewired State network, but that requirement has been relaxed with their growing popularity.

> **"If you're over 18 and have programming skills you can lend your experience as a mentor."**

If you fancy getting involved, the YRS website has loads of information on what it expects from a mentor. Mentors don't necessarily have to be physically present at a centre and can even guide the coders remotely from anywhere in the world. YRS is growing both within and outside the UK. After establishing a model for scaling the hackfest, the team is replicating it in other parts of the world, encouraging kids to solve their local civic problems with open data and open source code.



DOCKER

The ultimate deployment tool, or just another tech fad?

BEN EVERARD

Q Ok, let's start with an easy one: what is Docker?

A That's simple: it's a tool set for managing deployments of containers.

Q You're stretching the definition of 'simple' a bit. Can you break it down for me a bit more? Let's start by explaining what a container is.

A OK. Remember that Linux is just the kernel and the operating system is this plus all the tools that sit on top of it?

Q Yes, but I thought I got to ask the questions?

A Sorry. A mild digression only. The Linux kernel is the bit that sits on top of the hardware and controls access to the CPU, memory, and all the other stuff that makes up your computer. Normally, you can only use one kernel on a computer at a time.

"You could have a Docker image for OwnCloud and another for WordPress."

However, you can have many copies of everything else. Containers are a way of encapsulating this 'everything else' so that they can share a kernel, and this enables you to run multiple distros on the same hardware.

Q Like having a dual-booting Linux system?

A No. Using containers, you can run multiple distros at the same time, or, as is more common, using the same distro multiple times.

Q Ah, so containers are a form of virtualisation, like Virtualbox or Qemu?

A From a user's perspective they're pretty similar. You have a host OS, and within that host OS, you can boot more versions of Linux. However, at a technical level, they work in very different ways. In virtualisation, you have an application on the host OS that simulates a CPU, then you have another entire OS (including kernel) that runs on this simulated CPU.

In containers, you only ever have one kernel. It's the same for the host operating system and the other operating systems that you run. The containers have their own chunk of the filesystem where they keep all their data, and behave exactly like independent OSes, but it all runs atop the same kernel.

There are advantages and disadvantages to this. Because they run the same kernel, you can't run different operating systems like you can with virtualisation. However, on the other hand, because they don't simulate the CPU, the performance is better.

Q Great! Now I understand it, and we've still got a page to go. Shall we just stick a picture of Linus Torvalds in there and nip off to the pub early?

A Not so fast! That's containers. We need to get onto Docker itself.

Q Oh right, yes. You said before that it's a tool set for managing deployments on containers. I know what containers are, but why would you want to deploy them anywhere?

A The big advantage of containers is that you can encapsulate an entire environment into a single block. This enables developers to pull all the libraries, data, and software into a single container and distribute this. By sending the container, rather than just the software, it means they don't need to worry about dependencies, different configurations, or anything like that.

Q So it's a bit like statically compiling software, but including the whole OS?

A I'd never thought of it like that, but I suppose it is really.

Q Sounds awesome! What's the inevitable downside?

A Nothing major, but the containers will take up more disk space than just the plain software, and they have to be updated separately to keep them current with the latest bugfixes and security patches.

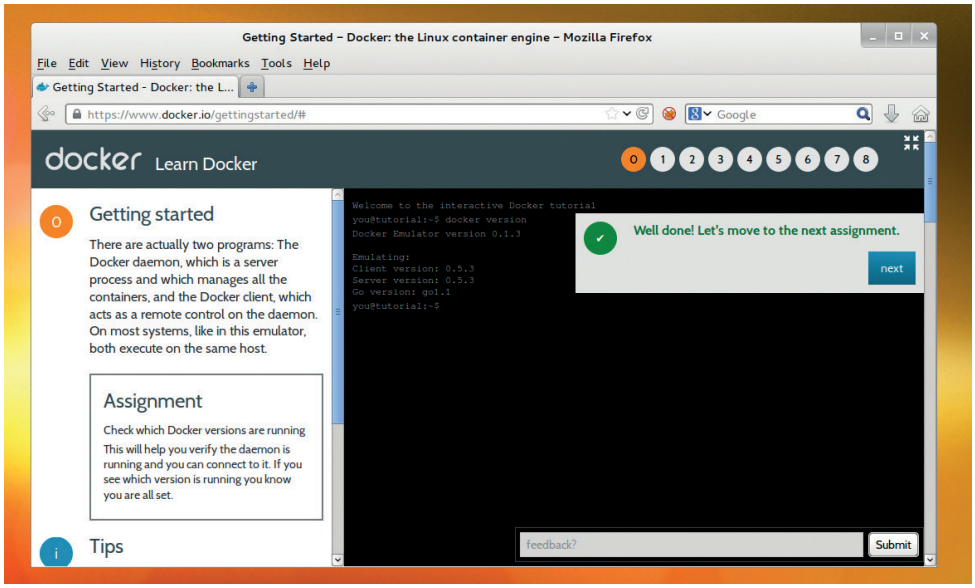
Q So is Docker poised to become a universal replacement for apt-get, Yum, and all the other package managers?

A Not really. No one's suggesting that containers are a sensible way of installing all your regular software. The main target market for Docker is people providing services across a network. For example, you could have a Docker image for OwnCloud, and another for WordPress. They would each have their own environments with everything installed, set up, and ready to run.

By keeping everything contained in this way, it's really easy to customise and deploy. A developer could pull the Docker image to his or her development machine, make any changes they like, then push it to the server. They don't need to worry about the development environment being different to the live environment, because the whole environment is included in the container. It it doesn't matter if it's developed on bleeding-edge Arch Linux, and deployed on ultra-stable Centos, it will always run the same. As well as making it easy to develop, this should remove much of the hassle of setting up test servers, or migrating to a new environment.

The developer can also make any changes they need to the environment without worrying about how these may affect other software running on the server, because that software will be in a separate container.

Q I almost understand your first point now: 'a tool set for managing deployments of containers'. What sort of tools are typically in the set?



You can get a taste for Docker without the hassle of installing anything by trying out the project's interactive tutorial at www.docker.io/gettingstarted.

A The main tool is (unsurprisingly) called **docker**, and it has options for getting and manipulating containers. There's a repository of containers that have been made for common purposes. You can grab these with:

```
docker pull <name>
```

Then, once you've got one installed, you can run commands on it with:

```
docker run <image-name> <command>
```

That's the basic use. There are also a few options to help you manage the containers. It's complex technology, but it's surprisingly easy to use.

Q This all sounds so good, you must have found it really useful when setting up the Linux Voice web services.

A Actually, no. We didn't use Docker. It is, as you say, really good, but it's also really new and still under heavy development. The current version is 0.8, and the people behind it recommend that you don't use it for production systems until version 1.0 at least. It's not unstable, but it's not yet as mature as we like our server software to be. Of course, some people are using it on production machines. We're just a little conservative about such things.

Q Ah... so I should expect it about the same time GNU/Hurd is ready? Perhaps in time for Linux Voice #100 (or Hurd Voice #1).

A Less of your cheek! The first release was in March 2013, so it's just one year old (happy Birthday Docker!). In that time it's come all the way to version 0.9. The earliest plan we heard about was for a release of version 1.0 in October 2013, but by November that year, it had been pushed back to February 2014. At the time of writing, there was still no sign of it, but we don't expect it to be much longer. Of course, just because it's called version 1.0 and the team behind it say that it's production-ready doesn't mean it's ready for everyone. Sysadmins are a conservative species by nature, so we don't expect many people to start using it in important services for a while yet.

Q I'm not a conservative sysadmin, I'm a reckless maverick programmer. How can I get started with Docker?

A You won't find it in many distros' repositories just yet, but there are packages for it for most major Linux distributions at **http://docs.docker.io/en/latest/installation**.

Q You said it ran on Linux containers. I have this friend who runs a commercial OS and won't listen to reason. Can he run it?

A Sort of. You can run Linux on OS X or Windows in VirtualBox, then run Docker in this. There are instructions at the installation website above. Of course, it's far better just to give your friend a talking to about the advantages of open source systems.

SYSADMIN

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

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

When new products are being developed, one of the most important choices to be made is which technologies you're going to build on top of. Obviously, the right tool for the job is the correct answer, but how you decide which tool is the right one is less obvious.

As a sysadmin, my goal is to ensure that our technologies are the most secure, the most stable and the most familiar. All of these characteristics will reduce the chances of me being woken early in the morning, and when I am inevitably woken in the morning, I'll at least have the knowledge and experience to have half a chance of fixing whatever went wrong.

For developers, however, operational requirements often seem restrictive. Enterprise distributions come with technology that's more than three years old and requires the developers to write far more lines of code. This means there'll be more bugs and they may well take much longer to get the product to market.

During the two years I've worked as a system administrator, I've seen this conflict crop up several times, but never seen any obvious solutions that make both parties happy. I agree with the 'DevOps' crowd, who argue that part of the solution is to 'better align incentives', making operations more responsible for delivering software and developers more responsible for maintaining it.

We also need improvements from distributions, providing more reliable means for getting recent, but stable, software packages on to well tested platforms. The Fedora.next wheeze may be going some way to address this, as do PPAs in Ubuntu. Hopefully we'll see this mentality begin to translate to Debian and RHEL in the coming years, too.

Linux containers

Enterprise-grade virtualisation on a real kernel.

While Linux containers have been around for a while, they've recently been gaining more recognition as a lightweight alternative to traditional virtualisation products like KVM or VMWare. With the arrival of LXC, Docker, and the next generation of distributions, we're all likely to see a lot more of them over the coming decade.

As with all virtualisation, the idea of containers is to make it easy to run multiple applications on a single host, all the while ensuring each remains separate. This enables the administrator to carefully manage the resources assigned to each application and to ensure that they can't interfere with each other.

What makes containers different to traditional products is that they don't do any hardware emulation. Instead, the applications in question all run directly on top of the host kernel, just like any other process. Separation between the running containers is achieved through the careful use of a number of Linux kernel features.

Control Groups (cgroups) are the first of these features, and are probably the best known. They provide a means for administrators to group processes, and all their future children, into hierarchical groups. Various subsystems can then be used to strictly manage the processes and the resources they interact with.

Control groups

If you have systemd installed, you can quickly inspect what cgroup your processes are running in with the **ps** command:

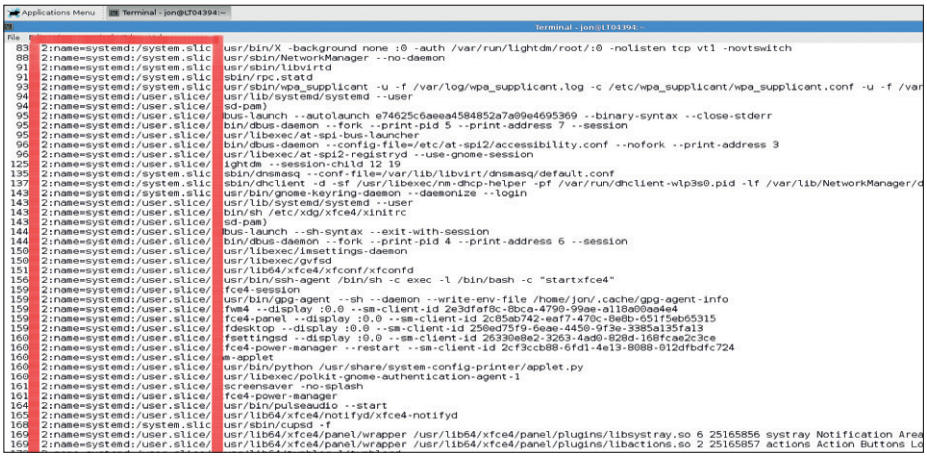
```
ps -aeo pid,cgroup,command
```

Running this, you should see that all processes are running in cgroups that exist in a hierarchy below the systemd cgroup. You could use systemd unit files to manage the resources assigned to a service (indeed, if you're using systemd, this is probably the best way to use cgroups), as described in the last issue, but you can also interact with cgroups directly, too.

There are a collection of tools available in the **libcgroup-tools** package, including **cgcreate**, for example. You can use this tool to create a new cgroup as follows:

```
cgcreate -g memory,cpu:mysql
```

This will create a new cgroup called **mysql** which has been tied to the **memory** and **cpu** subsystems. You can then take advantage



The highlighted area shows the cgroup in which the different processes are running. As you can see, all are either in the systemd defaults of **systemd:/user.slice** and **systemd:/system.slice**.

of a command, such as **cgset**, or interact directly with the virtual filesystem exposed by cgroups, to manipulate the resource limits of this newly created group:

```
cgset -r swappiness=xxx /sys/fs/cgroups/memory/mysql
```

This command will set the swappiness parameter of all processes running in the **mysql** cgroup to **xxx**. To add a process to the cgroup, all you need to do is echo its PID to the **tasks** file in the cgroup's filesystem or use the **cgclassify** command.

Namespace isolation

Namespace isolation is the other key technology that makes containers possible on Linux. Each namespace wraps a particular system resource, and makes processes running inside that namespace believe they have their own instance of that resource. There are six namespaces in Linux:

- **mount** Isolates the filesystems visible to a group of processes, similar to the chroot command.
- **UTS** Isolates host and domain names so that each namespace can have its own.
- **IPC** Isolates System V and POSIX message queue interprocess communication channels.
- **PID** Lets processes in different PID namespaces have the same PID. This is useful in containers, as it lets each container have its own init (PID 1) and allows for easy migration between systems.
- **network** Enables each network namespace to have its own view of the network stack, including network devices, IP addresses, routing tables etc.
- **user** Allows a process to have a different UID and GID inside a namespace to what it has outside.

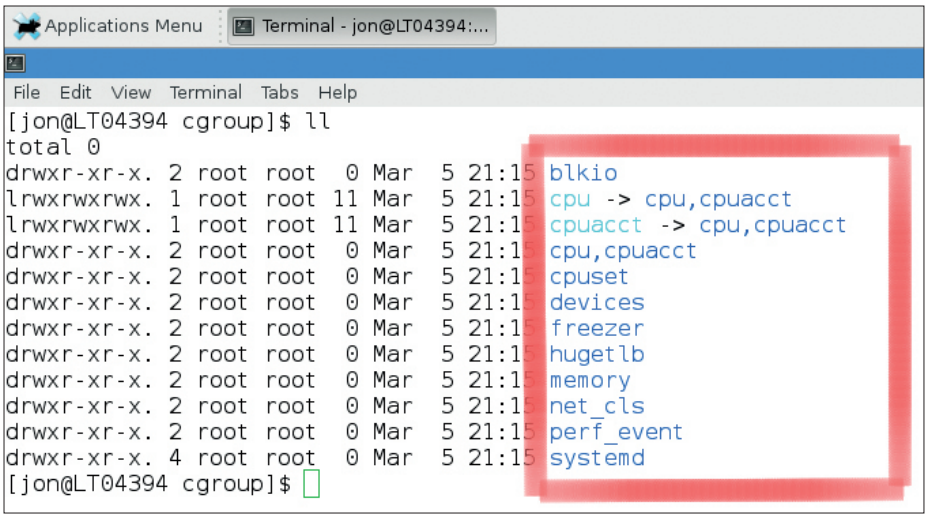
A quick way to experiment with namespaces yourself is to use the **unshare** command. This will run a particular program, removing its connection to a particular namespace of its parent:

```
sudo unshare -u /bin/bash
```

This will create a new bash process that doesn't share its parent UTS namespace. If you now set the hostname to **foo**, you'll then be able to look, in another shell on the same system, and see that the hostname in the root (original) namespace hasn't changed.

Linux containers

Now that you have an idea of what the underlying technologies do, let's take a look at Linux Containers (LXC), a userspace interface that brings them together. To install the LXC userspace tools, you need to install the **lxc** package on Ubuntu and Fedora, but in the case of the latter, you should also install **lxc-templates** and **lxc-extras** for a better experience.



The output of this long listing in the /sys/fs/cgroup directory shows all the different subsystems that are available for managing processes with cgroups on a default Fedora 20 installation.

Once that's done, creating a new container, depending on your requirements, can be simple. In the **/usr/share/lxc/templates** directory, you'll find a collection of scripts that will create some default containers, including Debian, Fedora and Ubuntu system containers, and sshd, BusyBox and Alpine application containers. To put one of these to use, all you need to do is run **lxc-create**:

```
lxc-create -n linux-voice -t /usr/share/lxc/templates/busybox --dir /home/jon/containers/linux-voice
```

- **-n** sets the name of the container.
- **-t** says which template you want to use.
- **--dir** says where you want the rootfs for the new container to be created.

This command creates a directory in **/var/lib/lxc** with the name set by the **-n** flag. The contents of this directory are populated by the script specified with the **-t** flag. If you look at, say, the BusyBox template, you'll see that this script sets up a filesystem hierarchy, copies appropriate binaries and installs important pieces of configuration with **heredoc** statements.

Inside the created directory, you'll also find that a config file has been created. This defines which system resources are to be isolated and controlled by the container. The **man lxc.conf** command goes in to detail on what options can be put in this file, but a few key examples will be helpful:

- **lxc.cgroup.cpu.shares = 1234** Sets the share of CPU that the container has.
- **lxc.utsname = linux-voice** Sets the hostname of the container.
- **lxc.mount.entry = /lib /home/jon/containers/busybox/lib** Specifies directories on the host filesystem that should be mounted in the container.

This configuration file means you can apply the existing templates in quite flexible ways, but if you really want to create a custom container, you're going to have to set to work creating your own template script.

> **"Creating a Linux container is paradoxically easier than creating an application container."**

As the LXC man page says, creating a system container is paradoxically easier than creating an application container. In the latter case, you have to start by figuring out which resources you want to isolate from the rest of the system, and then figure out how to populate the appropriate parts of the file system etc. In the former case, you simply isolate everything – much simpler.

Once you've created your container with **lxc-create** and modified the config file as you see fit, you can start it with the **lxc-start** command, use **lxc-console** to get a console in it, and shut it down with **lxc-shutdown**.

While cgroups and namespaces have reached a degree of maturity in Linux, the user experience still has some room for improvement. If you found the **lxc-** commands tricky to use, you might want to install **libvirt-sandbox**, which will provide a set of scripts and extensions for using LXC through the familiar **libvirt** tools.

CLOUDADMIN

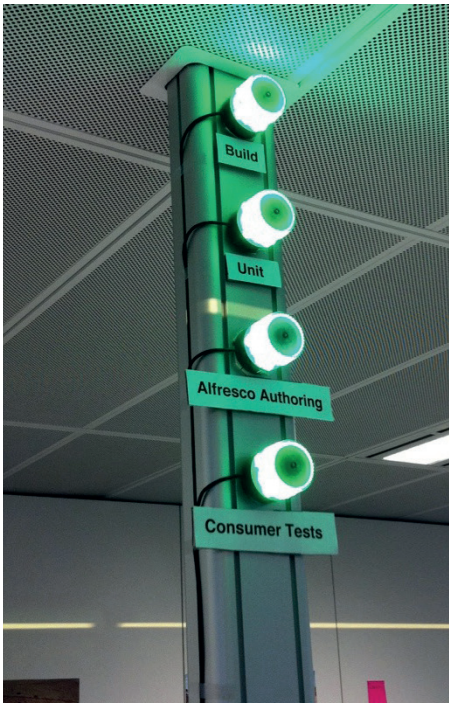
Nick Veitch opens your eyes to the technology behind the cloud server revolution.

Jenkins

Testing code yourself? Why not get a minion to do that for you?

Wouldn't it be great if things worked all the time? I mean, software things, not people things. Gosh, the world would be a better place if I worked hardly at all. That's why software was invented isn't it?

But things (including me) do not work all the time. In fact, in the world of software development, particularly in the new world we now call devops, things are almost perpetually operating in a mode of non-functionality. What is very useful to know is when, almost miraculously, there is a small window of time when software works as it is expected to. In a smart, switched-on world, we can capture this moment so we can work out what went wrong/right. This is the world of continuous integration.



One of the Jenkins plugins can automatically control lights to demonstrate build status (picture by Dushan Hanuska).

The name sounds like some sort of maths purgatory. Continuous integration was born out of the age that gave us agile programming. The idea is pretty good, and all but indispensable in the modern world of group coding on complex software properties. Commits made by coder #1 may be all well and good, and coder #2 is well known for being cautious and testing everything, but if they both land changes, perhaps the two new parts don't interact as they should?

Breaking code is usually ridiculously easy, even with parts that seem to be backwardly compatible – it is often the difference between the programmers' perception of how a particular block of code works, and how it actually works, that leads to ripples of wrongness in the edge cases.

Testing is therefore a good thing, and continuous integration is a blessing.

At your services

In the early days of Agile, one system rose head and shoulders above the others. Its name was Hudson. Hudson was originally a Sun project, but with the acquisition of Sun assets by Oracle (remember them), there were, erm, tensions. The project decided to rename itself 'Jenkins'. Oracle went off in a huff and took its newly trademarked name with it. Hudson still exists, but we'll leave you to decide who came off best from that spat. Anyhow, the open source Jenkins project became the *de facto* required tool for those seeking CI nirvana.

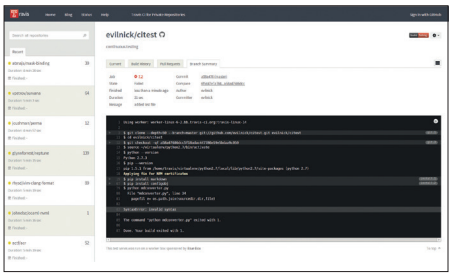
Jenkins itself is a behemothic Java app (aren't they all) which requires a server to run

If you are bored with butlers, you could always replace the Jenkins logo with Chuck Norris.

on. The basic principle is that you submit a job (maybe triggered automatically by time, or by a merge to a public repository) of code. Jenkins then spins up a clean environment for the code to work in, executes your tests and spits out a report. Actually, that is pretty much the least it will do. With an active and productive community, Jenkins has grown a legion of plugins to automate everything from audit trails to uploaders for zubheim (it's a mobile app testing platform). There is a list of plugins so long I couldn't possibly read them all without more gin than is medically advisable.

The only major downside to Jenkins is that it requires a bit of effort to get it working the way you want. Though it was originally imagined as a Java testing tool, Jenkins now covers an impressive range of languages and additional functionality, but almost all of these require adding plugins and some amount of configuration. The results of a

"Though it was originally imagined as a Java testing tool, Jenkins now covers an impressive range of languages and additional functionality."



Log on to the Travis website and you can see the build queue and check out the console output from your jobs. If only to find out what an idiot you are.

well configured, well honed system are very worth it though (you should definitely try the Chuck Norris plugin, which replaces the butler image on your reports with a random image of a culturally significant martial arts practitioner). As well as merely testing the build and functionality of your software, you can automate all sorts of other tasks like publishing documentation or building different versions. If you ever have the need to build hierarchical projects, there is really nowhere else to start.

Did anyone prophesise these people?

Due to the nature of its flexibility and all the gubbins that can be bolted onto it, you may find it takes quite a while to set Jenkins up in a working fashion. For people who don't want to spend half their working day writing new configs for Jenkins, there is Travis.

Travis is quite a bit simpler, but no less capable. Part of this simplicity is due to the fact that you don't have to set up a server to run Travis for you; it is a hosted service. Actually, you can very much download the open source code for Travis from github, but why go to the trouble of building your own service (the docs don't really give you any help on this either) when the hosted service is free for open source projects. Travis integrates really well with github. In fact, that's its primary purpose. And instead of editing acres of config files, you can set up everything that travis can do in a simple file called **travis.yml** and add it to your github repository. Register with the Travis website (sign in with your github credentials, flick a few switches to turn your repositories on and Travis will do the rest. Every time you push updates to the repository, Travis will notice and add your jobs to the end of the queue. Log in to the website and you can check the queue and see the console output of your jobs when they run. But there's no need, because Travis will send you plenty of mails as well, telling you when builds have succeeded or failed, and helpfully giving you plenty of version info.

If you really want to know how easy it is to configure Travis, here are a few examples; firstly, a simple python project:

```
language: python
# versions of python to use
python:
  - "3.3"
  - "2.7"
  - "2.6"
# command to install dependencies, e.g. pip install -r
requirements.txt --use-mirrors
install:
  - "pip install markdown"
  - "pip install configobj"
# command to run tests, e.g. python setup.py test
script:  python mdconverter.py
```

This example declares the language, optionally selects a few different versions (they will execute as separate jobs) and has the **install** command to add any dependencies. The **pip** tool is used in this case to install any additional modules required by the Python file.

The final line is the script to run. In this case, as our project is a simple linear conversion tool, we just execute that, but you could run a separate script to do a number of tests, pass special parameters or anything. For a C project, the Travis file can be as simple as:

```
language: c
```

That's because Travis assumes you are using Automake and will default to running **./configure && make && make test**, so you can put any magic in your Makefile. There is no real dependency management for C projects, but that doesn't stop you from using the **install** section (or **before_install**:, to run first) to fetch such things as you might need. You are also not limited to using GNU **make**. For example, to use **cmake** you would just need to add something like :

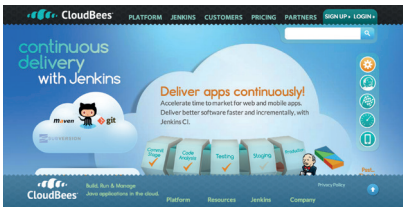
```
before_script:
  - mkdir build
  - cd build
  - cmake ..
script: make
```

Currently the Travis CI environment is an image of 64-bit Ubuntu 12.04, so you can do pretty much whatever you need to with it.

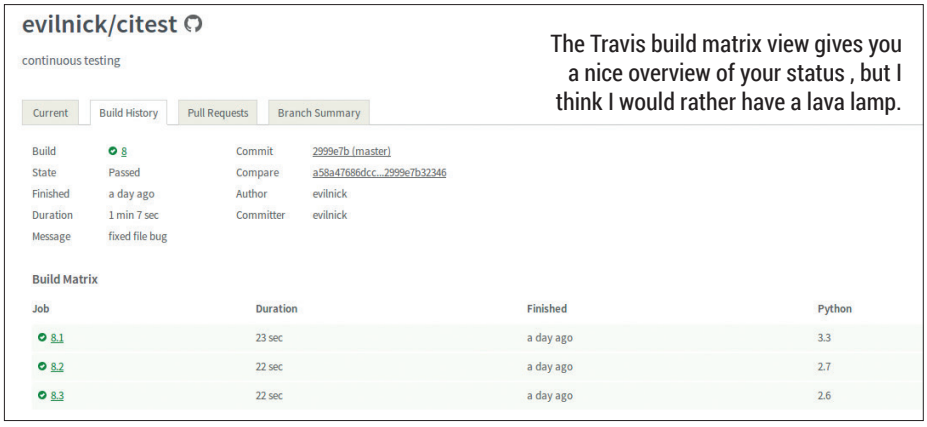
If you have an open source project on github and want a free, no fuss service, Travis could fit the bill. Jenkins is better at all-round automation – if you need to do more than just build a project and run a script, it is infinitely more powerful (with the configuration headaches to match). But whatever you choose, make sure you write good tests! Continuous Integration as a discipline means nothing if your tests don't make sure that things are working. Good tests don't end at "well, it compiled OK", but actually running meaningful tests with the running code itself. Your code (and users) will thank you for it. LV

CloudBees

Cloudbees is a hosted service that offers free Jenkins capabilities, so you can get pretty much everything you ever wanted out of Jenkins, but without having to go through the rigmarole of running your own service. The cloudbees people should know what they are doing too – the CTO is Kohsuke Kawaguchi, the original founder of the Jenkins project.



All the Jenkins service but without the trouble of having one of your own.



The Travis build matrix view gives you a nice overview of your status , but I think I would rather have a lava lamp.

FOSSpicks

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

Mike Saunders has spent a decade mining the internet for open source treasures. Here's the result of his latest haul…

Lightweight blog platform

Drukkar 2.0

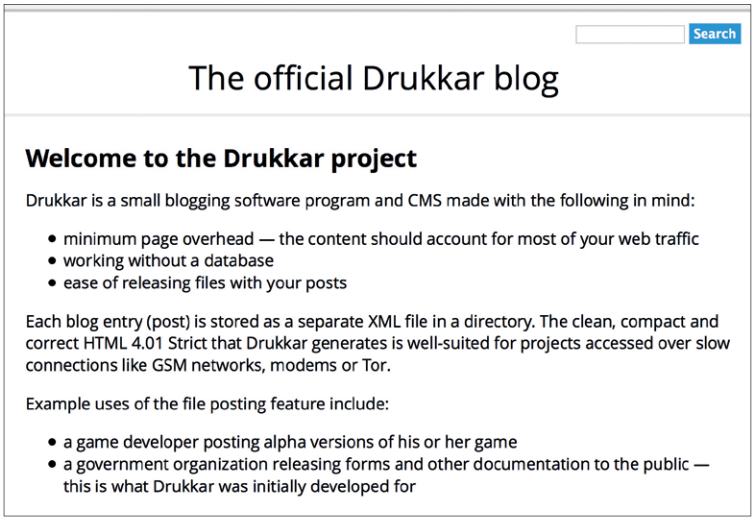
http://drukkar.sourceforge.net

Situation: you want to set up your own blog. Options: a million and one competing blog platforms, each one trying to be more feature-rich, flashy and Web 4.0 than the last. End result: you get tired of trying to work out what's right for you, give up, and go to the pub instead.

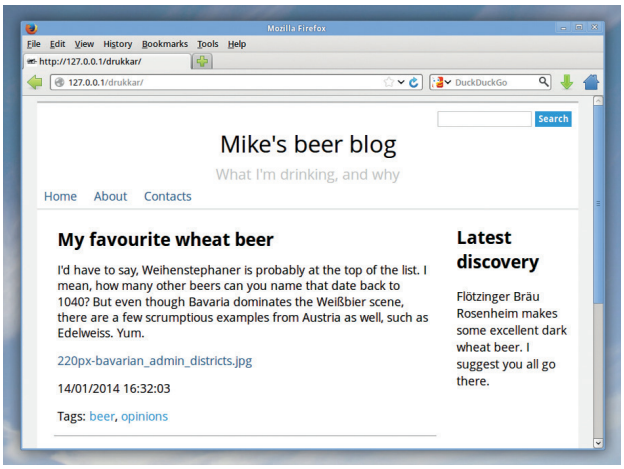
Sounds familiar? Most of us don't need whizz-bang WordPress installations with all the trimmings for our day-to-day musings, and Drukkar does a splendid job with minimal requirements. It doesn't need a database or special Apache modules or anything like that – its only requirement is a web server with PHP. Drukkar stores blog entries as simple XML files, and is designed with minimum bandwidth overheads, so the "content should account for most of your web traffic" as the developer puts it.

Nice and simple

To install Drukkar, extract its **.zip** file directly into a location on your web server (note that it won't create a subdirectory during extraction). Then edit **config.xml** and set the **base_url** and **base_location** settings to match your installation path. Finally, make sure that the entries, files and cache directories are writable by your web server account (eg **www-data** on Debian/Ubuntu systems).



The official webpage of Drukkar is made in, you guessed it, Drukkar – a good old open source "eating your own dogfood" approach.



Drukkar's default theme, "flat", is clean and simple. To change the text on the right, edit **inc/sidebar.php**.

Now access the Drukkar installation in your web browser, and *voilà*: your blog is ready to go. You can see that a sample entry has been created automatically. To add new entries, go to **edit.php** in your Drukkar installation and enter the admin password (by default it's just **password**). You can then add a new blog entry in plain text, HTML or Markdown. It's possible to add tags to entries for organising them later, and add files to them as well (go to files.php to manage them).

See **config.xml** for more options, such as changing the blog's title and creating a new password. It's all charmingly simple, lightning fast and no-nonsense – we've got it running for a tech-shy relative who wants his own type-and-go blog.

Operating system

Android-x86 4.4 RC

www.android-x86.org

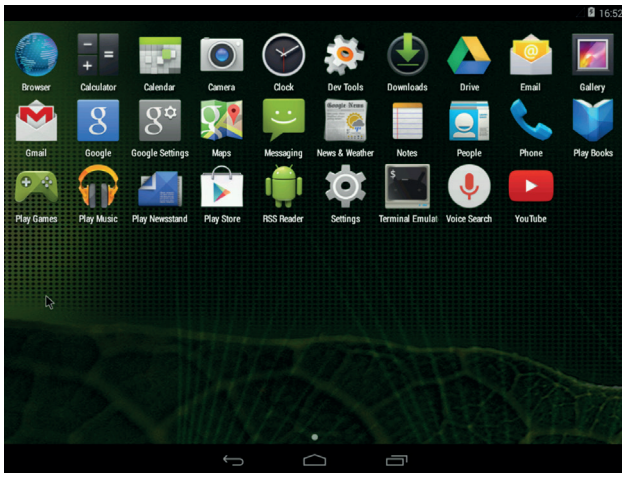
Yes, you can run the world's most popular mobile operating system on your desktop PC. And no, there are no pressing reasons to do so, other than out of curiosity. But that's why we love Linux and Free Software, right? If you're absolutely bursting to do some Android development but can't get your mitts on an Android-powered smartphone or tablet, this project might help you out somewhat. Bearing in mind that Android was developed specifically for handheld devices, don't expect to be amazed with this PC port.

Still, it's surprisingly usable. Android-x86 is available from the project website (and on the Linux Voice DVD) as an ISO image that you can burn to a CD-R, or alternatively, boot in VirtualBox. If you do the latter, make sure that you provide at least 32MB of video RAM to the VM, and choose SoundBlaster 16 for the emulated sound card. Boot from the ISO image and at the initial menu, hit Enter on the first entry to start the OS in live mode, running straight from the disc. You'll be asked a few questions before reaching the home screen. If you're running VirtualBox, you can skip the question about Wi-Fi networks; Android will use the emulated network adaptor. Also, if you can't see the mouse pointer, try disabling mouse integration from the VirtualBox menu.

From here, it's very much standard Android: the black bar at the bottom of the screen is almost always present, and includes three buttons. The first takes you back to the previous screen (useful if you've dived into several levels of settings), while the second takes you back to the home screen from whatever app you're using. The third button brings up a simple task manager for switching between apps.

Our new Google overlords

Click the circle button with six dots inside to view the included software. Android-x86 bundles Google's apps, so you can immediately use Google+, Google Maps and other services without extra fiddling. The Play Store is also provided as well. We're pleased to see that a terminal emulator is included by default: its fonts are ridiculously tiny, so click the three-dot icon in the top-right (Android's stock button for settings), go to Preferences, and finally Font Size. When you've selected something better (eg 16pt), click the back arrow in the bottom bar.

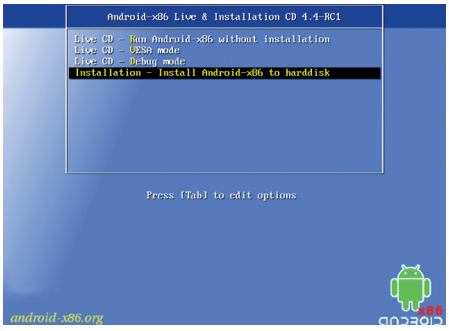


The usual gamut of Android apps is bundled, along with a terminal emulator.

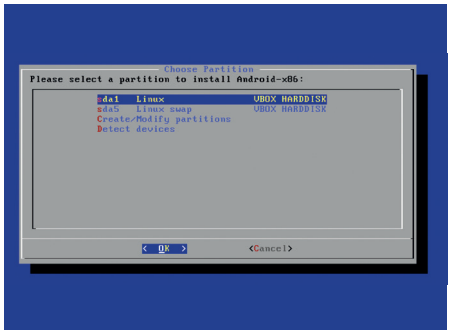
> "In time Android might be a useful desktop OS for users who are already familiar with its interface."

Android as a PC OS is little more than a novelty at this stage, but in time it might be a useful desktop OS for non-technical users who are already familiar with its interface on a smartphone or tablet. If we ever had the time to do our podcast challenges, "use Android-x86 as our sole desktop OS for a week" would be a darn good 'un...

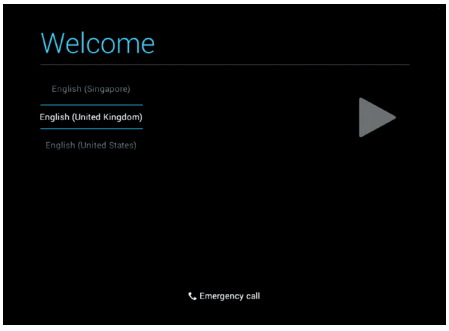
How it works: install Android-x86 to your hard drive



1 Boot
Burn the ISO image to a CD-R – you'll find it on this mont's cover DVD – and boot from it. At the initial menu, select the 4th option: Installation.



2 Partition
Choose a partition on your drive for the Android files. (This will wipe the data!) If you need to partition manually, the third option runs **cfdisk**.



3 Install
Choose ext3 as the file format, choose to install Grub, and the files will be copied over. Reboot and enjoy exploring the OS.

Screenshot creator/editor

HotShots 2.1.0

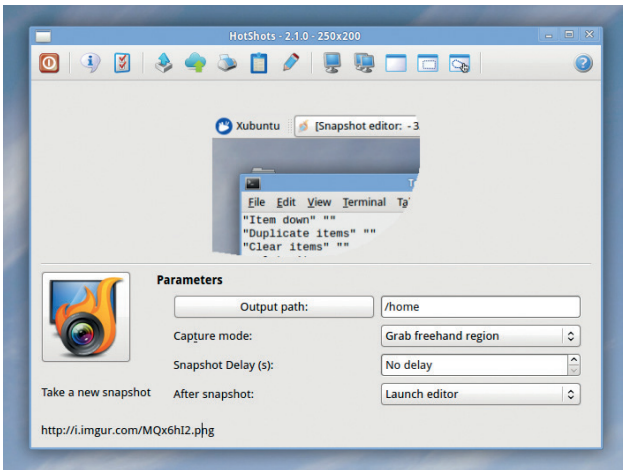
http://sourceforge.net/projects/hotshots

It's something of a cliché to say "Foo is like Bar on steroids", but that certainly applies here. HotShots is a screenshot creation and editing tool with nigh-on every feature you can imagine rolled in, and it does a sterling job. You might think: what's so special about taking a screenshot? Just hit the Print Screen key. Well, that's fine if you just want a rather generic capture of the screen, but if you want to add effects or annotations, it's better to use a specialist tool.

HotShots uses Qt4 for its interface, and the dependencies you'll need on Debian/Ubuntu-based systems are **qt4-qmake**, **libqt4-dev**, **libxtq-dev** and **libxfixes-dev**. Its installation process includes several alternative methods – see **INSTALL.txt** for the full details. Once you have it installed, enter **hotshots** and the main window will pop up. The app also drops an icon in your notification area, so you can easily pop it up when needed.

Many, many extras

In the main window, the most important widget is Capture Mode. Along with grabbing the whole screen or a specific region, this also lets you save hand-drawn regions with smoothed edges – something we haven't seen in other screenshot creation tools. Click the fire button on the left, and after you've chosen the area to grab, the editor will pop up. This is a mightily powerful tool that lets you add snazzy-looking shapes, arrows, numbered tags and text boxes; very useful if you're writing documentation and want to explain parts of an interface.



HotShots lets you draw areas on the screen to take as a screenshot, with the background set as transparent.

You can save your work as a **.hot** file for later editing (eg if you want to reposition elements), or export it in PNG, BMP, GIF and other formats. It's even possible to upload screenshots to an FTP server or image hosting site like Imgur with a single click – great if you need to quickly show someone else on the net your current screen state.

CSS compressor

rCSSmin 1.0.3

http://opensource.perlig.de/rcssmin

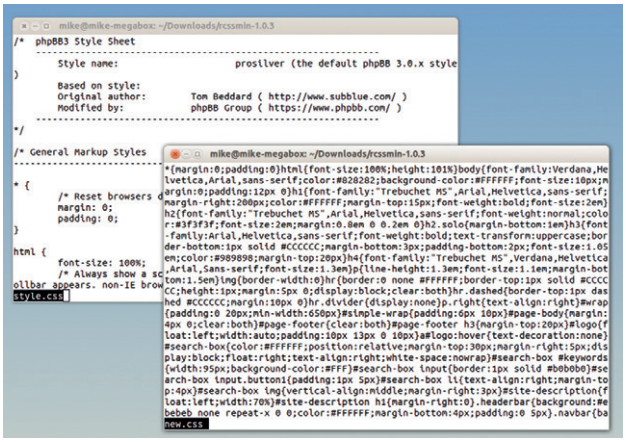
If you're developing a website, it's all too easy to focus on home or office users with fast connections and unlimited bandwidth. But there's an increasing number of people on expensive mobile data tariffs, and there are still some people stuck on dial-up, so it's important to consider all types of users and reduce your site's overall data requirements.

rCSSmin helps in this process by crunching your CSS files to the bare essentials. Chances are that your CSS contains lots of whitespace and comments to make it easy to read and edit, but the web browser doesn't care about that – it just wants the information. rCSSmin doesn't pull any fancy tricks or pretend it knows your CSS better than you do, though. It simply strips out spaces, comments, semi-colons (where possible) and other bits and bobs that aren't required.

Two versions are provided: one is written in Python, takes its input via **stdin**, and spits the results to **stdout**. So if you have a file called **style.css**, compress it like this:

```
cat style.css | ./rcssmin.py > new.css
```

Have a look in **new.css**, and you'll see that it's not as readable as before, but considerably smaller. We tested it with a CSS file from the Linux Voice Forums, and rCSSmin managed to shrink it down from 70k to 40k. This isn't a world-quaking change, but if you're running a big website serving up hundreds of thousands of CSS files every day, it all adds up and saves you bandwidth costs too.



Before and after: our 70k CSS file in the left-hand window, and its 40k crunched version on the right.

> **"It's important to reduce your site's overall data requirements."**

The C version is designed with performance in mind (it's up to 50 times faster than the Python one), so you can even use it at runtime on your sites (ie crunching CSS on the fly as it's served up).

Mountain finder

Gipfel 0.4.0

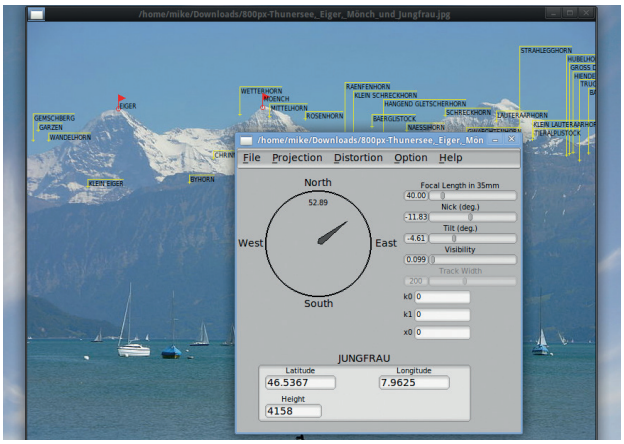
www.ecademix.com/JohannesHofmann/gipfel.html

So you've had a lovely holiday in the Alps, and taken plenty of photos. Back home, you load up the pictures on your Linux box and start browsing them. "Aha", you say, "there's the Zugspitze, Germany's highest mountain. But what's that one behind it?" With some messing around on Google Maps and Wikipedia, you could probably find the name of the mountain in question, but Gipfel makes it easier – and more fun.

Using FLTK for the interface, Gipfel's dependencies are **libtiff4-dev**, **libjpeg62-dev**, **libexiv2-dev**, **libgsl0-dev** and **libfltk1.3-dev**. Once you have the program started, click File > Load Image in the menu, and select a photo from your collection. Choose a picture that includes several distinct peaks, where you can identify two of them, but the rest are unknown to you.

With the picture open, click File > Choose Viewpoint and select the name of a mountain that you know is in the picture. (The database is extensive but some of the names are a bit off – for instance, the Großvenediger in Austria is written as "Gross Venediger".) Now right-click on the picture and choose Find Peak. Enter the name of the mountain again, click OK, and a red symbol will appear. Drag this to the top of the known mountain.

Now repeat this process, right-clicking and choosing Find Peak again, dragging the icon to the other known mountain. If all is well, Gipfel will calculate the location from which you took the photo, and add labels to the tops of other mountains in the picture.



Gipfel can show labels for mountains behind the ones in the picture, so reduce the visibility setting to make things clearer.

> **"Gipfel will add labels to the tops of other mountains in the picture."**

It's possible to manually adjust your latitude and longitude, and also tweak various angles and settings to match your photo more closely. It's a clever little app when it works correctly – it just takes some fiddling to get right.

Text-mode web browser

Lynx 2.8.8

http://lynx.isc.org

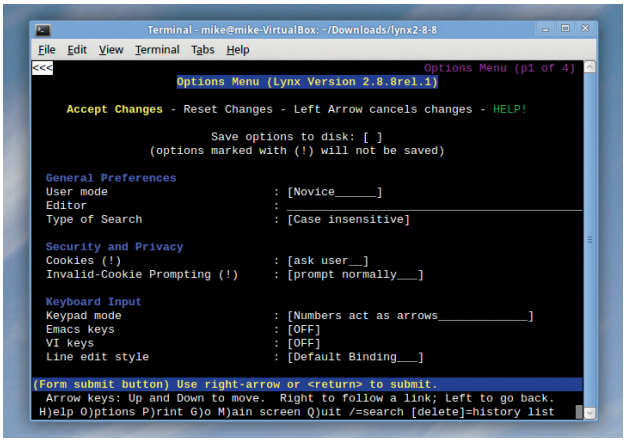
We love programs that run on almost every OS under the sun. Lynx, the venerable text-mode web browser that has been doing the rounds since 1992, has just seen a new release. And this release has been tested on (deep breath): Linux, FreeBSD, NetBSD, OpenBSD, OpenVMS, AIX, HP-UX, Haiku, IRIX, Solaris, Tru64, Mac OS X, and even that weird Windows thing that some people still use.

Now, Lynx is quite limited as a web browser, given that it operates in text mode, but it's good to have around. If something goes wonky with your X server and you need to search the web for help or drivers, you'll need a browser that runs in the plain text terminal. Lynx doesn't attempt to recreate complex layouts, but for text-heavy sites it does a respectable job.

Bare-bones browser

The only major requirement to build Lynx is (n)curses. Once you have it running, hit G to enter a URL. Use the cursor keys to navigate through links, enter to follow a link, the left cursor key to go back, and Ctrl+P/N to scroll two lines backwards/forwards. Hit Shift+G to edit the current URL, backspace to bring up the history, and forward slash to do a search. Lynx has an impressive range of features tucked away - so enter H to bring up the help. When you're finished, Q quits.

New features in version 2.8.8 include: **submit**, **reset** and **pwd** commands; a progress bar that's now compiled in by default and available in the options screen; improvements to SSL support; and better interpretation of HTML code.



Hit O to bring up the options screen, which is full of tweakable bits 'n' bobs.

We're really chuffed to see Lynx being updated to support the latest web technologies – it's good to know that it's always there, even when you're exploring some random, obscure Unix variant that few people have heard of.

PDF and PostScript file annotator

Flpsed 0.7.2

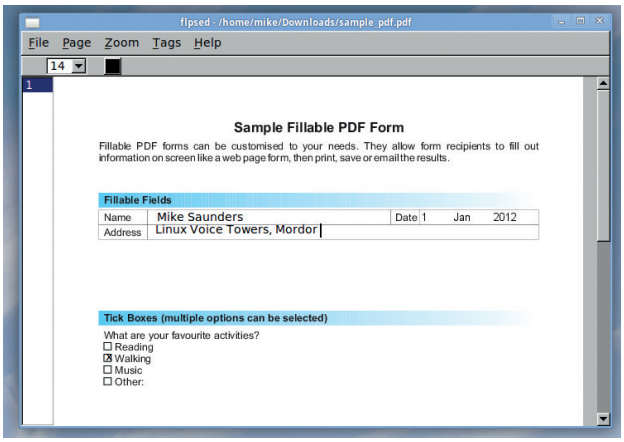
www.flpsed.org/flpsed.html

Despite the best efforts of the Free Software community, people still send Microsoft Word **.doc** files around as if they were some kind of standard. We always recommend that people install LibreOffice and use an open file format, but our words often fall on deaf ears. Usually, the best success we have is asking people to send us documents as PDFs – at least they will render the same across different platforms.

Now, that's all good and well until someone sends you a form to fill in. If your PDF reader doesn't support this facility, you might be tempted to install the dreadfully bloated Adobe Reader, which makes your hard drive grind on start-up and perpetually annoys you with "update me!" pop-ups.

Flpsed is a super-simple lightweight little app that lets you add text to PDF and PostScript documents, without changing the structure of them. You simply add lines of text at whatever position you want, and then re-export the file as a PDF. This means you can fill in forms in seconds without having to mess around in a much more complicated app.

It's built with FLTK, and has a very sparse interface: click File > Open File and choose a document. Then set your font size and colour using the widgets in the toolbar, click on the relevant position in the document, and start typing. When you're done, click File > Export PDF.



Flpsed doesn't alter the structure of PDF documents - it just adds another layer of text on top.

> **"You simply add lines of text, then re-export the file as a PDF."**

This is good enough, but Flpsed has an awesome extra feature: batch processing. It's possible to add tags to PDFs that will later be replaced by custom text strings, so you can, for instance, automatically add text to a bunch of forms without having to go through each one manually.

Security auditing tool

Lynis 1.4.1

http://cisofy.com/lynis

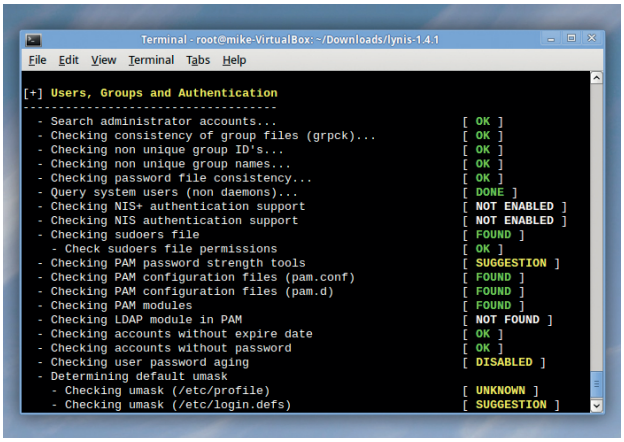
Security is a moving target: you can harden your systems to an almost unbreakable degree, but there's no guarantee that they'll be equally robust tomorrow, when new vulnerabilities are discovered. It's also a vast topic, and if you've been given the job of making a Linux box ultra secure, it's hard to know where to start. You've got packages to update, user accounts to maintain, settings in **/etc** to check... it adds up.

Lynis provides a one-stop-shop for checking your system's security. It's a script that pokes around in various parts of your Linux installation, alerting you to potential problems. Usefully, it's written in plain shell script (so it will run on virtually every Linux distro in existence) and can be executed in place without installation. To use it all you need to enter is:

```
tar xfv lynis-1.4.1.tar.gz
cd lynis-1.4.1/
sudo ./lynis -c
```

(The **-c** flag performs a complete check – just run **sudo ./lynis** on its own to see all available options.) Lynis works step-by-step through different aspects of your setup, pausing along the way to help you see what's going on. It looks at potential problems in user and group settings, looks for vulnerable packages, checks file permissions, tests your networking setup, looks at your compiler configuration, and many other things.

As the scan progresses, Lynis provides feedback in different colours: green means that everything's OK, yellow is used for warnings, and red signifies major problems. Once the scan has completed, you're given a much more detailed set of instructions for fixing issues that have cropped up.



Lynis doing a scan: it has some suggestions for user and group settings, which it will explain in more detail at the end.

Lynis is no silver bullet and won't make your machines impenetrable, but it's well worth keeping around and running periodically to keep one step ahead of crackers.

FOSSPICKS Brain relaxers

Deliberately annoying Tetris variant

NightmareTris 1.0

https://github.com/giacomodrago/nightmaretris

Tetris does funny things to your brain. Play it for long enough, and you'll start to wonder if the game is trying to taunt you, giving you especially awkward pieces at the most difficult moments. In reality, most Tetris variants pick pieces at random, so you just have to hope that your luck is in. NightmareTris is different though: it includes an algorithm that deliberately chooses the worst pieces for your current situation. Need the long thin one to get rid of four lines? You ain't going to get it...

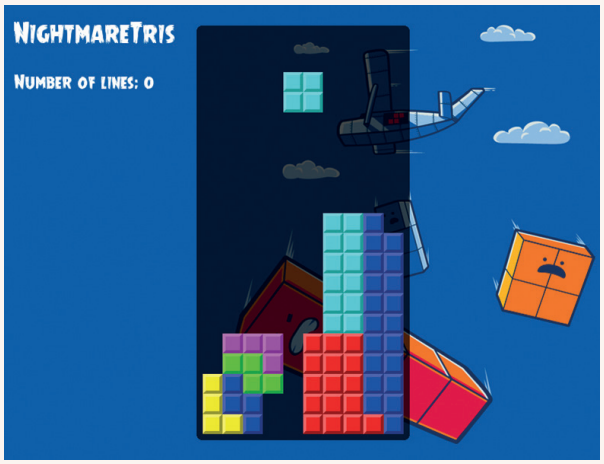
As well as being annoying to play, NightmareTris is a bit fiddly to install. It's written using LÖVE, a 2D game engine; the versions of **love** and **liblove** available in Ubuntu 13.10's repositories don't work properly with the game, so if you're using that distro, get the latest packages from **www.love2d.org**. Then run the game as follows:

```
love NightmareTris.love
```

(Add **--windowed** to the end of the command to disable full-screen mode.) Hit F2, and a standard-looking game of Tetris begins: use the left and right cursor keys to move pieces sideways, up to rotate them, and down to drop the piece to the bottom.

Curse my metal body!

There's no music or fancy effects, but the gameplay is great: it's classic Tetris, made extra difficult.



How many times are we going to get a 2x2 block? Until we lose the game, clearly...

Sometimes the game doesn't seem to be too evil, but you know that it's only planning to make your life harder down the line.

If you enjoy a spot of Tetris but got bored with the standard variants years ago (how could you?!), it's well worth trying this. Don't punch your monitor too hard though...

Souped-up Tron-like game

Zatacka X 0.1.6

https://github.com/simenheg/zatackax

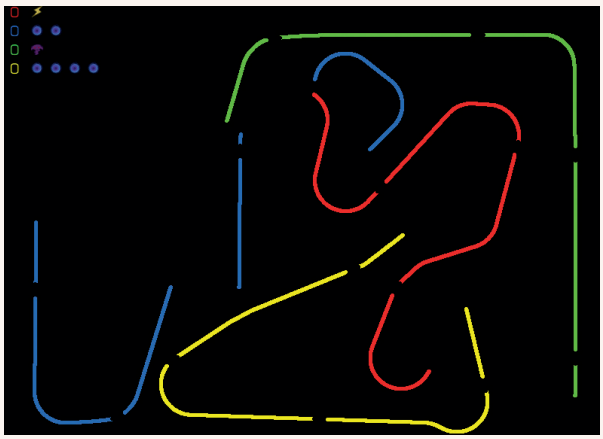
We've seen countless Tron-inspired games come and go over the years, and they're all pretty much the same: you control a moving dot that leaves a path behind it, and CPU-controlled opponents also zip around, leaving trails on the screen. Your goal is to survive for as long as possible without crashing into any of the trails.

Zatacka X takes a slightly different approach; it's based on a game called "Achtung, die Kurve" (watch out for the curve) with some extra goodies sprinkled on top. Getting it compiled is fairly simple as it just depends on common SDL libraries: in Debian/Ubuntu-based systems you can get these via **libsdl1.2-dev**, **libsdl-mixer1.2-dev**, **libsdl-ttf2.0-dev** and **libsdl-image1.2-dev**.

Build the game with **make** and run it in place with **./zatackax**. The game is set up for multiplayer, but if you want to play against the computer, go to Settings > Player Config and set the AI for extra players to On. At the main screen, highlight Start Game and press the right arrow key to add the AI players you just enabled, then hit Enter.

Race against the machine

Before the game begins, there's a choice of power ups: these give you temporary boosts, or afflict your opponents. Hit Enter again to start playing. Use the left and right keys to change the direction of your line, and up to use a power-up. You'll notice your line changes direction in a smooth curve, rather than at right-angles like in other Tron-esque games, and the power ups make it much more entertaining.



Trails occasionally have gaps in them, leaving possible escape routes when the arena starts to get full.

YOUR AD HERE

LV

Email andrew@linuxvoice.com to advertise here

LINUX VOICE

TUTORIALS

Dip your toe into a pool full of Linux knowledge with eight tutorials lovingly crafted to expand your Linux consciousness.

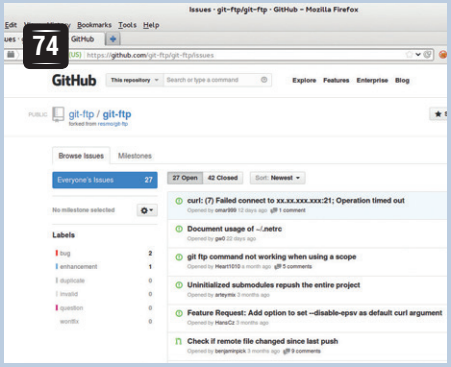
Ben Everard
is now developing a secret receipe for the ultimate porter ale.

You lot were pretty keen to get issue 1. As soon as we put it online for subscribers, our web server was hit pretty hard. Not enough to take it offline, but enough to slow it down. Not that we noticed of course, we were too busy trying to work out why some people couldn't get access, and why some subscribers' email addresses weren't working (it all seems to be OK now). Once the dust had settled, we needed to work out how our server had coped.

We turned to our sysadmin toolkit to find out. Piwik (our analytics platform) kept logs of how long it took pages to load, and Sar kept track of how much the processor power was being used. They're both great open source tools that make the life of a sysadmin so much easier. I've been using Linux for well over a decade, but they're both new to me, and proof that we can always keep learning new things in Linux.

(In case you're wondering, our page load times slowed from about 0.5 seconds to almost 0.8 seconds, and the processor usage barely topped 30%.)

In this issue...



Bug reporting

You don't have to program to help make open source software. **Ben Everard** introduces bug reporting.

Pi arcade

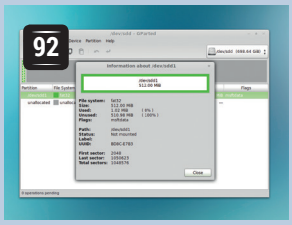
Get gaming on Linux without a trace of Valve's SteamOS in sight – with **Graham Morrison.**

Grace Hopper

Juliet Kemp uncovers programming on the UNIVAC in 1950s America.

KDE konfig

Build the perfect KDE desktop for you – with **Graham Morrison**



UEFI

Conquers the controversial boot technique – also with **Graham Morrison.**

PROGRAMMING

Drawing fractals

96 Picasso, Monet and Michaelangelo all used paint and brushes. The fools! It's far better to create pretty pictures by writing Python code. How, you ask. Well, with nothing more than a few loops and recursions, beautiful canvases can bloom from underneath a turtle…

Key distribution

100 Encrypted communication is a great way to protect your privacy online, but how you you get the keys to everyone in the first place? After all, without keys, encryption is useless. Through the wonders of mathematics and physics, you can foil the feds as they attempt to listen in.

Linux kernel

102 The Linux kernel is the product of the combined effort of thousands of people. If you want to join the illustrious ranks of kernel developers, the first step is learning how it all works. We'll walk you through the basics, and help you get started. Together we can make it even better.

LINUX VOICE TUTORIAL

FILING EFFECTIVE BUG REPORTS

BEN EVERARD

Found a mistake in your favourite software? Share the knowledge, help the developers out and we can all help make software better.

WHY DO THIS?

- Feel the warm glow of helping your fellow Linux users.
- Gain an insight into how Free Software development works.
- Have a say in how your favourite software progresses.

A bug is an incorrect behaviour in a piece of software. This could be anything from the program crashing, to not rendering graphics properly to small things like spelling mistakes in the user interface. They're a fact of life for anyone who uses computers and no software is completely immune to them. Open source software will get a lot better if people help the developers by filing good bug reports, because unless developers know what the problems are, they can't fix them.

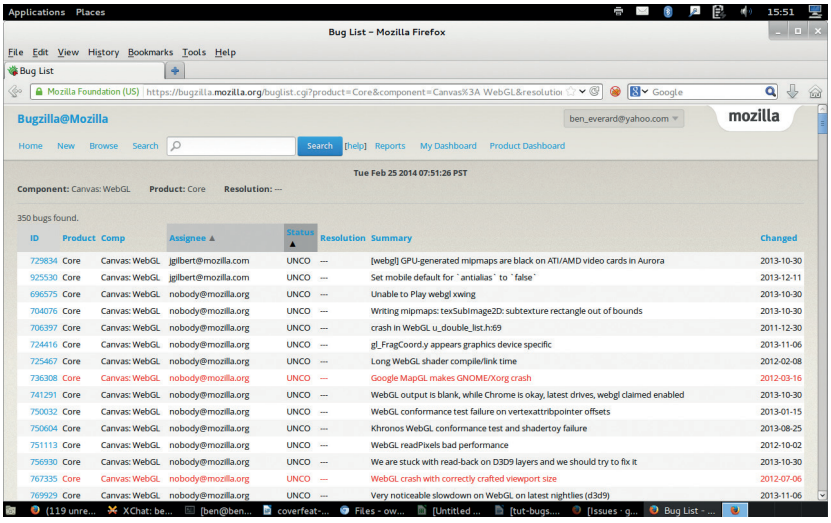
The most important thing with bug reports is to not be afraid of them. Anyone who's written software knows that bugs are a part of life and they won't be mortally offended by the suggestion that their software is somehow imperfect. In fact, they'll probably be grateful for the feedback.

"Public bug reporting is an essential part of the free software development cycle."

There are a few simple things that can make bug reports much more useful, and we'll have a look at these here. The first step, though, is to make sure you have the latest version of the software. You should upgrade through your package manager. If possible, you should check the latest version on the program's web page, and install this if it's more recent.

As far as filing bug reports are concerned, there are two types of software: those with bug trackers and those without. Bug trackers are databases of bug information, typically with a web front-end. If you notice a bug, the first stage is to go to the project's website and find out how to report bugs. Larger projects will usually have a website describing what to do, and any information in that obviously supersedes any general advice we give here. If there isn't a bug tracker, you'll need to email either the developer or a mailing list with information about the problem.

There's no point in flooding bug trackers with duplicate reports, so before you submit anything, check to see if the problem is already on the system. A bug tracker should let you search the current reports, while projects without trackers often have information about known problems in release notes, or elsewhere on their website.



Bugzilla (a bug tracker developed by Mozilla) is one of the most common bug management tools used in open source projects.

Filling a report

Regardless of the bug you've found, there are a few pieces of information that you absolutely must include for the report to be useful at all. This is the version of the software you're using, the operating system you're running, and information about the hardware you're running on. Most of the time, there will be specific fields in the bug tracker that you need to fill in for this. After that, there is usually a text box where you can enter a description of the problem.

The key to a good bug report is reproducibility. If a developer can't reproduce the bug, they can't investigate it and they certainly can't test if a fix works. If you come across a bug, the first step is to make sure you know what caused it. This means shutting down the software, then re-tracing your steps to see if it happens again. If it does, these are the steps you need to enter into the bug report. If it doesn't, you need to look a little bit harder to see what triggered the bug.

Take a look at these two reports:

"Yo, LibreOffice devs. The software breaks when I try to use a picture. Betta fix it quick or I'm movin back to MS Office"

and

"LibreOffice Writer is crashing when inserting a picture into a document. Steps to reproduce:

- Open LibreOffice Writer
- Go to File > New > Text Document
- Go to Insert > Image > From File and select image. Note this isn't happening with all images. I've attached an image that is causing a problem
- At this point, the window becomes unresponsive

This worked fine In LibreOffice 4.1, but has stopped working in LibreOffice 4.2"

(This is only an example, LibreOffice doesn't have a problem with image import.)

The top report is missing loads of key information. What does 'use an image' mean? What piece of the LibreOffice suite are they using? What image are they using? Without knowing this, there's simply no way to investigate the problem.

You might look at the bottom one and think that the steps are a bit simplistic. After all, surely a LibreOffice developer knows how to insert an image without step-by-step instructions? They probably do, but with most software, there's more than one way to accomplish a task, so it helps to go through everything in little steps. Nothing is too basic to be included in a bug report! Also remember that English may not be the developer's first language, so try to keep it as clear as possible.

Most bug trackers also enable you to attach files, and these are a great way of providing the developers with the information they need. In the above example, we attached an image that caused the problem. As a general rule, you should include any files that are involved in reproducing the bug (make sure they don't include any confidential information). Screenshots of the problem happening are often useful as well, though not always possible if the program is crashing.

After the report is filed

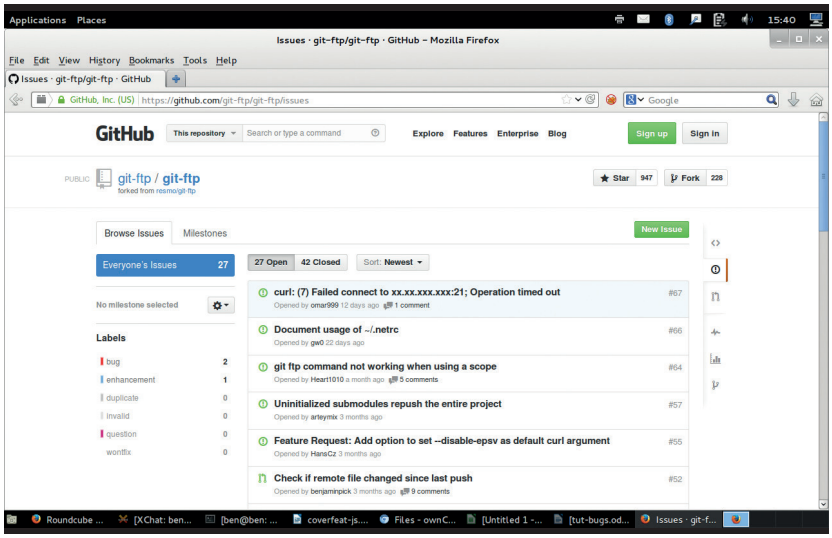
What happens after the bug is filed will depend on the project. On smaller projects, it may go straight to a developer who will look into it. In larger projects, they will often be triaged by a bugfixing team who will try to reproduce the bug before assigning it to the right development team.

It's important for you, as the bug submitter, to keep an eye on the bug report at this stage because they may need more information in order to reproduce the bug. Depending on the problem, they may also suggest a workaround so that you can side-step the bug until it's fixed.

Get more involved

If you want to get more involved in testing open source software, most large open source projects are looking for volunteers to help out. This can include working on bug hunts before big launches or helping triage and investigate reported bugs. It's a great way to contribute to a project, and it doesn't require any programming skill.

LibreOffice is an excellent place to start. The team are incredibly friendly to new testers, and they have a three-day bug hunting session before each point release. The last one (before 4.2) was in December, and you can see details about it on the project website (**https://wiki.documentfoundation.org/BugHunting_Session_4.2.0**). Keep an eye on The Document Foundation's blog (**http://blog.documentfoundation.org**) for details of upcoming events. Alternatively, you could start using beta releases of software that's important to you. These early releases tend to have more bugs in them than final releases, and they need people like you to find all these problems so they can be fixed before the final release. What's more, it gives you (as a user) a chance to make sure that new versions will work properly on your setup with your data.



GitHub (shown here) and most other popular source code management platforms also have bug trackers for the software they host.

If you're unsure about anything in a bug report, most projects have an IRC (Internet Relay Chat) channel, and this is usually the best place to get answers to problems like this, though this does vary from project to project.

Fixing the problem

It's possible that the developer will reject the bug. This could be because the problem is caused by something other than the software itself (such as incorrect configuration), or because they don't think it's a problem (for example, you could be doing something outside of the program's intended use).

Hopefully, though, the bug will be accepted and looked into by the development team. Usually, they'll release a fix and ask you (the bug submitter) to test it to see if it works. This obviously won't go straight into your distro's package manager, so you'll usually need to compile the new source code with this fix in. After this, you should update the bug with information about whether the fix has worked or not.

If all goes to plan, the final step is to mark the bug as resolved in the bug tracker (see the project's documentation for details of how to do this), or letting the developer know that it's worked.

There is one exception to the bug submission process we've talked about here: security issues. Most bug trackers are public, so you shouldn't post any information that could be used to exploit the system, unless the project's documentation explicitly tells you to. If you find a security issue, look at the software's website for guidance, or email the developers directly. It is possible to track security issues with CVEs (Common Vulnerabilities and Exposures) numbers, but this isn't essential.

Filing a bug report doesn't take long, and you should recoup that time by having working software once the bug's fixed. Public bug reporting is an essential part of the free software development cycle. It doesn't matter if you've never touched a line of code in your life – by helping the developers, you can contribute to the free software community and we'll all benefit.

LINUX VOICE TUTORIAL

BUILD A RASPBERRY PI ARCADE MACHINE

GRAHAM MORRISON

Relive the golden majesty of the 80s with a little help from a marvel of the current decade.

WHAT YOU'LL NEED

- Raspberry Pi w/4GB SD-CARD.
- HDMI LCD monitor.
- Games controller or...
- A JAMMA arcade cabinet.
- J-Pac or I-Pac.

DISCLAIMER

One again we're messing with electrical components that could cause you a shock. Make sure you get any modifications you make checked by a qualified electrician. We don't go into any details on how to obtain games, but there are legal sources such as old games releases and newer commercial titles based on the MAME emulator.

The 1980s were memorable for many things; the end of the cold war, a carbonated drink called Quatro, the Korg Polysix synthesiser and the Commodore 64. But to a certain teenager, none of these were as potent, or as perhaps familiarly illicit, as the games arcade. Enveloped by cigarette smoke and a barrage of 8-bit sound effects, they were caverns you visited only on borrowed time: 50 pence and a portion of chips to see you through lunchtime while you honed your skills at Galaga, Rampage, Centipede, Asteroids, Ms Pacman, Phoenix, R-Rype, Donkey Kong, Rolling Thunder, Gauntlet, Street Fighter, Outrun, Defender... The list is endless.

These games, and the arcade machine form factor that held them, are just as compelling today as they were 30 years ago. And unlike the teenage version of yourself, you can now play many of them without needing a pocket full of change, finally giving you an edge over the rich kids and their endless 'Continues'. It's time to build your own Linux-based arcade machine and beat that old high score.

We're going to cover all the steps required to turn a cheap shell of an arcade machine into a Linux-powered multi-platform retro games system. But that doesn't mean you've got to build the whole system at the same scale. You could, for example, forgo the large, heavy and potentially carcinogenic hulk of the cabinet itself and stuff the controlling innards into an old games console or an even smaller case. Or you could just as easily forgo the diminutive Raspberry Pi and replace the brains of your system with a much more capable Linux machine. This might make an ideal platform for SteamOS, for example, and for playing some of its excellent modern arcade games.

Over the next few pages we'll construct a Raspberry Pi-based arcade machine, but you should be able to see plenty of ideas for your own projects, even if they don't look just like ours. And because we're building it on the staggeringly powerful MAME, you'll be able to get it running on almost anything.

Cabinets can be cheap, but they're heavy. Don't lift them on your own. Older ones may need some TLC, such as a re-spray and some repair work.

1 THE CABINET

The cabinet itself is the biggest challenge. We bought an old two-player Bubble Bobble machine from the early 90s from eBay. It cost £220 delivered in the back of an old estate car. The prices for cabinets like these can vary. We've seen many for less than £100. At the other end of the scale, people pay thousands for machines with original decals on the side.

There are two major considerations when it comes to buying a cabinet. The first is the size: These things are big and heavy. They take up a lot of space and it

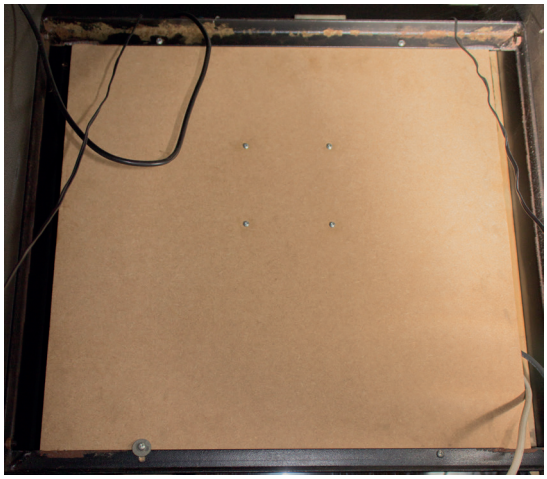
takes at least two people to move them around. If you've got the money, you can buy DIY cabinets or new smaller form-factors, such as cabinets that fit on tables. And cocktail cabinets can be easier to fit, too.

One of the best reasons for buying an original cabinet, apart from getting a much more authentic gaming experience, is being able to use the original controls. Many machines you can buy on eBay will be for two concurrent players, with two joysticks and a variety of buttons for each player, plus the player one and player two controls. For compatibility with the widest number of games, we'd recommend finding a machine with six buttons for each player, which is a common configuration. You might also want to look into a panel with more than two players, or one with space for other input controllers, such as an arcade trackball (for games like Marble Madness), or a spinner (Arkanoid). These can be added without too much difficulty later, as modern USB devices exist.

Controls are the second, and we'd say most important consideration, because it's these that transfer your twitches and tweaks into game movement. What you need to consider for when buying a cabinet is something called JAMMA, an acronym for Japan Amusement Machinery Manufacturers. JAMMA is a standard in arcade machines that defines how the circuit board containing the game chips connects to the game controllers and the coin mechanism. It's an interface conduit for all the cables coming from the buttons and the joysticks, for two players, bringing them into a standard edge connector. The JAMMA part is the size and layout of this connector, as it means the buttons and controls will be connected to the same functions on whichever board you install so that the arcade owner would only have to change the cabinet artwork to bring in new players.

But first, a word of warning: the JAMMA connector also carries the 12V power supply, usually from a power unit installed in most arcade machines. We disconnecting the power supply completely to avoid damaging anything with a wayward short-circuit or dropped screwdriver. We don't use any of the power connectors in any further stage of the tutorial.

Another concession to safety was the disconnection of the old power supply.

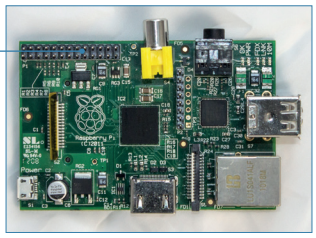


We took the coward's route, and replaced the CRT with a piece of MDF and a VESA mount for a more modern (lighter, less lethal) screen.

LV PRO TIP

For a cheap input interface, buy a PS3 USB controller, dismantle it and rewire the switches to connect to those on your cabinet.

RASPBERRY PI ARCADE SURVIVAL KIT



Raspberry Pi: Any Pi will do, but a Model B with USB and Ethernet is the best option.

4GB SD card: 4GB is the minimum, and you could choose to store games on the network or a USB storage device

Controllers: You don't need an arcade machine. A cheap USB converter and an old generation console controller is almost as good.

Powered hub: A USB 2 powered hub is essential for anything you do with the Raspberry Pi.

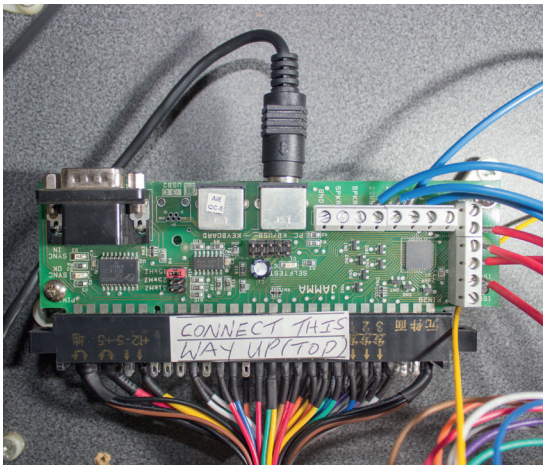
HDMI cable: Arcade monitors can be made to work, but it's a crazy hack. The easy option is to use HDMI or DVI.

2 J-PAC

What's brilliant is that you can buy a device that connects to the JAMMA connector inside your cabinet and a USB port on your computer, transforming all the buttons presses and keyboard movements into (configurable) keyboard commands that you can use from Linux to control any game you wish. This device is called the J-Pac (**www.ultimarc.com/jpac.html** – approximately £54).

Its best feature isn't the connectivity; it's the way it handles and converts the input signals, because it's vastly superior to a standard USB joystick. Every input generates its own interrupt, and there's no limit to the number of simultaneous buttons and directions you can press or hold down. This is vital for games like Street Fighter, because they rely on chords of buttons being pressed simultaneously and quickly, but it's also essential when delivering the killing blow to cheating players who sulk and hold down all their own buttons. Many other controllers, especially those that create keyboard inputs, are restricted by their USB keyboard controllers to six inputs and a variety of Alt, Shift and Ctrl hacks. The J-Pac can also be connected to a tilt sensor and even some coin mechanisms, and it works in Linux without any pre-configuration.

Another option is a similar device called an I-Pac. It does the same thing as the J-Pac, only without the JAMMA connector. That means you can't connect your JAMMA controls, but it does mean you can design your own controller layout and wire each control to the I-Pac yourself. This might be a little ambitious for a first project, but it's a route that many arcade aficionados take, especially when they want to design a panel for four players, or one that incorporates many different kinds of controls. Our approach isn't necessarily one we'd recommend, but we re-wired an old X-Arcade Tankstick control panel that suffered from input contention, replaced the joysticks and buttons with new units and connected it to a new JAMMA harness, which is an excellent way of buying all the cables you need plus the edge connector for a low price (£8).



Our J-Pac in situ. The blue and red wires on the right connect to the extra 1- and 2-player buttons on our cabinet.

LV PRO TIP

If you replace the Pi with a PC, you can configure it from the BIOS to automatically boot when it gets some power.

JAMMA connections

PIN	TOP	BOTTOM
1	GND	GND
2	GND	GND
3	+5V	+5V
4	+5V	+5V
5	-5V	-5V
6	+12V	+12V
7	lock/key	lock/key
8	counter 1	counter 2
9	lockout	lockout
10	speaker +	speaker -
11	not used	not used
12	CRT red	CRT green
13	CRT blue	CRT sync
14	video GND	service
15	test	tilt
16	coin 1	coin 2
17	P1 start	P2 start
18	P1 up	P2 up
19	P1 down	P2 down
20	P1 left	P2 left
21	P1 right	P2 right
22	P1 B1	P2 B1
23	P1 B2	P2 B2
24	P1 B3	P2 B3
25	P1 B4	P2 B4
26	not used	not used
27	GND	GND
28	GND	GND

Get connected

Whether you choose an I-Pac or a J-Pac, all the keys generated by both devices are the default values for MAME. That means you won't have to make any manual input changes when you start to run the emulator. Player 1, for example, creates cursor up, down, left and right as well as left Ctrl, left ALT, Space and left Shift for fire buttons 1–4. But the really useful feature, for us, is the two-button shortcuts. While holding down the player 1 button, you can generate the P key to pause the game by pulling down on the player 1 joystick, adjust the volume by pressing up and enter MAME's own configuration menu by pushing right. These escape codes are cleverly engineered to not get in the way of playing games, as they're only activated when holding down the Player 1 button, and they enable you to do almost anything you need to from within a running game. You can completely reconfigure MAME, for example, using its own menus, and change input assignments and sensitivity while playing the game itself.

Finally, holding down Player 1 and then pressing Player 2 will quit MAME, which is useful if you're using a launch menu or MAME manager, as these manage launching games automatically, and let you get on with playing another game as quickly as possible.

We took a rather cowardly route with the screen, removing the original, bulky and broken CRT that came with the cabinet and replacing it with a low-cost LCD monitor. This approach has many advantages. First, the screen has HDMI, so it will interface with a Raspberry Pi or a modern graphics card without any difficulty. Second, you don't have to configure the low-frequency update modes required to drive an arcade machine's screen, nor do you need the specific graphics hardware that drives it.

Minimise risk of death

And third, this is the safest option because an arcade machine's screen is often unprotected from the rear of a case, leaving very high voltages inches away from your hands. That's not to say you shouldn't use a CRT if that's the experience you're after – it's the most authentic way to get the gaming experience you're after, but we've fined-tuned the CRT emulation enough in software that we're happy with the output, and we're definitely happier not to be using an ageing CRT.

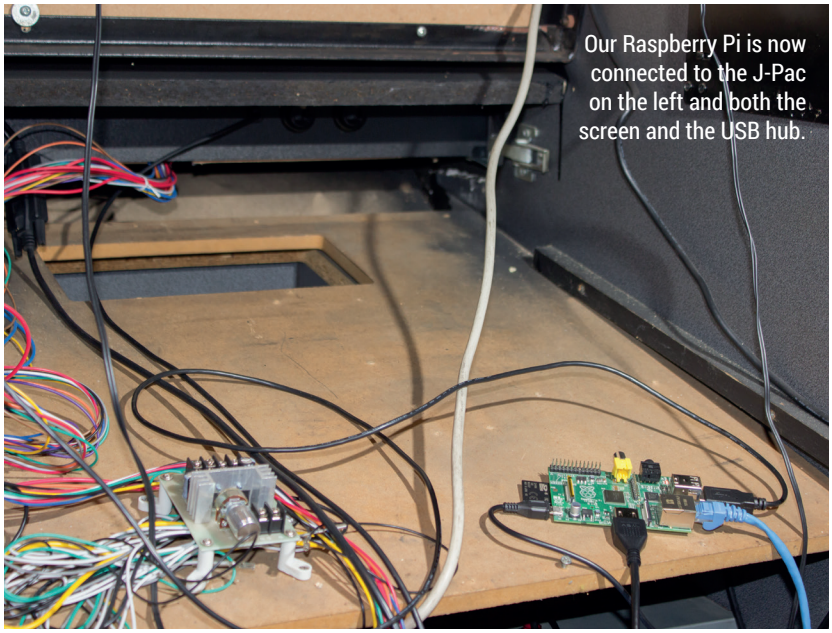
You might also want to look into using an older LCD with a 4:3 aspect ratio, rather than the widescreen modern options, because 4:3 is more practical for playing both vertical and horizontal games. A vertical shooter such as Raiden, for example, will have black bars on either side of the gaming area if you use a widescreen monitor. Those black bars can be used to display the game instructions, or you could rotate the screen 90 degrees so that every pixel is used, but this is impractical unless you're only going to play vertical games or have easy access to a rotating mount.

Mounting a screen is also important. If you've removed a CRT, there's nowhere for an LCD to go. Our solution was to buy some MDF cut to fit the space where the CRT was. This was then screwed into position and we fitted a cheap VESA mounting plate into the centre of the new MDF. VESA mounts can be used by the vast majority of screens, big and small. Finally, because our cabinet was fronted with smoked glass, we had to be sure both the brightness and contrast were set high enough.

3 INSTALLATION

With the large hardware choices now made, and presumably the cabinet close to where you finally want to install it, putting the physical pieces together isn't that difficult. We safely split the power input from the rear of the cabinet and wired a multiple socket into the space at the back. We did this to the cable after it connects to the power switch.

Nearly all arcade cabinets have a power switch on the top-right surface, but there's usually plenty of cable to splice into this at a lower point in the cabinet, and it meant we could use normal power connectors for our equipment. Our cabinet has a fluorescent tube, used to backlight the top marquee on the machine, connected directly to the power, and we were able to keep this connected by attaching a regular plug. When you turn the power on from the cabinet switch, power flows to the components inside the case – your Raspberry Pi and screen will come on, and all will be well with the world.



Our Raspberry Pi is now connected to the J-Pac on the left and both the screen and the USB hub.

The creation takes shape

The J-Pac slides straight into the JAMMA interface, but you may also have to do a little manual wiring. The JAMMA standard only supports up to three buttons for each player (although many unofficially support four), while the J-Pac can handle up to six buttons. To get those extra buttons connected, you need to connect one side of the button's switch to GND fed from the J-Pac with the other side of the switch going into one of the screw-mounted inputs in the side of the J-Pac. These are labelled 1SW4, 1SW5, 1SW6, 2SW4, 2SW5 and 2SW6. The J-Pac also includes passthrough connections for audio, but we've found this to be incredibly noisy. Instead, we wired the speaker in our cabinet to an old SoundBlaster amplifier and connected this to the audio outputs on the Raspberry Pi. You don't want audio to be pristine, but you do want it to be loud enough.

The J-Pac or I-Pac then connects to your PC or Raspberry Pi using a PS2-to-USB cable, which should also be used to connect to a PS2 port on your PC directly. There is an additional option to use an old PS2 connector, if your PC is old enough to have one, but we found in testing that the USB performance is identical. This won't apply to the PS2-less Raspberry Pi, of course, and don't forget that the Pi will also need powering. We always recommend doing so from a compatible powered hub, as a lack of power is the most common source of Raspberry Pi errors.

You'll also need to get networking to your Raspberry Pi, either through the Ethernet port (perhaps using a powerline adaptor hidden in the cabinet), or by using a wireless USB device. Networking is essential because it enables you to reconfigure your PI while it's tucked away within the cabinet, and it also enables you to change settings and perform administration tasks without having to connect a keyboard or mouse.

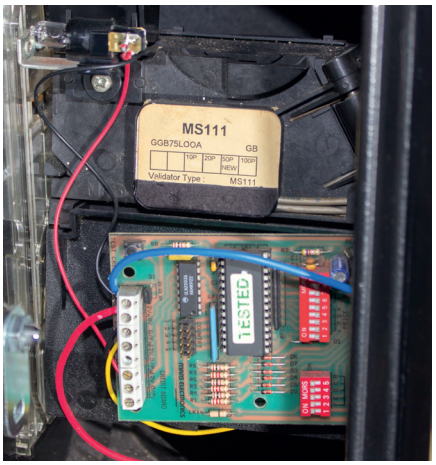
Coin mechanism

In the emulation community, getting your coin mechanism to work with your emulator was often considered a step too close to commercial production. It meant you could potential charge people to use your machine. Not only would this be wrong, but considering the provenance of many of the games you run on your own arcade machine, it could also be illegal. And it's definitely against the spirit of emulation. However, we and many other devotees thinking that a working coin mechanism is another step closer to the realism of an arcade machine, and is worth the effort in recreating the nostalgia of an old arcade. There's nothing like dropping a 10p piece into the coin tray and to hear the sound of the credits being added to the machine.

It's not actually that difficult. It depends on the coin mechanism in your arcade machine and how it sends a signal to say how many credits had been inserted. Most coin mechanisms come in two parts. The large part is the coin acceptor/validator. This is the physical side of the process that detects whether a coin is authentic, and determines its value. It does this with the help of a credit/logic board, usually attached via a ribbon cable and featuring lots of DIP switches. These switches are used to change which coins are accepted and how many credits they generate. It's then usually as simple as finding the output switch, which is triggered with a credit, and connecting this to the coin input on your JAMMA connector, or directly onto the J-Pac. Our coin mechanism is a Mars MS111, common in the UK in the early 90s, and there's plenty of information online about what each of the DIP switches do, as well as how to programme the controller for newer coins. We were also able to wire the 12V connector from the mechanism to a small light for behind the coin entry slot.

We've not been able to try one, but apparently the 25-cent coin mechanism used by nearly all arcade machines in the USA throughout the 80s, and built by HAPP, are even easier to use. These embed a simple microswitch into the coin path, and wiring this to your JAMMA connector will create a credit whenever an accepted coin is inserted.

Whichever system you choose, we've found a working coin mechanism to be the perfect piggy bank as long as you don't raid the coin reservoir too often, or lose the keys.



Our programmable coin mechanism is a Mars MS111. It accepts 10p, 20p, 50p and £1 coins and will send credit pulses at regular intervals to the JAMMA connector.

4 SOFTWARE

MAME is the only viable emulator for a project of this scale, and it now supports many thousands of different games running on countless different platforms, from the first arcade machines through to some more recent ones. It's a project that has also spawned MESS, the multi-emulator super system, which targets platforms such as home computers and consoles from the 80s and 90s.

Configuring MAME could take a six-page article in itself. It's a complex, sprawling, magnificent piece of software that emulates so many CPUs, so many sound devices, chips, controllers with so many options, that like MythTV, you never really stop configuring it.

But there's an easier option, and one that's purpose-built for the Raspberry Pi. It's called PiMAME. This is both a distribution download and a script you can run on top of Raspbian, the Pi's default distribution. Not only does it install MAME on your Raspberry Pi (which is useful because it's not part of any of the default repositories), it also installs a selection of other emulators along with front-ends to manage them. MAME, for example, is a command-line utility with dozens of options. But PiMAME has another clever trick up its sleeve – it installs a simple web server that enables you to install new games through a browser connected to your network. This is a great advantage, because getting games into the correct folders is one of the trials of dealing with MAME, and it also enables you to make best use of whatever storage you've got connected to your Pi. Plus, PiMAME will update itself from the same script you use to install it, so keeping on top of updates couldn't be easier. This could be especially useful at the moment, as at the time of writing the project was on the cusp of a major upgrade in the form of the 0.8 release. We found it slightly unstable in early March, but we're sure everything will be sorted by the time you read this.

LV PRO TIP

MAME is not actually Free Software. It uses a modified version of the BSD licence to restrict commercial redistribution.

Install MAME the easy way

The best way to install PiMAME is to install Raspbian first. You can do this either through NOOBS, using a graphical tool from your desktop, or by using the **dd** command to copy the contents of the Raspbian image directly onto your SD card. As we mentioned in last month's BrewPi tutorial, this process has been documented many times before, so we won't waste the space here. Just install NOOBS if you want the easy option, following the instructions on the Raspberry Pi site. With Raspbian installed and

running, make sure you use the configuration tool to free the space on your SD card, and that the system is up to date (**sudo apt-get update**; **sudo apt-get upgrade**). You then need to make sure you've got the **git** package already installed. Any recent version of Raspbian will have installed git already, but you can check by typing **sudo apt-get install git** just to check.

You then have to type the following command to clone the PiMAME installer from the project's GitHub repository:

```
# git clone https://github.com/ssilverm/pimame_installer
```

After that, you should get the following feedback if the command works:

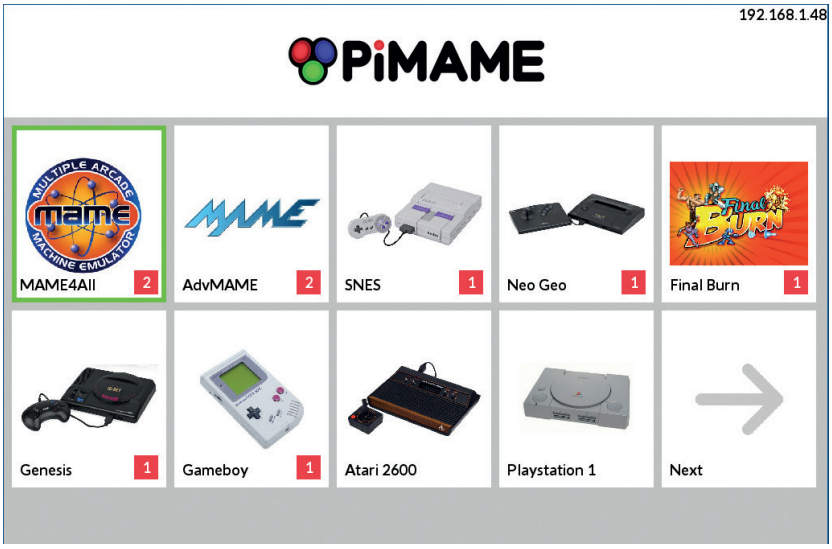
```
Cloning into 'pimame_installer'...
remote: Reusing existing pack: 2306, done.
remote: Total 2306 (delta 0), reused 0 (delta 0)
Receiving objects: 100% (2306/2306), 4.61 MiB | 11 KiB/s, done.
Resolving deltas: 100% (823/823), done.
```

This command will create a new folder called 'pimame_installer', and the next step is to switch into this and run the script it contains:

```
cd pimame_installer/
sudo ./install.sh
```

This command installs and configures a lot of software. The length of time it takes will depend on your internet connection, as a lot of extra packages are downloaded. Our humble Pi with a 15Mb internet connection took around 45 minutes to complete the script, after which you're invited to restart the machine. You can do this safely by typing **sudo shutdown -r now**, as this command will automatically handle any remaining write operations to the SD card.

And that's all there is to the installation. After rebooting your Pi, you will be automatically logged in and the PiMAME launch menu will appear. It's a great-looking interface in version 0.8, with photos of each of the platforms supported, plus small red icons to indicate how many games you've got installed. This should now be navigable through your controller. If you want to make sure the controller is correctly detected, use SSH to connect to your Pi and check for the existence of **/dev/input/by-id/usb-Ultimarc_I-PAC_Ultimarc_I-PAC-event-kbd**.



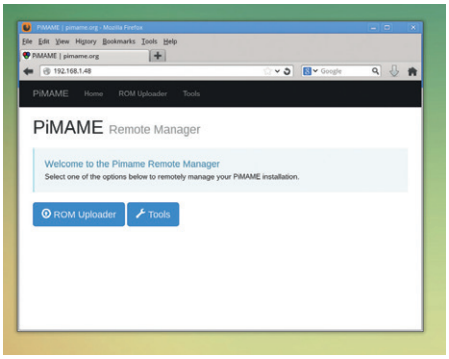
The latest version of PiMAME (0.8) has a great user interface that works well with an arcade machine.

The default keyboard controls will enable you to select what kind of emulator you want to run on your arcade machine. The option we're most interested in is the first, labelled 'AdvMAME', but you might also be surprised to see another MAME on offer, MAME4ALL. MAME4ALL is built specifically for the Raspberry Pi, and takes an old version of the MAME source code so that the performance of the ROMS that it does support is optimal. This makes a lot of sense, because there's no way your Pi is going to be able to play anything too demanding, so there's no reason to belabour the emulator with unneeded compatibility. All that's left to do now is get some games onto your system (see the boxout below), and have fun!

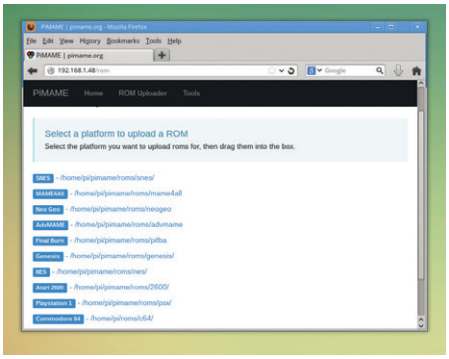
> **"After rebooting, you will be logged in and the PiMAME launcher will appear."**

Graham Morrison wastes too many hours of his life playing Asteroids and weeping for his lost teenage years.

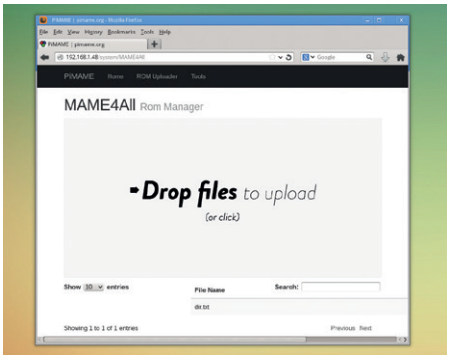
Step-by-step/How To Copying games to your arcade machine



1 Browse to the Pi
Open the IP address of your arcade machine in a web browser. You can find the IP address from the Pi by selecting the Tools menu.



2 Choose the system
After clicking on ROM Uploader, choose the destination emulator for your game. This will then open a new upload page.



3 Upload the file
Dragging the file onto the page didn't work for us, but if you click inside the page a file requester lets you choose the file.

LINUX VOICE
TUTORIAL

GRACE HOPPER AND UNIVAC: BEFORE THERE WAS COBOL

JULIET KEMP

In the days before cheap silicon chips, valves ruled the roost – and it took a special kind of brain to handle these magnificent beasts.

After Babbage and the (never actually built) Analytical Engine in the 19th century, computer development languished for a while. During the first half of the 20th century, various analog computers were developed, but these solved specific problems rather than being programmable. In 1936, Alan Turing developed the idea of the 'Universal Machine', and the outbreak of World War II shortly afterwards was a driver for work on developing these machines, including UNIVAC, famously worked on by Grace Hopper.

Grace Hopper, born in New York in 1906, was an associate professor of mathematics at Vassar when WWII broke out. Volunteering for the US Navy Reserve, she was assigned to the Bureau of Ships Computation Project, where she worked on the Harvard Mark I project (a calculating machine used in the war effort), from 1944–9, co-authoring several papers.

In 1949, she moved to the Eckert-Mauchly Computer Corporation (later acquired by Remington Rand, and later still by Unisys), and joined the UNIVAC team. UNIVAC, which first ran in 1951, was the second commercially available computer in the US, and the first designed for business and admin rather than for scientific use. That meant that it was intended to execute many simple calculations rapidly, rather than performing fewer complex calculations. Punch-card calculating machines already existed, but crucially, UNIVAC was programmable. The first customers included the US Census Bureau and the US Air Force (who had the first on-site installation, in 1952). In 1952, as a promotional stunt, they worked with CBS to have UNIVAC predict the result of the 1952 US presidential election. It correctly (and quickly!) predicted an Eisenhower win, beating out the pollsters who had gone for Stevenson. So let's take a look at what it was and what it was doing.

Grace Hopper, who studied mathematics and physics at Harvard and Vasser universities, at a later UNIVAC in 1960. By Unknown (Smithsonian Institution) (Flickr: Grace Hopper and UNIVAC)

UNIVAC: mercury and diodes

UNIVAC weighed about 13 tons, and needed a whole garage-sized room to itself, with a complicated water cooling system and fans. It had 10 UNISERVO tape drives for input and output, 5,200 electron (vacuum) tubes, 18,000 diodes, and a 1,000 word memory (more on that in a moment); it required about 125kW of power to run. (A modern laptop uses around 0.03kW. It also required a lot of maintenance; replacing diodes, contacts and tubes, not to mention keeping the cooling systems running.

UNIVAC's memory and operational registers were both based on mercury delay lines. The main memory consisted of seven 'long tanks', each containing eighteen ten-word channels. Each channel was a column of mercury with quartz crystals at each end, and held 910 bits (840 bits for the words and 70 for the spaces between each word). The main clock (at 2.25MHz) was in sync with the carrier wave of the column (11.25MHz) and acted as the timer for all UNIVAC operations.

To store data in a channel, the sending crystal (at one end of the channel) was vibrated with the data bits (ones and zeros) of the word. The rate was controlled by the main clock, then the signal was mixed with the carrier wave. The whole signal would move through the column to the receiving crystal, where a bunch of circuitry picked it up, amplified it, analysed it, and sent it back to the sending crystal for another trip through the mercury. So the data was constantly rotating through the mercury, which meant that you could only access a word when it popped out at the receiving crystal end. The average access time for a word was 222 microseconds, so a fair amount of UNIVAC's time involved waiting for word access, with obvious practical programming implications.

You may have noticed that seven lots of 18 channels gives 126 channels of 10 words each; so why only 1,000 words of memory? The remaining 26

channels were used for input and output buffering, for the register, and for the vitally important mercury temperature control. The mercury had to be at an exact operating temperature for the correct transit time and to avoid bit creep; from a cold start, it could be up to half an hour before the tanks were able to hold memory.

Control and computation operations were also run via mercury delay lines, each tank working with a single 12-character word. This made access much quicker, and they also had distribution delay lines to allow multi-bit access to characters. There were four types of register:

- Four Input/Output Synchronizers, used for the 60 word read/write buffers.
- Three Control registers, used for controlling program instruction flow.
- One two-word register (rV) used as a holding area during a two-word move.
- Several of these registers were duplicated, then compared bit-by-bit, to increase accuracy.

Finally, it had an operator's console and an oscilloscope connected. The console had switches that allowed any of the memory locations to be displayed and monitored on the oscilloscope. A typewriter and printer were also connected for output.

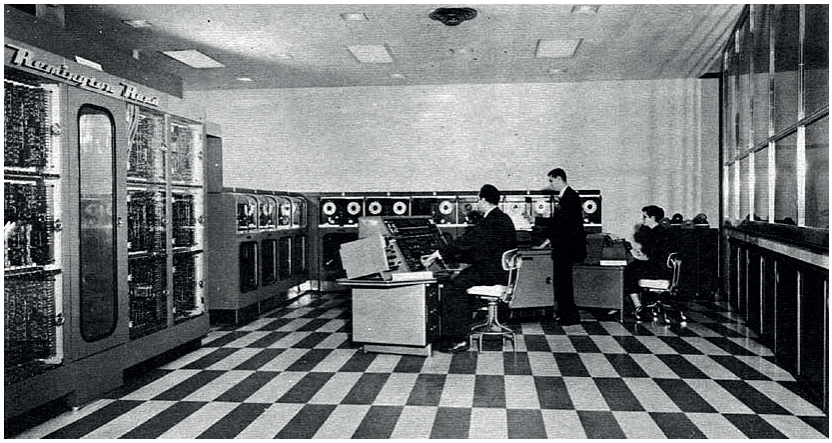
Programming UNIVAC

UNIVAC had quite a big instruction set, which covered transferring the contents of memory into registers, moving the contents of registers around, performing operations, jumping to specific memory addresses, shifting contents of registers a given number of digits, and controlling input/output. The full list (with explanations) is available at **https://wiki.cc.gatech.edu/folklore/index.php/UNIVAC_I_Instruction_Set**. Unfortunately there's no Linux-compatible emulator (see boxout), but here is a small example, with comments:

```
L00 101
   loads contents of memory register 101 into register A.
A00 102
   loads contents of register 102 into register X, adds X to A.
C00 103
   stores contents of register A into register 103, clears A.
P00 103
   print contents of register 103 on the console printer.
```

If you read last month's article on Ada Lovelace and the Analytical Engine, this may look familiar. The instructions (L, A, P) are made up to 3 digits with zero padding. So if register 101 contains the value 2, and register 102 contains the value 3, this will add 2 to 3, store 5 in register 103, and output 5 to the console.

To run this program, it would have been typed onto a program tape as a series of numeric words ('translated' from the programmer's mnemonics given here). The tape (and any needed data tapes) would be latched onto the UNISERVOs, and the operator would manually set various options and begin the booting process from the console. The first 60 instructions would be read into the input buffer, then transferred into memory. The operator would then set the machine back into 'normal' mode, hit the Start Bar, and UNIVAC would begin executing the instructions from memory, beginning with the first block. So the programmer would have to make sure that from then on in, everything that the program needed to do (including reading in more instructions or data from tape) was referenced in the program itself. The operator's role would be limited (at least in theory!) to replacing tape reels as indicated by console messages, and rescuing any minor problems such as a dropped tape loop. Breakpoints could be set in the code (instruction **,**) to aid recovery from problems.



UNIVAC I at the Franklin Imstitute, Philadelphia.

Here's a longer example from the 1954 UNIVAC operating manual. The far-left number is the memory register that contains the instruction. Instructions were saved in memory in pairs, as shown, with the left-hand six-digit instruction run first, then the right-hand six-digit instruction. In this example, registers 100–999 contain a set of numbers, and the code adds them all together.

```
000  C00 099  C00 099
001  B00 099  A00 100
002  C00 099  B00 001
003  L00 007  Q00 006
004  A00 008  C00 001
005  000 000  U00 001
006  900 000  U00 001
007  B00 099  A00 999
008  000 000  000 001
```

Line by line, here's how that code works:

000 C 099 stores register A into memory and zeroes it; so repeating this twice zeros register 099.

001 B 099 loads register 099 into register A; A 100 loads register 100 into register Z, then adds it to register A.

002 C 099 stores register A (now containing A+Z) into register 099, and zeroes register A. B 001 loads the contents of register 001 into register A. The contents of register 001 are the program instructions in step 001; so we are preparing to alter the instructions themselves.

003 L 007 loads the contents of register 007 (see step 007 below) into both register L and register X. Q 006

Grace Hopper remains a source of quotable quotes, our favourite being: "It's easier to ask forgiveness than it it to get permission."

checks whether register L is equivalent to register A; if so, it jumps to register 006 (that is, step 006).
004 A 008 loads the contents of register 008 into register X, and adds it to register A. As register A currently holds the instruction from register 001, and register 008 holds (effectively) a single 1, this alters the instruction from register 001 to read B00 099 A00 101 instead of B00 099 A00 101. In other words, next time we run step 001 we'll add the contents of the next register in the list to our running total. C 001 dumps the contents of register A back into register 001, so we've edited the program on the fly.
005 The LHI is blank; the RHI (U 001) is an unconditional jump back to register 001, ready to add the next number in the list.
006: This simply stops the computer. (Remember from 003: we jump here if the program is finished.)
007 B 099 A 999. This instruction is never actually run. It is used in 003 to check against register A. If register A looks like this at step 003, then we have added the final number (in register 999) and our program is done. 003 will then jump to 006 and the program ends.
008 End of program.

Fundamentally, this is a for loop that sums each element in an array. But UNIVAC programmers had to physically rewrite the instruction inside the loop each time.

One apparently excellent emulator for UNIVAC does exist. It's by Peter Zilahy Ingerman and is described at **www.ingerman.org/niche.htm#UNIVAC**. Unfortunately it's written in Visual Basic 6 and only runs on Windows. The download link on that page doesn't work, but it can be obtained by contacting the author on the given email address. The code above should run on it, but as it's Windows we haven't been able to test it.

Grace Hopper created the first operational compiler, in 1952, while working on the UNIVAC project. Initially, no one believed her. "I had a running compiler and nobody would touch it," she said later. "They told me computers could only do arithmetic." In fact, the A-0 system was more like what we would today call a loader or a linker than a modern 'compiler'. For A-0, Hopper transferred all her subroutines to tape, each identified with a call number, so that UNIVAC could find it. She then wrote down the call numbers, and any arguments, and this was converted into machine code to be run directly. Effectively, A-0 allowed the programmer to reuse code and to write in a more human-readable way, and get the machine to do more of the work.

Programming with A-0

The next versions were A-1 and A-2, with A-2 the first compiler to be in more general use. I found a short paper from a 1954 MIT course, which Hopper also tutored. In it she describes A-2 as handling two types of subroutine: static ones (stored in memory or on tape, either from a general library or specific to the problem) and dynamic ones. Dynamic subroutines could be generated from a 'skeleton' stored subroutine and some specific parameters, or could be generated to handle data. A-2 had four phases of operation:

1. Expands/translates the provided code, adding in data such as call-numbers and operation numbers. (In later compilers this translated from 'code' into 'machine code'.)
2. Divides the result from phase 1 into segments which can be processed in a single storage load, and creates references to each subroutine.
3. Creates the jump instructions needed to complete the necessary jumps (for example between storage loads and subroutines) required by the result of phase. After this phase, you have a complete description of the program, but not a complete program that can be run sequentially.
4. Main compilation. All subroutines are read in and transformed as necessary from 'general' to 'specific' (so any parameters are included), all jumps, reads, and writes are included, and a complete program is generated which can now be run as-is.

(If you check out the PDF course notes in the resources, there are some very cute line drawings illustrating the four phases.)

FLOW-MATIC & English-language programming

A couple of iterations later, Hopper and her team produced FLOW-MATIC, which was the first English-language-like data processing language. (Meanwhile, FORTRAN was completed at IBM in 1957, and is generally agreed to be the first complete compiler.)

Here's a quick sample of FLOW-MATIC code, taken from the FLOW-MATIC product brochure).

```
(0) INPUT INVENTORY FILE-A PRICE FILE-B ; OUTPUT
PRICED-INV FILE-C UNPRICED-INV
    FILE-D ; HSP D .
```

```
(1)  COMPARE PRODUCT-NO (A) WITH PRODUCT-NO (B) ; IF
GREATER GO TO OPERATION 10 ;
   IF EQUAL GO TO OPERATION 5 ; OTHERWISE GO TO
OPERATION 2 .
(2)  TRANSFER A TO D .
(3)  WRITE-ITEM D .
(4)  JUMP TO OPERATION 8 .
(5)  TRANSFER A TO C .
(6)  MOVE UNIT-PRICE (B) TO UNIT-PRICE (C) .
(7)  WRITE-ITEM C .
(8)  READ-ITEM A ; IF END OF DATA GO TO OPERATION 14 .
(9)  JUMP TO OPERATION 1 .
(10)  READ-ITEM B ; IF END OF DATA GO TO OPERATION 12 .
(11)  JUMP TO OPERATION 1 .
(12)  SET OPERATION 9 TO GO TO OPERATION 2 .
(13)  JUMP TO OPERATION 2 .
(14)  TEST PRODUCT-NO (B) AGAINST ZZZZZZZZZZZZ ; IF EQUAL
GO TO OPERATION 16 ;
   OTHERWISE GO TO OPERATION 15 .
(15)  REWIND B .
(16)  CLOSE-OUT FILES C ; D .
(17)  STOP . (END)
```

The **PRODUCT-NO** and **UNIT-PRICE** fields would have been defined separately, in the **DIRECTORY** section of the program. This is just the executable part. Let's step through it:

(0) Load in two input files (A is inventory, B is price), and set two output files (C is priced inventory, D is unpriced).

(1) The key part: this compares the current product number from file A with that from file B:

If they match, then product 1 has a matching price, and we go to section (5)–(9).

If A is greater, we go to section (10)–(13).

If B is greater, we go to section (2)–(4).

(2)–(4) this implies that we have an unpriced product (product 1, for example, exists on list A but on list B the lowest number is product 2). We write it out on the unpriced file D. We then jump to (8), read in the next item A and return to (1).

(5)–(9) Items A and B match; the product has a price. We write it, together with its price, on file C. Then we read in the next item A and go back to (1).

(10)–(13) Item A is greater than item B. Read in the next item B, if there is an item B, and go back to (1). Note that the result of (5)-(9) (a matching pair) will be to read in the next A product but not the next B product, so this balances that out. If we have run out of B data, we rewrite (9) so that all the rest of the products go directly to the unpriced output file.

(14)-(16): close the output files and/or rewind input file B; stop the program.

So this would generate a list of priced items with their prices, and a list of unpriced items. As you can see, FLOW-MATIC was squarely aimed at the business market.

When is a bug not a bug?

Famously, Grace Hopper popularised the term "debugging" about computer programs, after an error while working on the Mark II in 1947 was tracked down to an actual bug (a moth) stuck in a relay. The term "bug" had been used before in engineering, but Hopper brought it into popularity.

A UNIVAC at US Steel in Indiana, on the other hand, had a bug that was in fact a fish; its cooling system, which used water from Lake Michigan, got its intake blocked by a fish and thereby overheated.

COBOL and later

After FLOW-MATIC came COBOL, which Hopper and her team designed from 1959 onwards. COBOL is still in use today, with the 2002 update including OO features, and the compiler GNU Cobol (formerly OpenCOBOL) is available for Linux, with plenty of online resources available. COBOL was intended to be comprehensible by non-programmers, hence its use of English-like syntax and structure. Modern COBOL is still recognisably the same language, and indeed recognisably inherits from FLOW-MATIC. (The first COBOL compiler was itself written in FLOW-MATIC, and was the first compiler to be written in a high-level language.)

"Grace Hopper created the first operational compiler while working on the UNIVAC project."

Grace Hopper moved back into the Navy in the late 1960s. She was on active duty for several years beyond mandatory retirement with special approval of Congress, eventually retiring in 1986, at the age of 79, as a Rear Admiral. She continued to lecture widely on early computing and other aspects of user-friendly computing until she died in 1992.

Juliet Kemp is a scary polymath, and is the author of O'Reilly's *Linux System Administration Recipes*.

More resources

My great thanks to Allan Reiter, whose page at **http://univac1.0catch.com** is invaluable for technical details of UNIVAC operation. Check it out for much more detailed info and plenty of photos and diagrams.

There are some other wonderful UNIVAC resources available online:

- The 1951 'Introduction to UNIVAC' leaflet.
- Remington Rand UNIVAC advertising film from 1950-2.
- Notes from the 1954 MIT special program on Digital Computers (see A-0 section above).
- Bitsavers have a whole bunch of documents from the early days of UNIVAC. These include operating manuals, programming references, and the course materials for an Advanced Programming Course.
- The FLOW-MATIC brochure from 1957 (includes the FLOW-MATIC sample code above.

LINUX VOICE TUTORIAL

THE AWESOMELY EPIC GUIDE TO KDE

GRAHAM MORRISON

Everything you ever wanted to know about KDE (but were too afraid of the number of possible solutions to ask).

WHY DO THIS?

- Make your desktop look the way you want it to look, not the developer.
- Dazzle your friends with graphical glitz.
- Save time with file manager shortcuts.

Desktops on Linux. They're a concept completely alien to users of other operating systems because they never having to think about them. Desktops must feel like the abstract idea of time to the Amondawa tribe, a thought that doesn't have any use until you're in a different environment. But here it is – on Linux you don't have to use the graphical environment lurking beneath your mouse cursor. You can change it for something completely different. If you don't like windows, switch to xmonad. If you like full-screen apps, try Gnome. And if you're after the most powerful and configurable point-and-click desktop, there's KDE.

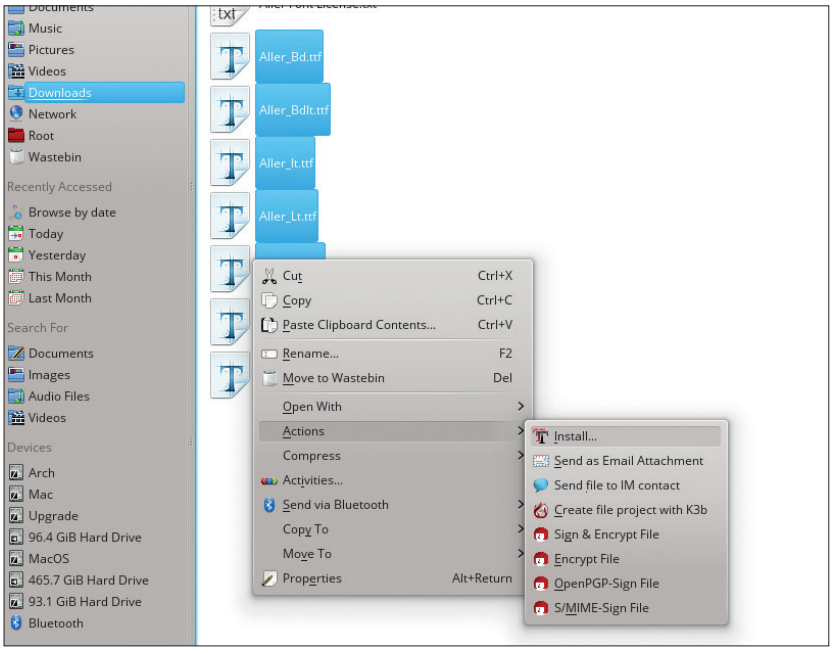
KDE is wonderful, as they all are in their own way. But in our opinion, KDE in particular suffers from poor default configuration and a rather allusive learning curve. This is doubly frustrating, firstly because it has been quietly growing more brilliant over the last couple of years, and secondly, because KDE should be the first choice for users unhappy with their old desktop – in particular, Windows 8 users pining for an interface that makes sense.

But fear not. We're going to use a decade's worth of KDE firefighting to bring you the definitive guide to making KDE look good and function slightly more like how you might expect it to. We're not going to look at KDE's applications, other than perhaps Dolphin; we're instead going to look at the functionality in the desktop environment itself. And while our guinea pig distribution is going to be Mageia 4, as found on this month's DVD, this guide will be equally applicable to any recent KDE desktop running from almost any distribution, so don't let the default Mageia background put you off.

1 FONTS

Most distributions don't include decent fonts. But KDE enables you to quickly install new ones and apply them to your desktop.

A great first target for getting your system looking good is its selection of fonts. It used to be the case that many of us would routinely copy fonts across from a Windows installation, getting the professional Ariel and Helvetica font rendering that was missing from Linux at the time. But thanks to generic quality fonts such as DejaVu and Nimbus Sans/Roman, this isn't a problem any more. But it's still worth finding a font you prefer, as there are now so many great alternatives to choose between.



The best source of free fonts we've found is **www.fontsquirrel.com** – it hosts the Roboto, Roboto Slab (Hello!) and Roboto Condensed (Hello!) typefaces used throughout this magazine, and also on the Nexus 5 smartphone (Roboto was developed for use in the Ice Cream Sandwich version of the Android mobile operating system).

TrueType fonts, with their **.ttf** file extensions, are incredibly easy to install from KDE. Download the zip file, right-click and select something from the Extract menu. Now all you need to do is drag a selection across the TrueType fonts you want to install and select 'Install' from the right-click Actions menu. KDE will take care of the rest.

Another brilliant thing about KDE is that you can change all the fonts at once. Open the System Settings panel and click on Application Appearances, followed by the fonts tab, and click on Adjust All Fonts. Now just select a font from the requester. Most KDE applications will update with your choice immediately, while other applications, such as Firefox, will require a restart. Either way, it's a quick and effective way of experimenting with your desktop's usability and appearance. We'd recommend either Open Sans or the thinner Aller fonts.

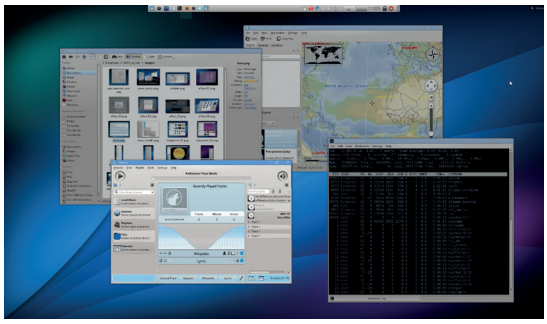
2 EYE CANDY

One of KDE's most secret features is that backgrounds can be dynamic. We don't find much use for this when it comes to the desktops that tells us the weather outside the window, but we do like backgrounds that dynamically grab images from the internet. With most distributions you'll need to install something for this to work. Just search for **plasma-wallpaper** in your distribution's package manager. Our favourite is **plasma-wallpaper-potd**, as this installs easy access to update-able wallpaper images from a variety of sources.

Changing a desktop background is easy with KDE, but it's not intuitive. Mageia, for example, defaults to using 'Folder' view, as this is closer to the traditional desktop where files from the Desktop folder in your home directory are displayed on the background, and the whole desktop works like a file manager. Right-click and select 'Folder Settings' if this is the view you're using. Alternatively, KDE defaults to 'Desktop', where the background is clear apart from any widgets you add yourself, and files and folders are considered links to the sources. The menu item in this mode is labelled Desktop Settings. The View Configuration panel that changes the background is the same, however, and you need to make your changes in the Wallpaper drop-down menu. We'd recommend Picture Of The Day as the wallpaper, and the Astronomy Picture Of The Day as the image source.

In the glow ring

Another default option we think is crazy is the blue glow that surrounds the active window. While every other desktop uses a slightly deeper drop-shadow, KDE's active window looks like it's bathed in radioactive light. The solution to this lies in the default theme, and this can be changed by going to KDE's System Settings control panel and selecting Workspace Appearance. On the first page, which is labelled Window Decorations, you'll find that Oxygen is nearly always selected, and it's this theme that contains the option to change the blue glow. Just click on the Configure Decoration button, flip to the Shadows tab and disable Active Window Glow'.



Remove the blue glow and change a few of the display options, and KDE starts to look pretty good in our opinion.

Alternatively, if you'd like active windows to have a more pronounced shadow, change the inner and outer colours to black.

You may have seen the option to download wallpapers, for example, from within a KDE window, and you can see this now by clicking on the Get New Decorations button. Themes are subjective, but our favourite combination is currently the Chrome window decoration (it looks identical to Google's default theme for its browser) with the Aya desktop theme. The term 'desktop theme' is a bit of a misnomer, as it doesn't encapsulate every setting as you might expect. Instead it controls how generic desktop elements are rendered. The most visible of these elements is the launch panel, and changing the desktop theme will usually have a dramatic effect on its appearance, but you'll also notice a difference in the widgets system.

The final graphical flourish we'd suggest is to change the icon set that KDE uses. There's nothing wrong with the default Oxygen set, but there are better options. Unfortunately, this is where the 'Get New Themes' download option often fails, probably because icon packages are large and can overwhelm the personal storage space often reserved for projects like these. We'd suggest going to **kde-look.org** and browsing its icon collections. Open up the Icons panel from KDE's System Settings, click on the Icons tab followed by Install Theme File and point the requester at the location of the archive you just downloaded. KDE will take it from there and add the icon set to the list in the panel. Try Kotenza for a flat theme, or keep an eye on Nitrux development.

LV PRO TIP

Move any window by holding Alt and click+dragging the window with your mouse. This also applies to the KRunner dialog and the Plasma cashew widget.

3 THE PANEL

Our next target is going to be the panel at the bottom of the screen. This has become a little dated, especially if you're using KDE on a large or high-resolution display, so our first suggestion is to re-scale and centre it for your screen. The key to moving screen components in KDE is making sure they're unlocked, and this accomplished by right-clicking on the 'plasma' cashew in the top-right of the display where the current activity is listed. Only when widgets are unlocked can you re-size the panel, and even add new applications from the launch menu.

With widgets unlocked, click on the cashew on the side of the panel followed by More Settings and select Centre for panel alignment. With this enabled you can re-size the panel using the sliders on either side and the panel itself will always stay in the middle of your screen. Just pretend you're working on indentation on a word processor and you'll get the idea. You can also

Activities

No article on KDE would be complete without some discussion of what KDE calls Activities. In many ways, Activities are a solution waiting for a problem. They're meta-virtual desktops that allow you to group desktop configuration and applications together. You may have an activity for photo editing, for example, or one for working and another for the internet. If you've got a touchscreen laptop, activities could be used to switch between an Android-style app launcher (the Search and Launch mode from the Desktop Settings panel), and the regular desktop mode. We use a single activity as a default for screenshots, for instance, while another activity switches everything to the file manager desktop mode. But the truth is that you have to understand what they are before you can find a way of using them.

Some installations of KDE will include the Activity applet in the toolbar. Its red, blue and green dots can be clicked on to open the activity manager, or you can click on the Plasma cashew in the top-right and select Activities. This will open the bar at the bottom of the screen, which lists activities installed and primed on your system. Clicking on any will switch between them; as will pressing the meta key (usually the Windows key) and Tab.

We'd suggest that finding a fast way to switch between activities, such as with a keyboard shortcut or with the Activity Bar widget is the key to using them more. With the Activity Manager open, clicking on Create Activity lets you either clone the current desktop, add a blank desktop or create a new activity from a list of templates. Clone works well if you want to add some default applications to the desktop for your current setup. To remove an activity, switch to another one and press the Stop and Delete buttons from the Activity Manager.

Activities let you quickly switch between different desktop modes, such as the search and launch mode, which is ideal for tablets.

change its height when the sliders are visible by dragging the central height widget, and to the left of this, you can drag the panel to a different edge on your screen. The top edge works quite well, but many of KDE's applets don't work well when stacked vertically on the left or right edges of the display.

There are two different kinds of task manager applets that come with KDE. The default displays each running application as a title bar in the panel, but this takes up quite a bit of space. The alternative task manager displays only the icon of the application, which we think is much more useful. Mageia defaults to the icon version, but most others – and KDE itself – prefer the title bar applet. To change this, click on the cashew again and hover over the old applet so that the 'X' appears, then click on this 'X' to remove the applet from the panel. Now click on Add Widgets, find the two task managers and drag the icon version on to your panel. You can re-arrange any other applets in this mode by dragging them to the left and right.

LV PRO TIP

Spacers can be added to your panel so that icons don't push up against one another. This is great for separating quick launchers from the task manager.

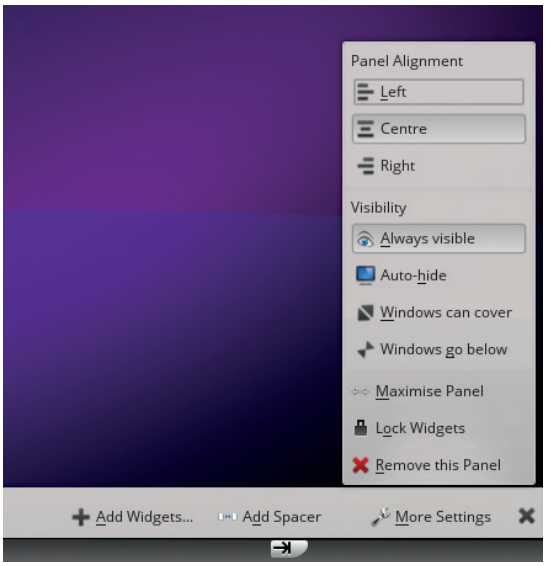
More sensible defaults

By default, the Icon-Only task manager will only display icons for tasks running on the current desktop, which we think is counterintuitive, as it's more convenient to see all of the applications you may have running and to quickly switch between whatever desktops on which they may be running with a simple click. To change this behaviour, right-click on the applet and select the Settings menu option and the Behaviour tab in the next window. Deselect 'Only Show Tasks From The Current Desktop', and perhaps 'Only Show Tasks From The Current Activity' if you use KDE's activities.

Another alteration we like to make is to reconfigure the virtual desktops applet from showing four desktops as a 2x2, which doesn't look too good on a small panel, to 4x1. This can be done by right-clicking on the applet, selecting Pager Settings and then clicking on the Virtual Desktops tabs and changing the number of rows to '1'.

Finally, there's the launch menu. Mageia has switched this from the new style of application launcher to the old style originally seen in Microsoft Windows. We prefer the former because of its search field, but the two can be switched by right-clicking the icon and selecting the Switch To... menu option.

If you find the hover-select action of this mode annoying, where moving the mouse over one of the categories automatically selects it, you can disable it by right-clicking on the launcher, selecting Launcher Settings from the menu and disabling 'Switch Tabs On Hover' from the General settings page. It's worth reiterating that many of these menu options are only available when widgets are unlocked, so don't despair if you don't see the correct menu entry at first.

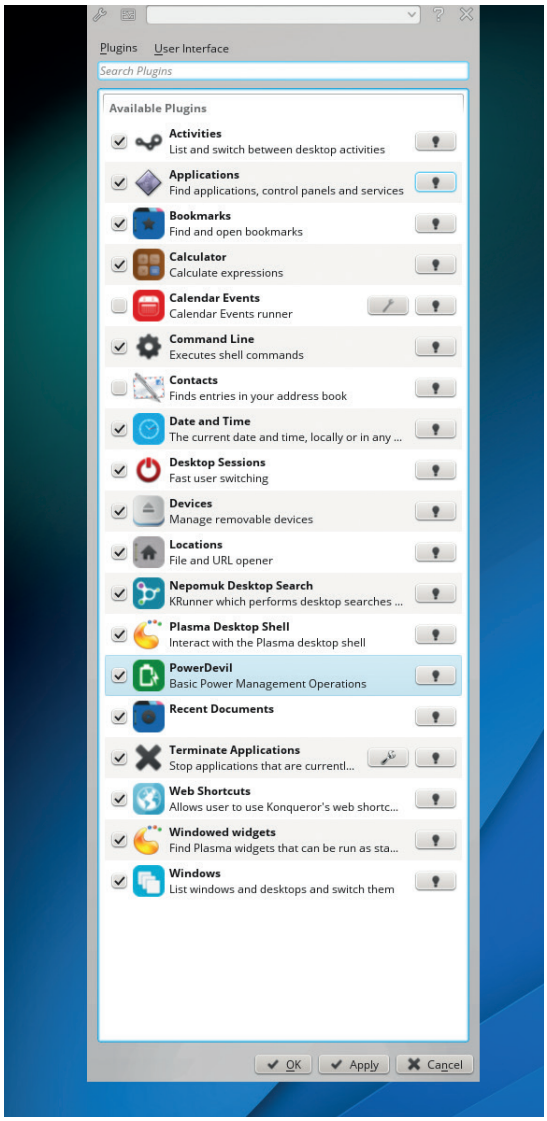


We'd recommend reducing the size and centrally scaling the KDE launch panel.

4 UPGRADED LAUNCH MENU

You may want to look into replacing the default launch menu entirely. If you open the Add Widgets view, for instance, and search for menus, you'll see several results. Our current favourite is called Application Launcher (QML). It provides the same kind of functionality as the default menu, but has a cleaner interface after you've enlarged the initial window. But if we're being honest, we don't use the launcher that much. We prefer to do most launching through KRunner, which is the seemingly simple requester that appears when you hold Alt+F2.

KRunner is better than the default launcher, because you can type this shortcut from anywhere, regardless of which applications are running or where your mouse is located. When you start to type the name of the application you want to run into KRunner, you'll see the results filtered in real time beneath the entry field – press Enter to launch the top choice.



KRunner isn't a great name, but it's one of the most powerful parts of the KDE desktop, doing away with almost every other element of the GUI.

More than just a launcher

KRunner is capable of so much more. You can type in calculations like **=sin(90)**, for example, and see the result in real time. You can search Google with **gg:** or Wikipedia with **wp:** followed by the search terms, and add many other operations through installable modules. To make best use of this awesome KDE feature, make sure you've got the **plasma-addons** package installed, and search for **runner** on your distribution's package manager. When you next launch KRunner and click on the tool icon to the left of the search bar, you'll see a wide variety of plugins that can do all kinds of things with the text you type in. In classic KDE style, many don't include instructions on how to use them, so here's our breakdown of the most useful things you can do with KRunner:

The 11 most useful KRunner commands

kill <process>	Terminate the selected process.
#<command	Open the man page for the command.
<argument>	Open a website, app or document.
file:/	Launch Dolphin on the root directory.
smb://<share>	Open a Samba share in Dolphin.
sftp://<SSH site>	Open an SFTP folder in Dolphin.
vnc://<server:1>	Access a remote desktop.
desktop 2	Switch to desktop 2.
window <app>	List and switch between windows.
<name@server>	Send an email to name@server.
=solve(x-20=9)	Solve equations plus many other functions.

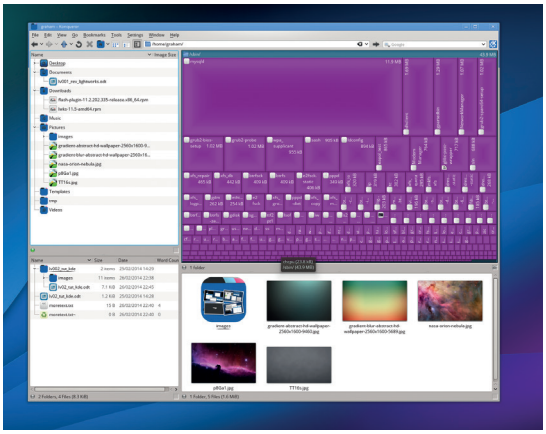
5 FILE MANAGEMENT

File management may not be the most exciting subject in Linux, but it is one we all seem to spend a lot of time doing, whether that's moving a download into a better folder, or copying photos from a camera. The old file manager, Konqueror, was one of the best reasons for using KDE in the first place, and while Konqueror has been superseded by Dolphin in KDE 4.x, it's still knocking around – even if it is labelled a web browser.

If you open Konqueror and enter the URL as **file:/**, it turns back into that file manager of old, with many of its best features intact. You can click on the lower status bar, for example, and split the view vertically or horizontally, into other views. You can fill the view with proportionally sized blocks by selecting Preview File Size View from the right-click menu, and preview many other file types without ever leaving Konqueror.

LV PRO TIP

Use your mouse wheel on KDE's desktop background to switch between desktops.



Konqueror may be vanquished, but many of its best features have made it into the Dolphin file manager.

Click control

Mageia uses a double-click for most options, whereas we prefer a single click. This can be changed from the System-Settings panel by opening Input Devices, clicking on Mouse and enabling 'Single-click To Open Files And Folders'. If you've become used to Apple's reverse scroll, you'll also find an option here to reverse the scroll direction on Linux.

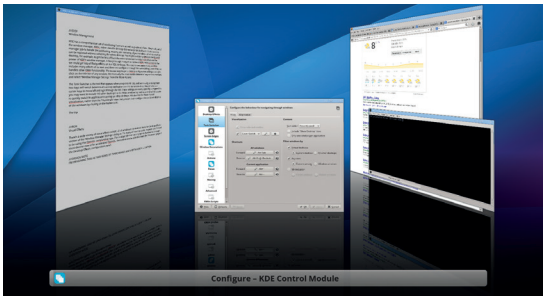
Konqueror is a great application, but it hasn't been a focus of KDE development for a considerable period of time. Dolphin has replaced it, and while this is a much simplified file manager, it does inherit some of Konqueror's best features. You can still split the view, for instance, albeit one only once, and only horizontally, from the toolbar. You can also view lots of metadata. Select the Details View and right-click on the column headings for the files, and you can add columns that list the word counts in text files, or an image's size and orientation, or the artist, title and duration of an audio file, all from within the contents of the data. This is KDE's semantic desktop in action, and it's been growing in functionality for the last couple of years. Apple's OS X, for example, has only just started pushing its ability to tag files and applications – we've been able to do this from KDE for a long time. We don't know any other desktop that comes close to providing that level of control.

6 WINDOW MANAGEMENT

KDE has a comprehensive set of windowing functions as well as graphical effects. They're all part of the window manager, KWin, rather than the desktop, which is what we've been dealing with so far. It's the window manager's job to handle the positioning, moving and rendering of your windows, which is why they can be replaced without switching the whole desktop. You might want to try KWin on the RazorQt desktop, for example, to get the best of both the minimal environment RazorQt offers and the power of KDE's window manager.

The easiest way to get to KWin's configuration settings is to right-click on the title bar of any window (this is usually the most visible element of any window manager), and select Window Manager Settings from the More Actions menu.

The Task Switcher is the tool that appears when you press Alt+Tab, and continually pressing those two keys will switch between all running applications on the current desktop. You can also use cursor keys to move left and right through the list. These settings are mostly sensibly configured, but you may want to include All Other Desktops in the Filter Windows By section, as that will allow you to quickly switch to applications running on other desktops. We also like the Cover Switch visualisation rather than the Thumbnails view, and you can even configure the perceived distance of the windows by clicking on the toolbar icon.



KDE is perhaps the best desktop for people who run applications as windows, rather than full screen.

The next page on the window manager control module handles what happens at the edges of your

screen. At the very least, we prefer to enable Switch Desktop On Edge by selecting Only When Moving Windows from the drop-down list. This means that when you drag a window to one edge, the virtual desktop will switch beneath, effectively dragging the window on to a new virtual desktop.

The great thing about enabling this only for dragged windows is that it doesn't interfere with KDE's fantastic window snapping feature. When you drag a window close to the left or right edge, for instance, KDE displays a ghosted window where your window will snap to if you release the mouse. This is a great way of turning KDE into a tiling window manager, where you can easily have two windows split down the middle of the screen area. Moving a window into any of the corners will also give you the ability to neatly arrange your windows to occupy a quarter of the screen, which is ideal for large displays.

Bird's-eye view

We also enable a mode similar to Mission Control on OS X when the cursor is in the region of the top-left corner of the screen. On the screen edge layout, click on the dot in the top-right of the screen (or any other point you'd prefer) and select Desktop Grid from the drop-down menu that appears. Now when you move to the top-right of your display, you'll get an overview of all your virtual desktops, any of which can be chosen with a click.

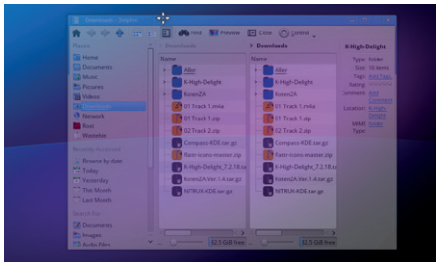
Two pages down in the configuration module, there's a page called Focus. This is an old idea where you can change whether a window becomes active when you click on it, or when you roll your mouse cursor over it. KDE adds another twist to this by providing a slider that progresses from click to a strict hover policy, where the window under the cursor always becomes active. We prefer to use one of the middle options – Focus Follows Mouse – as this chooses the most obvious window to activate for us without making too many mistakes, and it means we seldom click to focus. We also reduce the focus delay to 200ms, but this will depend on how you feel about the feature after using it for a while.

KDE has so many features, many of which only come to light when you start to use the desktop. It really is a case of developers often adding things and then telling no one. But we feel KDE is worth the effort, and unlikely some other desktops, is unlikely to change too much in the transition from 4.x to 5. That means the time you spend learning how to use KDE now is an investment. Dive in!.

Graham Morrison is the editor and only KDE user on the Linux Voice team. He likes weird synthesizers.

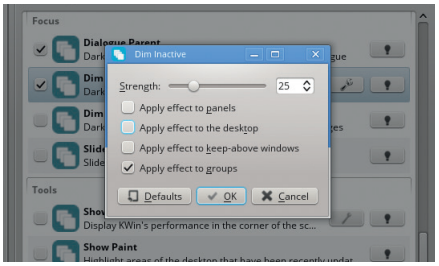
Visual effects

There's a wide variety of visual effects in KDE, all of which can be enabled from the Desktop Effects section of the Window Manager Settings dialog. For many of them to work, however, you'll need to be using the OpenGL compositing type. This is dependent on your graphics hardware: although most devices now offer accelerated OpenGL, the option can be selected from the Advanced page of the Desktop Effects configuration panel. If you run 3D games or other 3D full-screen applications, you should also enable the 'Suspend Desktop Effects For Fullscreen Windows' option to maximise performance. Here's a selection of our favourite desktop effects, some of which have a functional reason to exist:

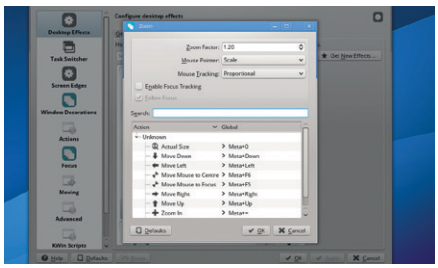


Translucency: The window you're dragging becomes partly translucent. Options can be used to adjust for any kind of window and element.

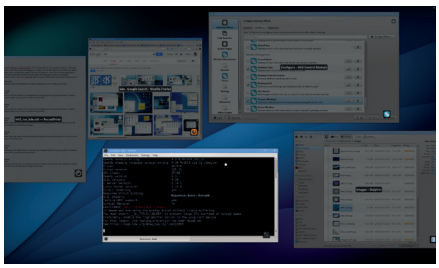
Magic Lamp: When minimising/maximising windows the window will stretch and zoom into the toolbar. It's useful for checking up on your minimised apps.



Dim Inactive: Windows that aren't currently active will go slightly dimmer. We prefer to lessen this effect to a strength of 5 from the Tools page.



Zoom: Hold down the system meta key (usually the Windows one) and press plus or minus to zoom the desktop around the cursor.



Present Windows: This effect works in a similar way to Apple's Expose. Press Ctrl+F10 to display thumbnails of all running desktop applications.

Wobbly Windows: OK, there's no functional reason to enable this other than the endorphin released by contentment. Use the options to change the amount.

LINUX VOICE
TUTORIAL

UEFI BOOTING BOOT CAMP (REBOOTED)

GRAHAM MORRISON

Upgrade your the way your system boots without installing a distribution or resorting to Grub.

We've been using the BIOS for decades. It's as perennial as your keyboard and mouse, breathing life into inert hardware when a little electricity is applied. These days, the POST status messages delivered after your BIOS initialises the system race across the screen so quickly you seldom get the chance to read the text, making entering the BIOS itself a mad keyboard-bashing mini-game that more often than not ends with Grub than the configuration menus you're after. Modern PCs aren't well suited to the old-school charm of the BIOS. They don't want to wait for permission, they don't want low-res large white fonts on a blue background. They just want to get on with the job at hand, and that's booting your computer.

And so the BIOS is being wheeled out, albeit slowly, while its replacement makes itself comfortable. Initially developed by Intel, the booting heir was called the EFI – the Extensible Firmware Interface. But it's now better know as UEFI. The U is for unified, because it's not just Intel anymore. UEFI has been hanging over the Linux boot system like the Sword of Damocles, threatening to upend the booting status quo and exclude us from installing our own operating systems, thanks to the spectre of Secure Boot. Secure Boot is a system that embeds a key without your firmware so that only operating systems signed by the key are allowed to boot. It's primarily a way for Microsoft – in part, legitimately – to ensure nothing has been tampered with from the very first moment your PC gets power to the moment you get to play with the inspirational Windows 8.1 interface. But it could also make life harder for when you do intentionally want to tamper with your PC by making the choice to install another operating system. In reality, the Secure Boot cataclysm has yet to materialise, as many PCs still include a traditional BIOS or allow you to disable Secure Boot. The latter option should always be available, and you'll need to disable Secure Boot unless you want to start dealing with signing a bootloader shim.

It doesn't look much, but this is the Refind bootloader running from our new EFI partition.

Muddy waters

Another potentially confusing option is something called the Compatibility Support Module. To the user, this will appear as a hybrid between UEFI and the BIOS, a magical panacea that seems to allow us to forget about UEFI and BIOS completely. You'll typically see its effects from your computer's own boot device selection menu, usually the one you get when you hold F12 after turning on your machine. What's not always made clear is that the mode you boot into from this point will affect how your Linux distribution installs itself, which in turn affects whether you'll be able to boot Linux from a UEFI boot. An installer won't install a UEFI bootloader, for instance, unless you boot into UEFI mode. And if your install medium doesn't support a UEFI bootloader, you're stuck.

But defaulting to a UEFI installation and forgetting about the BIOS and the Compatibility Support Module is beginning to make more sense. Modern laptops are often pre-configured to boot UEFI, and there will be a time when falling back to the BIOS won't be an option. But these days, there's nothing to be scared of, and in many ways, UEFI can make the whole booting process more transparent. The bootloaders may, at the moment, feel slightly more primitive that their well worn BIOS equivalents, but to us the boot process actually makes more sense than the black arts involved in the old methods. If you've spent the last decade thinking about booting in terms of MBR bootloaders, Grub and old-style partitions, get ready to update your notes.

We're going to create our own UEFI boot environment, and we'll be doing this primarily from the Mint Live desktop as found on last month's DVD, in much the same way you might fix a broken MBR installation or reconfigure Grub. You can use any similar distribution, however, as there's nothing Mint-specific about our instructions. We're also going to use a 1GB USB stick to get around the limitation of BIOS-only booting DVD drives, but we'll only use this to 'fix' the installation, rather than initiate it.

The system we create won't be perfect. It won't handle distribution updates to the kernel without a little further tinkering, and you'll need to make plenty of considerations for your own hardware rather than these instructions for ours. But you will learn how UEFI works from a practical perspective, and learn how to troubleshoot the future of Linux booting.

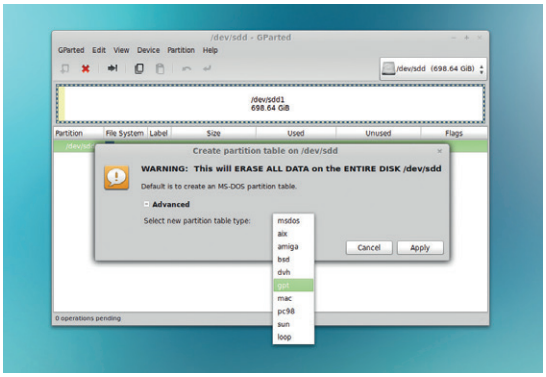
Look into the black box

The great thing about taking control of UEFI yourself is that you don't have the problem of which mode your system has booted from – UEFI or BIOS, which is especially useful if you're booting off a DVD that can only boot in the old BIOS mode. When you get one distribution running , it's easy to add more, and it can also be the only way of running the latest Microsoft Windows or even Apple's OS X alongside.

Mint 16 and many other distributions have their own preliminary support for UEFI bootloaders, as long as you've booted into the correct boot mode, but we've found its approach a little unpredictable, along with many other distributions. We had similar problems with Mageia, for example. Which is why we want to roll our own – the intention being to learn more about how it works and how you might approach installation with a distribution that doesn't support UEFI. And the real trick isn't installing the distribution, it's configuring your drive in such a way that it works with UEFI. The most important part is booting to a Live distribution,

But before we get to the booting part, we need to start with partitioning. To boot UEFI, need to use a different partitioning scheme. So you'll need a spare drive – or one you're willing to sacrifice, as all the data it contains will be removed in the process, and you'll need to be confident about your current drive configuration. We're going to be reformatting the drive and you don't want to overwrite or repartition personal data in the process of experimentation, so it may even be wise to disconnect any other drives. With all that in mind, locate your nearest Linux live CD and USB stick and boot your machine.

There's nothing wrong with the command line, but when it comes to partitioning drives, we like the visual safety net provided by GParted. Fortunately, this essential application is part of most live distributions,



You need to create a GPT partition table for UEFI booting.

Depending on which boot option you take, your system will boot into either UEFI or BIOS boot modes.

and you'll find it in Mint 16's Administration menu. It's an application that hides a lot of power. In the top-right you'll find a drop-down list of all the drives detected and connected to your system. When you select one of these drives, the horizontal bar beneath the menu will become populated with a graphical representation of the partitions on that drive. Each partition is a self-contained horizontal block and its border colour is used to show the filesystem used for each partition. Within each partition, a yellow bar is used to indicate how much space is taken up by data, with white used to indicate free space on the partition. This is handy if you want to use free space to resize a partition.

Danger: partitioning!

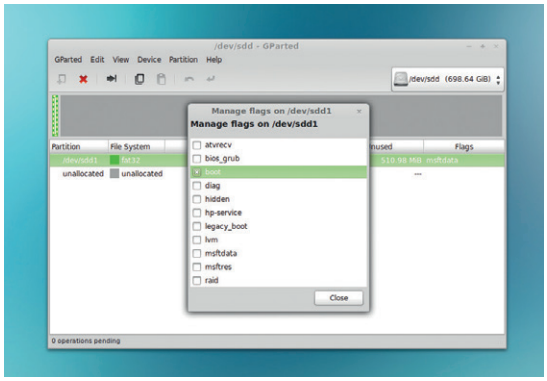
Make sure you select the correct drive from the drop-down list. If you've only got one drive installed, this isn't going to be a problem. If you've got five, you need to be certain the drive you're selecting is the one you intend to partition for a UEFI bootloader, because you're going to remove all the data on the drive in the process. Our drive, for example, already has a Linux partition on it, but this is going to disappear in the very next paragraph – you have been warned.

The old partitioning scheme used a table to store the partition data, and this table was stored on the Master Boot Record (MBR), a statically located 512 bytes allocated to explain the layout of a drive to the BIOS. Nearly all Linux drives prior to UEFI used MBR, and MBR can still be used in some cases with UEFI. But it's better to make clean break. The first thing we need to do with our drive is create a new partition table.

With your drive definitely selected, click on a partition on the drive and select Device > Create Partition Table from the menu. From the dialog that appears, click on 'Advanced' and while avoiding the temptation to click on 'amiga', select 'gpt' as the partition type followed by Apply. All the data on that drive is effectively dead to us now, and you'll see there are no partitions on your drive. Just the cold grey of unallocated space.

LV PRO TIP

If you're installing Linux alongside Windows, make sure you disable Fast Startup and Secure Boot.



It's vital that the EFI partition you create has a partition type of EF00. Either use cgdisk on the command line or enable the 'boot' flag for the partition in GParted.

We're now going to create a couple of partitions to fill the space, but the first is mandatory. This is the EFI system partition, and it's this that UEFI expects to find on your drive and where it will eventually find your UEFI bootloader. For that reason, it's operating system-agnostic, and needs to be formatted as FAT32 for maximum compatibility. It should also be a certain size. The UEFI standard recommends this as 512MB, although in execution we've found that 100MB partitions work just as well. Eventually, you could install Linux kernel images into this partition, so there's no harm in making it larger unless you're working with an expensive SSD. To create this partition, click on the 'plus' icon in the toolbar, set its size to 512MB and make sure it uses the FAT32 filesystem.

The next step is important. If you were doing this from the command line, using a tool like **gdisk**, you'd need to mark this partition as type EF00. This tells UEFI that this is the system partition (also known as the ESP – the EFI System Partition), and it's the one to use for booting. GParted doesn't use hex codes, but you still have to tell UEFI about the partition. You do this by setting the 'boot' flag, which is a little incongruous when you may be used to using a similar flag in MBR systems to tell the BIOS which partition to boot. Right-click on the freshly created partition and select 'Manage Flags'. From the list of flags that appears, select 'boot', this should disable the default 'msftdata' flag as well as cause some drive activity.

With the EFI partition created, assigned a partition type and formatted FAT32, we can now install the bootloader. There are several that work with EFI – and even Grub can be made to work with the new scheme, although you don't win any house points for simplicity of you take that route. The two we tried for this tutorial were Gummiboot and Refind. Both have a couple of things in common. Firstly, their names are terrible. But they're both straightforward to install and use a simple directory structure on your UEFI partition plus a configuration file to hold information on the operating systems you want to boot. We went with Refind.

We've now got to the point where we can install the UEFI bootloader, and there are two stages to the process. The first is to mount the distribution you want to add, and to now make the boot folder the UEFI partition we just created. The second is to move all the files you need to the UEFI partition and add the new UEFI boot scheme to your system firmware so that it knows there's a new way to boot the system.

You will need to know where your distribution is installed. The easiest way of doing this is from GParted's drop-down device menu, as you'll be able to see the device node (**/dev/sda1**, for instance) along with the partition configuration and the UUID of the device if you make a note of it.

To mount the partition, open a terminal and type the following, replacing **sda2** with the location of your own distribution's root partition:

```
sudo -s
mount /dev/sda2 /mnt/
```

With an MBR installation, Grub uses the **/boot** folder to not only hold its configuration files, but also the kernel and filesystem image for booting. We need both of these for UEFI and the UEFI partition needs to replace **/boot** on the filesystem tree. Here's the list of commands we used to move the old boot aside, mount the new one and copy the files we need over (remember to replace filenames and devices with ones that match your own system):

```
cd /mnt
mv boot boot_old
mkdir boot
mount /dev/sda1 /mnt/boot
mkdir boot/EFI
cp  boot_old/vmlinuz-3.11.0-12-generic boot/vmlinuz
cp  boot_old/initrd.img-3.11.0-12-generic boot//initrd.img
```

We now need to add the new UEFI partition as a mount point, and to do this we need to add the partition's unique identifier (its UUID) to the **etc/ftsab** configuration file. You can get the UUID from GParted or by typing the following:

```
blkid
/dev/sda1: UUID="BD8C-E7B3" TYPE="vfat"
/dev/sda2: UUID="0abcc4da-c2aa-437b" TYPE="ext4"
```

We've shortened the output slightly, but you can see the UUID for the UEFI 'vfat' partition on the first line. This needs to be added as a new line in **etc/fstab** on your distribution's root partition by editing the file with **nano etc/fstab**:

```
UUID=BD8C-E7B3 /boot/efi       vfat    defaults        0       2
```

Installing the bootloader

We can now install the bootloader itself. If we'd been able to boot into the distribution using UEFI, we could simply install this through a package manager and everything else would be handled automatically. But because our system is currently booted from BIOS mode, we need to copy the files manually, edit a config file and then add the bootloader to the UEFI firmware by booting in UEFI mode off a USB stick.

Let's first download the binary version of the Refind bootloader (**refind-bin-0.7.7.zip**) plus the image of the same bootloader (**refind-flashdrive-0.7.7.zip**) we're going to use to boot off the USB stick. Both can be grabbed from **www.rodsbooks.com/refind** via links to SourceForge. To install the bootloader, we need to unzip it and copy the folder to the mounted

LV PRO TIP

GParted can create an incompatible EFI boot partition. If this happens, we'd recommend using the command line tool **cgdisk** to create a EF00 type partition formatted with fat32.

boot partition on our distribution:

```
cd ~/Download
unzip refind-bin-0.7.7.zip
cd refind-bin-0.7.7/
cp -r refind /mnt/boot/EFI/
cd /mnt/boot/EFI/refind
```

From here you need to remove either the 32-bit or the 64-bit bootloader, depending on what your system is capable of, with **rm refind_ia32.efi** or **rm refind_x64.efi**, and edit the configuration file (**nano refind.conf**) to add the details about the partition that contains the distribution you want to boot. Here's the contents if ours for booting Mint 16 – you should take a look at your boot options first, to make sure you get any kernel options specific to your system:

```
resolution 1024 768
menuentry "Mint Linux" {
        icon     EFI/refind/icons/os_linuxmint.icns
        loader  vmlinuz
        initrd  initrd.img
        options "root=/dev/sda2 rw rootfstype=ext4
add_efi_memmap"
}
```

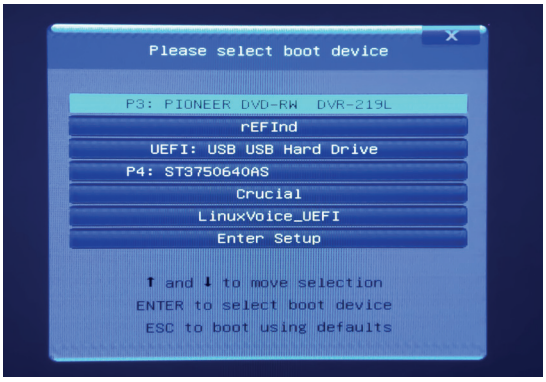
Our final challenge is to tell the UEFI firmware that we've created a new EFI partition and bootloader. Had we been able to boot into the live desktop through UEFI, the firmware variables would be mounted as part of the system, and we'd be able to add the bootloader by typing:

```
sudo apt-get install efibootmgr
efibootmgr -c -l \\EFI\\refind\\refind_x64.efi -L new_refind
```

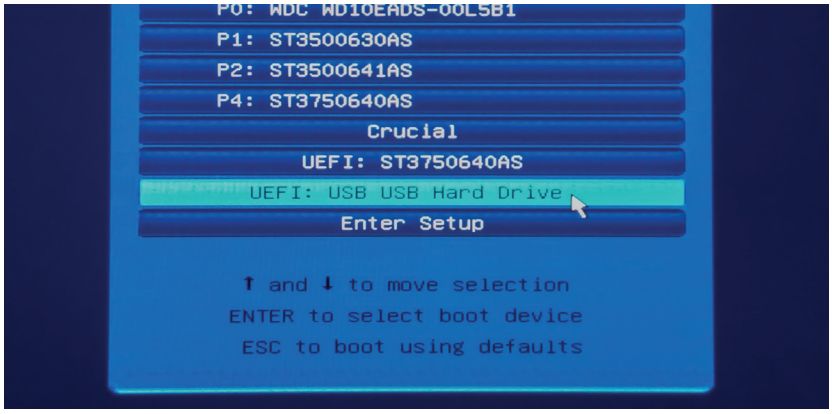
But we can't. Instead, one solution is to create a USB stick with the Refind bootloaders installed, and from there, use the EFI shell to add the bootloader manually. This isn't really what we'd recommend. You're better off installing Mint through a UEFI-booted USB live image, but the EFI shell is much more interesting and can be a very powerful tool if your system doesn't boot. Plug in your USB stick and use either GParted or **dmesg** to find for certain what its device node is and type the following from the unzipped folder of the Refind flash image:

```
dd if=refind-flashdrive-0.7.7.img of=/dev/sde
```

Remember to replace **/dev/sde** with the location of your own USB drive and also remember that this will



With Refind added to the system firmware, our boot entry should appear from the system (press F12) boot menu.



With Refind copied to a USB stick, you will be able to boot into UEFI mode and select the EFI shell.

delete all data at that location, so get it right and make sure there's nothing on there you want to keep. You can now reboot your system and launch your BIOS/system boot menu. You should see the USB stick appear as a UEFI boot source. Select this and from the graphical boot menu that appears, choose the first option, which should take you to the EFI shell.

Welcome to your new shell

The EFI shell is full of commands for adding, removing and managing storage from the EFI bootloader. Before you get to the prompt itself, you'll see how EFI is interpreting your various filesystems and the aliases it's giving them. For us, **fs0:** was the USB drive and **fs1:** was the EFI partition we just created on the hard drive, but these assignments will depend on your own system. From the command prompt, type **fs1:** to switch to the root folder of our new EFI partition. The EFI shell is crammed full of commands to help you manage storage and booting. Type **help** if you want to see what it's capable of – you can use **ls**, **cp** and **rm**, for example. But we're only going to use one command to add our bootloader to the system firmware. We're assuming you don't have any other EFI boot loaders installed, because using one of them would have been a much easier solution for all of this, but you can check by typing **bcfg boot dump -b**. If you do have another installed, you'll need to adjust the number **1** to a free slot in the command below, which is going to add the new bootloader to the firmware:

```
bcfg boot add 1 fs1:\EFI\refind\refind_x64.efi "LV_Refind"
**bcfg instructions output
Target = 0001.
bcfg: Add Boot0001 as 1
```

And that 's all there is too it. It's been a challenge, but when you now reboot your machine (type **reset** from the EFI shell), you'll see LV_Refind as a new EFI boot option. Hopefully, you've learnt how UEFI works and how it's implemented, and also how you might be able to troubleshoot UEFI problems in the future.

Adding new distributions, for instance, is now a case of copying their kernel and filesystem images to the partition and adding a new configuration entry. You might also want to look into making symbolic links for these files for when your distribution updates itself. Other than that, you're ready to go. LV

LINUX VOICE TUTORIAL

PYTHON DRAWING PRETTY PATTERNS

BEN EVERARD

Use the Python turtle to illustrate loops and recursion, and prove that not all art is quite useless.

WHY DO THIS?

- Gain a better understanding common programming techniques
- Draw pretty pictures
- Win a T-shirt!

There are some programming techniques, like loops and recursion, that we use all the time, almost without thinking. However, sometimes it's hard to really see what's going on. Being able to really visualise what's going on behind the code can help you become a better programmer.

Python comes with a turtle module. It's about as simple as a drawing program can be. It enables you to control a turtle with a pen around the screen. You can tell it to go forwards, turn through various angles, put the pen down or lift it up, and change its colour. In a world where almost everything seems to have OpenGL accelerated graphics, sometimes it's nice to take a step back and look at what you can create with very little. In this tutorial, we're going to look at how we can build complex pictures using just this turtle module and few coding techniques.

Let's start really simply, and just draw a square:

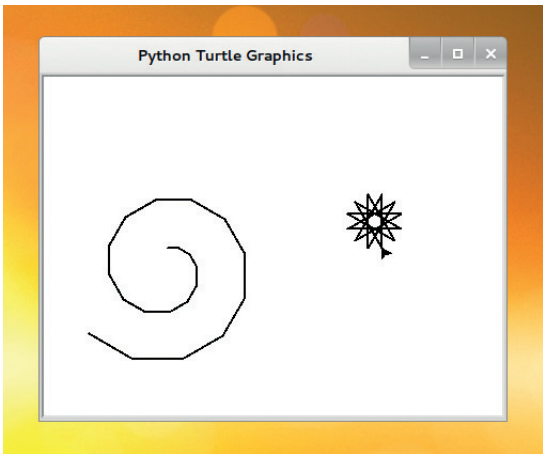
```
import turtle
jonney  = turtle.Turtle()
for i in range(0,4):
  jonney.forward(100)
  jonney.right(90)
turtle.exitonclick()
```

That's all you need. Python will take care of creating the window to display it in. For some reason this author always feels the need to give the turtles anthropomorphised variable names.

You can turn this code into more general polygon drawing code by moving the loop into a function like as follows:

```
import turtle
def draw_polygon(sides, length):
  for i in range(0,sides):
    jonney.forward(length)
    jonney.right(360/sides)
jonney  = turtle.Turtle()
draw_polygon(6,20)
turtle.exitonclick()
```

Because the turtle module only works in whole numbers, this won't work properly for polygons where 360/sides isn't a whole number, but it's good enough for our purposes.



The empty handed painter from your streets is drawing crazy patterns on your screen, with simple Python.

Intensify the artiness

This is an article about creating art from code, and simple polygons aren't very attractive. With a few tweaks, the function can be made a little more artistic:

```
import turtle
def draw_spiral(angle, length_start, length_increase, sides):
  for i in range(0,sides):
    jonney.forward(length_start+(i*length_increase))
    jonney.right(angle)
def draw_petals(length, number):
  for i in range(0, number):
    jonney.forward(length)
    jonney.right(180-(360/number))
jonney  = turtle.Turtle()
draw_spiral(30, 10, 2, 20)
jonney.penup()
jonney.goto(0,200)
jonney.pendown()
draw_petals(50,20)
turtle.exitonclick()
```

You can also vary the colour through the loops. This both helps you see how the images are drawn, and makes the outcomes a little more impressive. We changed the **draw_petals()** function to the following to

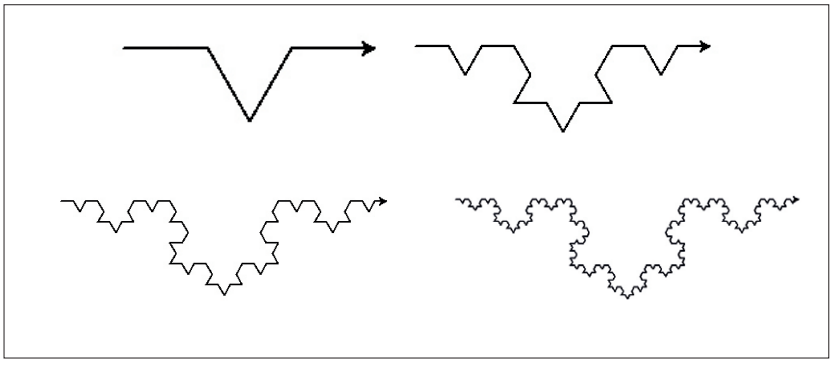


Figure 1. Recursion can quickly build up the number of lines in a fractal as you go to greater depths, so it's best to call **speed(0)** to set it to the fastest speed.

fade the lines from blue through purple to red.

```
def draw_petals(length, number):
  red=0.0
  blue=1.0
  for i in range(0, number):
    red=red + (1.0/number)
    blue = blue - (1.0/number)
    jonney.color(red,0.0,blue)
    jonney.forward(length)
    jonney.right(180-(360/number))
turtle.exitonclick()
```

Fractals and recursion

You can get quite artistic using loops to draw shapes, but you can take drawing a stage further using recursion. (Recursion is just when a function calls itself.) You can use this to progressively process all the data in a set, or to draw pretty pictures.

This might sound a little strange if you've never thought of trying to draw with code, but actually, recursion is a really versatile tool in the coder-artist's toolbox, and it's all thanks to a mathematical trick called fractals.

The idea is really simple. You take a simple line shape like figure 1 part 1. Then you replace every line in the shape with a copy of the lineshape. The result of doing this once is figure 1 part 2. However, you can keep doing it as many times as you like, each time you replace every line with a copy of the original line shape. Figure 1 part 3 shows it done a third time, but assuming you had a high enough resolution display (or if you kept zooming in), you could just go on repeating this more and more times.

The code we used to create figure 1 is:

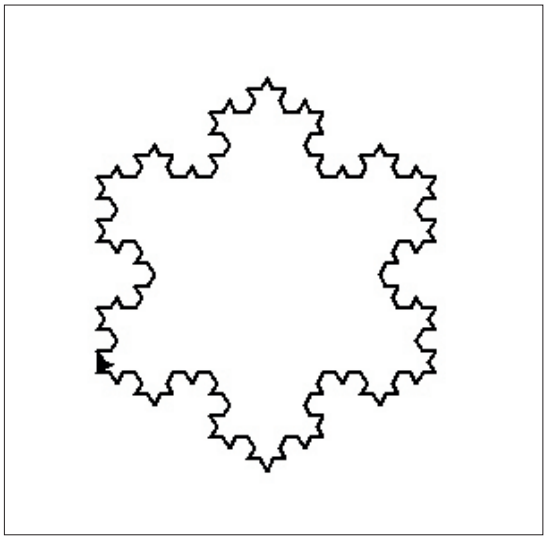
```
import turtle
def draw_fractal(length, depth):
  if depth == 1:
    jonney.forward(length)
  else:
    draw_fractal(length, depth-1)
  jonney.right(60)
  if depth == 1:
    jonney.forward(length)
  else:
    draw_fractal(length, depth-1)
  jonney.left(120)
  if depth == 1:
    jonney.forward(length)
  else:
    draw_fractal(length, depth-1)
  jonney.right(60)
  if depth == 1:
    jonney.forward(length)
  else:
    draw_fractal(length, depth-1)

jonney = turtle.Turtle()
jonney.penup()
jonney.goto(-200,0)
jonney.pendown()
draw_fractal(15,1)
turtle.exitonclick()
```

In this case, we use the **depth** function to limit the number of times the recursion happens, otherwise it could go on indefinitely.

This particular fractal is known as a Kock curve after its creator, Helge von Kock. However, this isn't its best-known form. If you use a triangle for the first iteration, but revert to the line segment for every other iteration, you get a Kock snowflake as shown above.

This is created with the code:

```
import turtle
def draw_snowflake(length, depth):
  draw_fractal(length, depth-1)
  jonney.left(120)
  draw_fractal(length, depth-1)
  jonney.left(120)
  draw_fractal(length, depth-1)
jonney = turtle.Turtle()
jonney.penup()
jonney.goto(-200,0)
jonney.pendown()
draw_snowflake(7,4)
turtle.exitonclick()
```



The Kock snowflake was originally designed by taking a triangle and placing a smaller triangle on every edge, then a smaller triangle on each new edge, etc.

Running Python programs

Python is a very easy programming language to get started in. It's interpreted, which means you don't need to compile the code you've written before you can run it, and it's installed by almost every distribution of Linux by default. What's more, the turtle module is part of the standard Python library, so you don't need to install anything to run the code in this article. Just enter it into a text editor, save the file (a **.py** file extension is usual, but not required), then run it from the command line with:

```
python filename.py
```

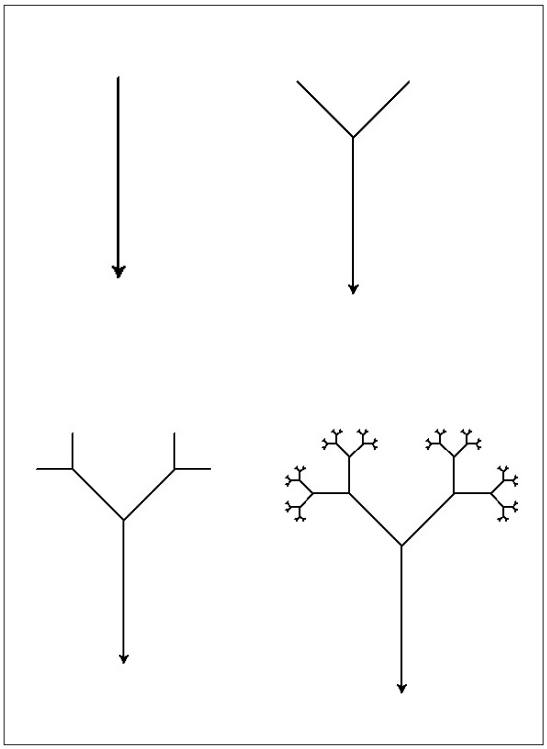
If you don't end your code with the following, then the window will shutdown as soon as it's finished running.

```
turtle.exitonclick()
```

It also includes the **fractal()** function from the previous example.

You can use this same method of recursion in another way. For example, you can use it to continually add lines in a particular place. For example, you can generate a tree by drawing a 'Y' shape, then continually adding smaller 'Y' shapes to the end of each branch. See right for how this works out. In part 1 to 3, the depth is 1 to 3 respectively. Part 4 has a depth of 7. This was generated with the code:

```
import turtle

def draw_tree(length, depth):
  jonney.forward(length)
  if depth > 1:
    jonney.left(45)
    draw_tree(length/2, depth-1)
    jonney.left(90)
    draw_tree(length/2, depth-1)
    jonney.right(135)
  jonney.right(180)
  jonney.forward(length)

jonney = turtle.Turtle()
jonney.penup()
jonney.goto(0,-100)
jonney.pendown()
jonney.left(90)
jonney.speed(0)
draw_tree(160,7)

turtle.exitonclick()
```

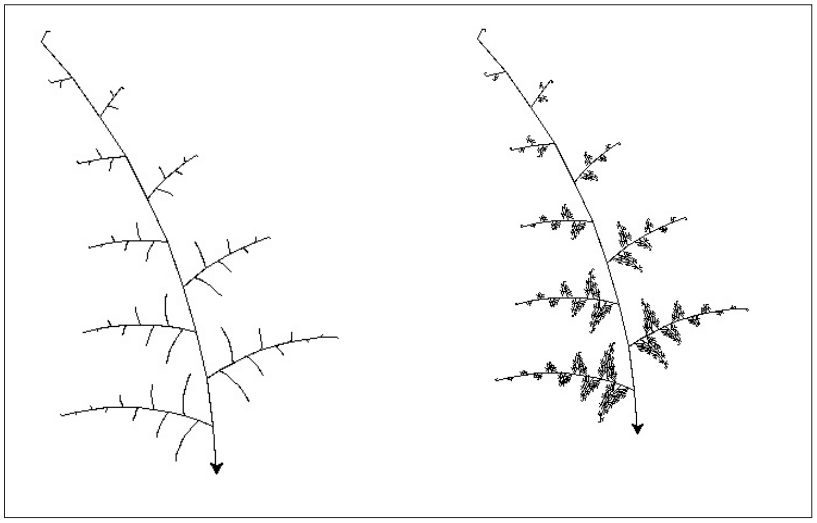


With fractals that get smaller and smaller, you'll quickly reach the limit of the resolution of the screen.

You can extend this. Instead of drawing a Y shape, you can expand a herring bone in the same way (see below). Instead of creating the classic tree shape, this creates a fern-like drawing. The code for this is:

```
import turtle
def floor(x,y):
  if x > y:
    return x
  return y

def draw_fern(length1, angle1, length2, angle2, depth):
  flip = 1
  for i in range (0, length2):
    jonney.left(angle2)
    jonney.forward(length1)
    if depth > 1:
      jonney.left(angle1*flip)

      draw_fern(floor((length1/3)-(i/2), 1), angle1*flip,
floor(length2-i,1), angle2*flip, depth-1)
      jonney.left(180-angle1*flip)
    flip = flip * -1

  jonney.left(180)

  for i in range(0, length2):
      jonney.forward(length1)
      jonney.right(angle2)

jonney = turtle.Turtle()
jonney.penup()
jonney.goto(0,-100)
jonney.pendown()
jonney.left(90)
jonney.speed(0)
draw_fern(40,60,8,4,1)

turtle.exitonclick()
```

These ferns are depths 3 and 5. We've spaced it out to make it easier to see what's going on, but you can change the parameters to create more realistic plants.



This particular code is very sensitive to the parameters you give it. You also can customise the fractal by changing the way the line lengths are passed to the next level of recursion, or by progressively increasing the angle so that the fern starts to curl towards the end.

If you're feeling really adventurous, you could write a program that flips from the tree recursion to the fern recursion at a certain depth. You can develop fractals like this using different shapes. The key is to make sure that, at the end of each run of the function, you return the turtle to the same place it started from.

Fantastic Mr Fractal

There are loads of possible fractals you could draw, and a quick web search will pull some up. One thing to remember when programming fractals like the fern and the tree is to make sure you always finish the fuction at the same physical location the turtle started it. Otherwise it'll end up chaotic.

What we've covered in this tutorial may seem a bit pointless, flippant even, but we've used exactly the same programming techniques that are used in normal software. By learning how to exploit them to draw shapes, you hone your knowledge of how to structure code, and this can only make you a better programmer whatever language you use.

There are also a few cases where fractals themselves are useful to programmers – for example, in creating complex terrain in video games.

Ben Everard is the co-author of *Learning Python with Raspberry Pi*, soon to be published by Wiley. He's also pretty good at turning foraged fruit into alcohol.

Competition time

You've seen how, by using iteration and loops, you can create complex drawings with very little code. In this competition, we're going to put this to the test. The challenge is to create a Python program that uses the turtle module to draw something. I have to be able to output the code, and the winner will be the person that creates what we think is the best piece of art. To put your coding skills to the test, we're going to give you a limit of 100 lines of Python – no more.

You may have noticed that we haven't really tried to keep our code short in this tutorial and quite a few of the functions we've used can be shortened if needed.

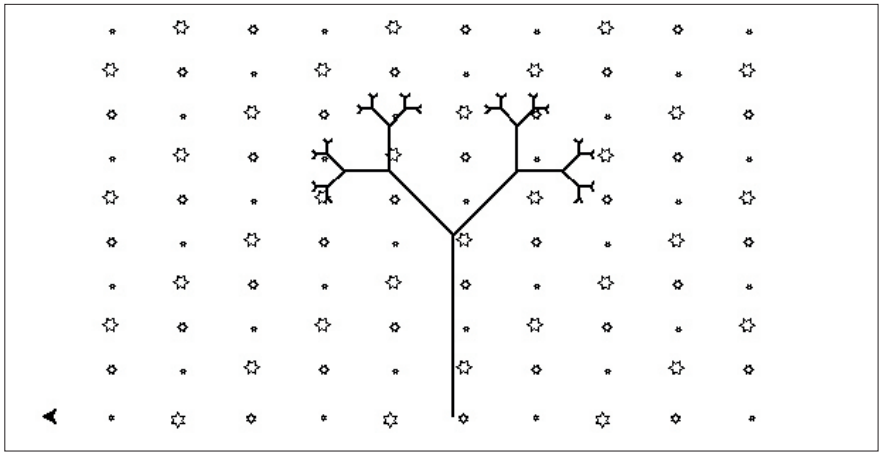
The rules are:

- Using only the turtle module, and no more than 100 lines of Python (2 or 3, your choice), you must draw a picture.
- You may use any of the techniques here, or any others you invent, copy, steal or otherwise come across.
- The only module you can use is the turtle module. Any other import lines will be deleted.
- Pictures will be judged on artistic merit. The judge's decision is final.
- Lines of comments are allowed (and encouraged) and won't be included in the 100 line count. Feel free to add information to your program, and all programs must be licenced under an OSI approved open source licence. Preferably the GPLv3, though you can use a different one should you so choose.
- In order to enter, send your entries to ben@linuxvoice.com by 1 May 2014.
- The winner will receive an exclusive Linux Voice competition winner's T-shirt. These are not, and will not, be available in the shops. The only way to get one is to submit a winning entry to a Linux Voice competition.
- The artwork produced can be in any category. Abstract, still life, impressionism, cubeism. Sculpture is probably out, but otherwise, anything goes as long as it looks good.
- Up to three entries per person will be accepted.
- You don't have to buy a copy of Linux Voice to be eligible. Feel free to pass on the competition details to non LV readers, and details will be posted on **www.linuxvoice.com**.
- Turtles don't have to be called Jonney, and you don't have to limit yourself to a single turtle.

To give you an example of what we're looking for, take a look at figure 5. This was created with the following code (the functions for the **tree()** and **snowflake()** as as they're given earlier in the tutorial, and not repeated here, though if you wish to use them in your example, you WILL have to include them in your 100 lines). Although this picture doesn't make it look like it, the judge does like colour, and garish entries will be looked upon favourably.

```
jonney = turtle.Turtle()
jonney.speed(0)
jonney.left(90)
jonney.width(2)
draw_tree(128,6)
jonney.width(1)
jonney.penup()
jonney.goto(210,0)
for i in range(0, 10):
  for j in range(0,10):
    jonney.pendown()
    draw_snowflake((i+j)%3+1,2)
    jonney.penup()
    jonney.setheading(90)
    jonney.forward(30)
  jonney.setheading(270)
  jonney.forward(300)
  jonney.setheading(180)
  jonney.forward(50)
turtle.exitonclick()
```

Good luck, have fun, and remember that while good artists borrow, great artisits steal!



It may be spring now, but the long, dark winter is still vivid in our memories. Help us forget it with some uplifting artwork, and give yourself the opportunity to win clothing!

LINUX VOICE TUTORIAL

KEY EXCHANGE: THE SCIENCE OF SECURITY

BEN EVERARD

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

WHY DO THIS?

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

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

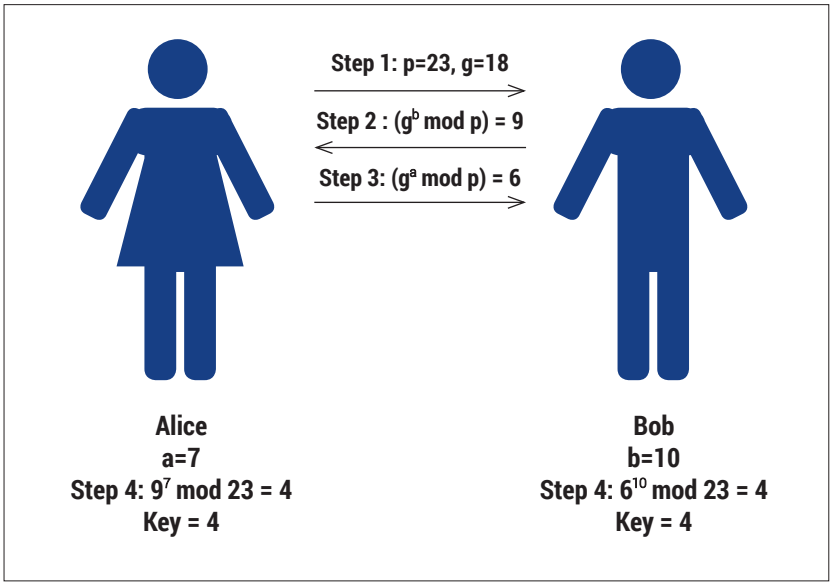
One chink in the electronic armour is the key exchange. That is, the process by which the two parties decide on which key to use for the symmetric encryption. They both have to know the key, so this has to travel between them in order to communicate. However, an attacker may be listening in, and the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

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

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

Stopping this is known as Perfect Forward Secrecy (or PFS), and is possible using the Diffie-Hellman key exchange algorithm, which doesn't use public keys. There's no long-term data that could be compromised that could be used to decrypt past data wholesale.

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



Perfect security

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

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

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

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

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

A quantum of security

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

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

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

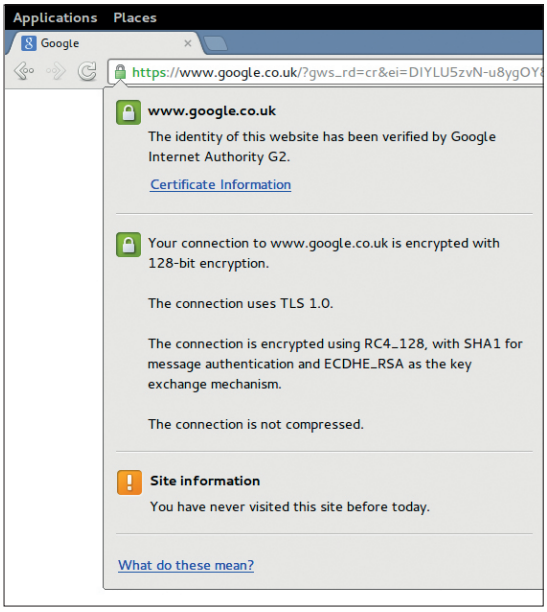
This is possible using the Quantum Key Distribution algorithm. In this method, each bit of information is encoded as a single photon sent from Alice to Bob. The data is in the polarisation. If you think of the light wave travelling through space, the wave could be moving up and down, side to side, or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

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

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

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

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



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

Public and symmetric key encryption

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

In symmetric key encryption (sometimes known as private key encryption), there is just one key to encrypt and decrypt the message.

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

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

Keep GCHQ in the dark

After Alice has sent a long enough string of bits, she sends Bob a list of what orientations she was using for which bits. Bob compares this to how he had his receiving equipment set up. On average, they should have had the same orientation for half of the bits, so Bob replies by saying which bits he was correctly set up for. Both of these messages can go unencrypted since they are no use to an attacker. Alice and Bob can then use the values of bits that Bob received correctly as the key.

> **"Quantum Key Distribution sounds fanciful, but there are already some implementations."**

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

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

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

LINUX VOICE TUTORIAL

LINUX KERNEL: YOUR FIRST MODULE

VALENTINE SINITSYN

Ever wanted to start hacking the kernel? Don't have a clue how to begin? Let us show you how it's done...

WHY DO THIS?

- Understand the common features shared by all kernel modules.
- Earn brownie points in job interviews.
- Tinker with the foundations of your Linux machine.

Kernel programming is often seen as a black magic. In Arthur C Clarke's sense, it probably is. The Linux kernel is quite different from its user space: many abstractions are waived, and you have to take extra care, as a bug in you code affects the whole system. There is no easy way to do floating-point maths, the stack is fixed and small, and the code you write is always asynchronous so you need to think about the concurrency. Despite all of this though, the Linux kernel is just a very large and complex C program that is open for everyone to read, learn and improve, and you too can be a part of it.

Probably the easiest way to start kernel programming is to write a module – a piece of code that can be dynamically loaded into the kernel and removed from it. There are limits to what modules can do – for example, they can't add or remove fields to common data structures like process descriptors. But in all other ways they are full-fledged kernel-level code, and they can always be compiled into the kernel (thus removing all the restrictions) if needed. It is fully possible to develop and compile a module outside the Linux source tree (this is unsurprisingly called an out-of-tree build), which is very convenient if you just want to play a bit and do not wish to submit your changes for inclusion into the mainline kernel.

In this tutorial, we'll develop a simple kernel module that creates a **/dev/reverse** device. A string written to this device is read back with the word order reversed ("Hello World" becomes "World Hello"). It is a popular programmer interview puzzle, and you are likely to get some bonus points when you show the ability to implement it at the kernel level as well. A word of warning before we start: a bug in your module may lead to a system crash and (unlikely, but possible) data loss. Be sure you've backed up all your important data before you start, or, even better, experiment in a virtual machine.

LXR (Linux Cross Reference) is a web-based system for navigating Linux sources (and other large codebases).

The environment

To develop kernel code, you'll need a decent text editor. Choosing one is actually a matter of taste, however many kernel developers tend to use either Vim or Emacs, and Linus prefers his own flavor of the MicroEMACS editor. If you prefer IDEs (although they aren't very useful for kernel development), you can easily google for recommendations how to setup Eclipse CDT.

Regardless the editor you chose, your code should always follow Linux Coding Style guidelines (Documentation/CodingStyle). Probably the easiest way to do this is to use indent(1) utility. Simply run indent --linux-style on your module's source before letting anyone to see it.

A module's anatomy

As most of the Linux kernel modules are written in C (apart from low-level architecture-specific parts), it is recommended that you keep your module in a single file (say, **reverse.c**). To begin, let's include some common headers and describe the module using predefined macros:

```
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/module.h>

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Valentine Sinitsyn <valentine.sinitsyn@gmail.com>");
MODULE_DESCRIPTION("In-kernel phrase reverser");
```

Everything is straightforward here, except for **MODULE_LICENSE()**: it is not a mere marker. The kernel strongly favours GPL-compatible code, so if you set the licence to something non GPL-compatible (say, "Proprietary"), certain kernel functions will not be available to your module.

Since kernel programming is always asynchronous, there is no **main()** function that Linux executes sequentially to run your module. Instead, you provide callbacks for various events, like this:

```
static int __init reverse_init(void)
{
  printk(KERN_INFO "reverse device has been registered\n");
  return 0;
}

static void __exit reverse_exit(void)
{
  printk(KERN_INFO "reverse device has been unregistered\n");
}

module_init(reverse_init);
module_exit(reverse_exit);
```

Here, we define functions to be called on the module's insertion and removal. Only the first one is required. For now, they simply print a message to the kernel ring buffer (accessible from the userspace via the **dmesg** command); **KERN_INFO** is a log level (note there is no comma). **__init** and **__exit** are attributes – the pieces of metadata attached to functions (or variables). Attributes are rarely seen in userspace C code but are pretty common in the kernel. Everything marked with **__init** is recycled after the initialisation (remember the old "Freeing unused kernel memory..." message?). **__exit** denotes functions that are safe to optimise out when the code is built statically into the kernel. Finally, the **module_init()** and **module_exit()** macros set **reverse_init()** and **reverse_exit()** functions as lifecycle callbacks for our module. The actual function names aren't important; you can call them **init()** and **exit()** or **start()** and **stop()**, if you wish. They are declared static and hence invisible outside your module. In fact, any function in the kernel is invisible unless explicitly exported. However, prefixing your functions with a module name is a common convention among kernel programmers.

These are bare bones – let's make things more interesting. Modules can accept parameters, like this:

```
# modprobe foo bar=1
```

The **modinfo** command displays all parameters accepted by the module, and these are also available under **/sys/module/<name>/parameters** as files. Our module will need a buffer to store phrases – let's make its size user-configurable. Add the following three lines just below **MODULE_DESCRIPTION()**:

```
static unsigned long buffer_size = 8192;
module_param(buffer_size, ulong, (S_IRUSR | S_IRGRP | S_
IROTH));
MODULE_PARM_DESC(buffer_size, "Internal buffer size");
```

Here, we define a variable to store the value, wrap it into a parameter, and make it readable by everyone via **sysfs**. The parameter's description (the last line) appears in the modinfo's output.

As the user can set **buffer_size** directly, we need to sanitize it in **reverse_init()**. You should always check the data that comes outside the kernel – if you don't, you are opening yourself to kernel panics or even security holes.

```
static int __init reverse_init()
{
  if (!buffer_size)
    return -1;
  printk(KERN_INFO
      "reverse device has been registered, buffer size is %lu
bytes\n",
      buffer_size);
  return 0;
}

Non-zero return value from a module init function indicates a
failure.
```

Now it's time to compile the module. You will need the headers for the kernel version you are running (**linux-headers** or equivalent package) and **build-essential** (or analogous). Next, it's time to create a boilerplate Makefile:

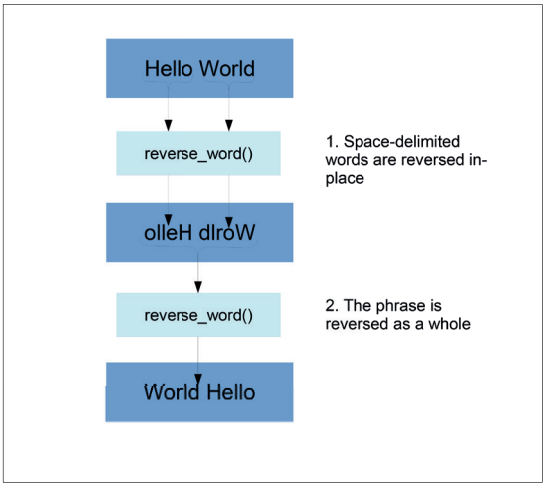
```
obj-m += reverse.o
all:
  make -C /lib/modules/$(shell uname -r)/build M=$(PWD)
modules
clean:
  make -C /lib/modules/$(shell uname -r)/build M=$(PWD)
clean
```

Now, call **make** to build your first module. If you typed everything correctly, you will find **reverse.ko** in

Navigation

The Linux kernel is the ultimate source for everything you may need when developing modules. However, it's quite big, and you may have trouble trying to find what you are after. Luckily, there are tools that make it easier to navigate large codebases. First of all, there is Cscope – a venerable tool that runs in a terminal. Simply run **make cscope && cscope** in the kernel sources top-level directory. Cscope integrates well with Vim and Emacs, so you can use it without leaving the comfort of your favorite editor.

If terminal-based tools aren't your cup of tea, visit **http://lxr.free-electrons.com**. It is a web-based kernel navigation tool with not quie as many features as Cscope (for example, you can't easily find usages for the function), but it still provides enough for the quick lookups.



Reversing a word is simple, but it is also a main building block for reversing a phrase.

the current directory. Insert it with **sudo insmod reverse.ko**, and run:

```
$ dmesg | tail -1
[ 5905.042081] reverse device has been registered, buffer size is 8192 bytes
```

Congratulations! However, for now this line is telling lies – there is no device node yet. Let's fix it.

Miscellaneous devices

In Linux, there is a special character device type called "miscellaneous" (or simply "misc"). It is designed for small device drivers with a single entry point, and is exactly what we need. All misc devices share the same major number (10), so the one driver (**drivers/char/misc.c**) can look after all of them, and they are distinguished by their minor numbers. In all other senses, they are just normal character devices.

To register a minor number (and an entry point) for the device, you declare **struct misc_device**, fill its fields (note the syntax), and call **misc_register()** with a pointer to this structure. For this to work, you will also need to include the **linux/miscdevice.h** header file:

```
static struct miscdevice reverse_misc_device = {
  .minor = MISC_DYNAMIC_MINOR,
  .name = "reverse",
  .fops = &reverse_fops
};
static int __init reverse_init()
{
  ...
  misc_register(&reverse_misc_device);
  printk(KERN_INFO ...
}
```

Here, we request a first available (dynamic) minor number for the device named "reverse"; th ellipsis indicates omitted code that we've already seen. Don't forget to unregister the device on the module's teardown:

```
static void __exit reverse_exit(void)
{
  misc_deregister(&reverse_misc_device);
  ...
}
```

The 'fops' field stores a pointer to a struct **file_operations** (declared in **linux/fs.h**), and this is the entry point for our module. **reverse_fops** is defined as:

```
static struct file_operations reverse_fops = {
  .owner = THIS_MODULE,
  .open = reverse_open,
  ...
  .llseek = noop_llseek
};
```

Again, **reverse_fops** contains a set of callbacks (also known as methods) to be executed when userspace code opens a device, reads from it, writes to it or closes the file descriptor. If you omit any of these, a sensible fallback will be used instead. That's why we explicitly set the **llseek** method to **noop_llseek()**, which (as the name implies) does nothing. The default implementation changes a file pointer, and we don't want our device to be seekable now (this will be your home assignment for today).

I open at the close

Let's implement the methods. We'll allocate a new buffer for each file descriptor opened, and free it on close. This is not really safe: if a userspace application leaks descriptors (perhaps intentionally), it may hog the RAM, and render the system unusable. You should always think about these possibilities in the real world, but for the tutorial, it's acceptable.

We'll need a structure to describe the buffer. The kernel provides many generic data structures: linked lists (which are double-linked), hash tables, trees and so on. However, buffers are usually implemented from scratch. We will call ours "struct buffer":

```
struct buffer {
  char *data, *end, *read_ptr;
  unsigned long size;
};
```

data is a pointer to the string this buffer stores, and **end** is the first byte after the string end. **read_ptr** is where **read()** should start reading the data from. The buffer size is stored for the completeness – for now, we don't use this field. You shouldn't assume the users of your structure will correctly initialise all of these, so it is better to encapsulate buffer allocation and deallocation in functions. They are usually named **buffer_alloc()** and **buffer_free()**.

```
static struct buffer *buffer_alloc(unsigned long size)
{
  struct buffer *buf;
  buf = kzalloc(sizeof(*buf), GFP_KERNEL);
  if (unlikely(!buf))
    goto out;
  ...
out:
  return buf;
}
```

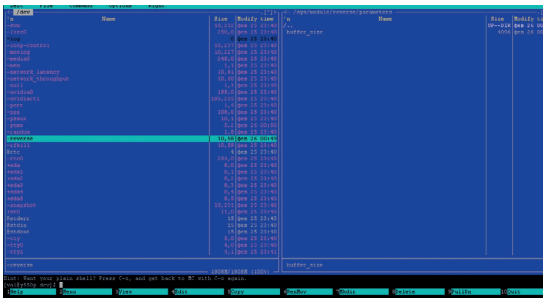
Kernel memory is allocated with **kmalloc()** and freed with **kfree()**; the **kzalloc()** flavour sets the memory to all-zeroes. Unlike standard **malloc()**, its kernel counterpart receives flags specifying the type of memory requested in the second argument. Here, **GFP_KERNEL** means we need a normal kernel memory (not in DMA or high-memory zones) and the

Avoid root if possible

By default, **/dev/reverse** is available to root only, so you'll have to run your test programs with **sudo**. To fix this, create **/lib/udev/rules.d/99-reverse.rules** file that contains:

```
SUBSYSTEM=="misc", KERNEL=="reverse", MODE="0666"
```

Don't forget to reinsert the module. Making device nodes accessible to non-root users is generally not a good idea, but it is quite useful during development. This is not to mention that running test binaries as root is not a good idea either.



Our first module is small but real: it registers a device node and has its own entry in sysfs.

When not to write a kernel module

Kernel programming is fun, but writing (and especially debugging) kernel code in a real-world project requires certain skills. In general, you should descend to the kernel level only if there is no other way to solve your problem. Chances are you can stay in the userspace if:

- You develop a USB driver – have a look at libusb (www.libusb.org).
- You develop a filesystem – try FUSE (fuse.sf.net).
- You are extending Netfilter – libnetfilter_queue may help you then (www.netfilter.org/projects/libnetfilter_queue).

Generally, native kernel code will perform better, but for many projects this performance loss isn't crucial.

function can sleep (reschedule the process) if needed. **sizeof(*buf)** is a common way to get the size of a structure accessible via pointer.

You should always check **kmalloc()**'s return value: dereferencing NULL pointer will result in kernel panic. Also note the use of **unlikely()** macro. It (and the opposite **likely()** macro) is widely used in the kernel to signify that the condition is almost always true (or false). It doesn't affect control flow, but helps modern processors to boost performance with branch prediction.

Finally, note the **gotos**. They are often considered evil, however, the Linux kernel (and some other system software) employs them to implement centralised function exiting. This results in less deeply nested and more readable code, and is much like the **try-ctach** blocks used in higher-level languages.

With **buffer_alloc()** and **buffer_free()** in place, the implementation of the **open** and **close** methods becomes pretty straightforward.

```
static int reverse_open(struct inode *inode, struct file *file)
{
  int err = 0;
  file->private_data = buffer_alloc(buffer_size);
  ...
  return err;
}
```

struct file is a standard kernel data structure that stores information about an opened file, like current file position (**file->f_pos**), flags (**file->f_flags**), or open mode (**file->f_mode**). Another field, **file->private_data** is used to associate the file with some arbitrary data. Its type is void ***, and it is opaque to the kernel outside the file's owner. We store a buffer there.

If the buffer allocation fails, we indicate this to the calling user space code by returning negative value (**-ENOMEM**). A C library doing **open(2)** system call (probably, **glibc**) will detect this and set **errno** appropriately.

Learn to read and write

"Read" and "write" methods are where the real job is done. When data is written to a buffer, we drop its previous contents and reverse the phrase in-place, without any temporary storage. The **read** method simply copies the data from the kernel buffer into the userspace. But what should the **reverse_read()** method do if there is no data in the buffer yet? In userspace, the **read()** call would block until the data is available. In the kernel, you must wait. Luckily, there is a mechanism for this, and it is called 'wait queues'.

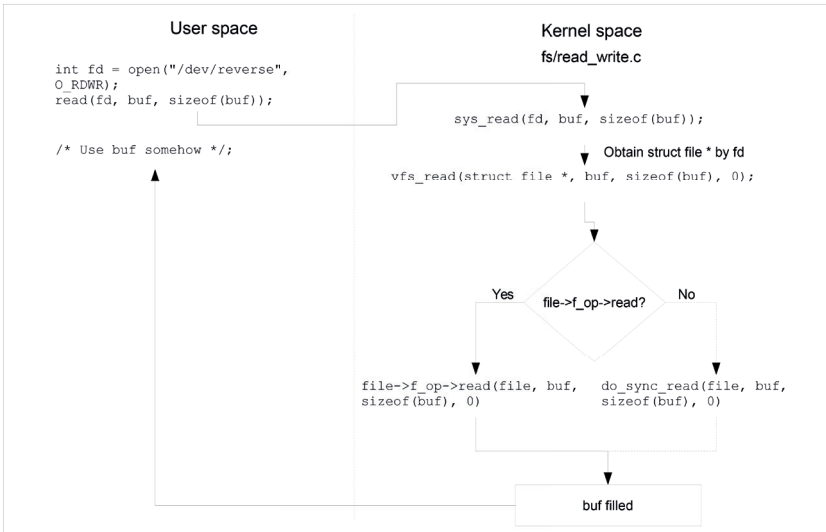
The idea is simple. If a current process needs to wait for some event, its descriptor (a **struct task_struct** stored as 'current') is put into non-runnable (sleeping) state and added to a queue. Then **schedule()** is called to select another process to run. A code that generates the event uses the queue to wake up the waiters by putting them back to the **TASK_RUNNING** state. The scheduler will select one of them somewhere in the future. Linux has several non-runnable process states, most notably **TASK_INTERRUPTIBLE** (a sleep that can be interrupted with a signal) and **TASK_KILLABLE** (a sleeping process that can be killed). All of this should be handled correctly, and wait queues do this for you.

A natural place to store our read wait queue head is struct buffer, so start with adding **wait_queue_head_t read_queue** field to it. You should also include **linux/sched.h**. A wait queue can be declared statically with **DECLARE_WAITQUEUE()** macro. In our case, dynamic initialisation is needed, so add this line to **buffer_alloc()**:

```
init_waitqueue_head(&buf->read_queue);
```

We wait for the data to be available; or for **read_ptr != end** condition to become true. We also want the wait to be interruptible (say, by Ctrl+C). So the "read" method should start like this:

```
static ssize_t reverse_read(struct file *file, char __user * out,
        size_t size, loff_t * off)
{
  struct buffer *buf = file->private_data;
  ssize_t result;
  while (buf->read_ptr == buf->end) {
    if (file->f_flags & O_NONBLOCK) {
      result = -EAGAIN;
      goto out;
    }
    if (wait_event_interruptible
      (buf->read_queue, buf->read_ptr != buf->end)) {
      result = -ERESTARTSYS;
      goto out;
    }
  }
  ...
```



Ever wondered how the **read(2)** system call reaches **reverse_read()**? This diagram explains.

We loop until the data is available and use **wait_event_interruptible()** (it's a macro, not a function, that's why the queue is passed by value) to wait if it isn't. If **wait_event_interruptible()** is, well, interrupted, it returns a non-zero value, which we translate to **-ERESTARTSYS**. This code means the system call should be restarted. **file->f_flags** check accounts for files opened in non-blocking mode: if there is no data, we return **-EAGAIN**.

We can't use **if()** instead of **while()**, since there can be many processes waiting for the data. When the **write** method awakes them, the scheduler chooses the one to run in an unpredictable way, so by the time this code is given a chance to execute, the buffer can be empty again. Now we need to copy the data from **buf->data** to the userspace. The **copy_to_user()** kernel function does just that:

```
    size = min(size, (size_t) (buf->end - buf->read_ptr));
    if (copy_to_user(out, buf->read_ptr, size)) {
        result = -EFAULT;
        goto out;
    }
```

The call can fail if the user space pointer is wrong; if this happen, we return **-EFAULT**. Remember not to trust anything coming outside the kernel!

```
    buf->read_ptr += size;
    result = size;
out:
    return result;
}
```

Simple arithmetic is needed so the data can be read in arbitrary chunks. The method returns the number of bytes read or an error code.

Write method is simpler and shorter. First, we check that the buffer have enough space, then we use the **copy_from_userspace()** function to get the data. Then **read_ptr** and **end** pointers are reset and the buffer contents are reversed:

```
    buf->end = buf->data + size;
    buf->read_ptr = buf->data;
    if (buf->end > buf->data)
        reverse_phrase(buf->data, buf->end - 1);
```

reverse_phrase() does all heavy lifting. It relies on the **reverse_word()** function, which is quite short and marked inline. This is another common optimisation; however, you shouldn't overuse it, since aggressive inlining makes the kernel image unnecessarily large.

Finally, we need to wake up processes waiting for the data at **read_queue**, as described earlier. **wake_up_interruptible()** does just that:

```
    wake_up_interruptible(&buf->read_queue);
```

Phew! You now have a kernel module that at least compiles. It's time to test it.

Surprise, surprise!

Compile the module and load it into the kernel:

```
$ make
$ sudo insmod reverse.ko buffer_size=2048
$ lsmod
reverse              2419  0
$ ls -l /dev/reverse
crw-rw-rw- 1 root root 10, 58 Feb 22 15:53 /dev/reverse
```

Everything seems to be in place. Now, to test how the module works, we'll write a small program that reverses its first command line argument. The **main()** function (sans error checking) may look like this:

```
int fd = open("/dev/reverse", O_RDWR);
write(fd, argv[1], strlen(argv[1]));
read(fd, argv[1], strlen(argv[1]));
printf("Read: %s\n", argv[1]);
```

Run it as:

```
$ ./test 'A quick brown fox jumped over the lazy dog'
Read: dog lazy the over jumped fox brown quick A
```

It works! Play with it a little: try passing single-word or single-letter phrases, empty or non-English strings (if you have a keyboard layout set) and anything else.

Now let's make things a little trickier. We'll create two processes that share the file descriptor (and hence the kernel buffer). One will continuously write strings to the device, and another will read them. The **fork(2)** system call is used in the example below, but **pthreads** will work as well. I also omitted the code that opens and closes the device and does the error checking (again):

```
char *phrase = "A quick brown fox jumped over the lazy dog",
if (fork())
    /* Parent is the writer */
    while (1)
        write(fd, phrase, len);
else
    /* child is the reader */
    while (1) {
        read(fd, buf, len);
        printf("Read: %s\n", buf);
    }
```

What you expect this program to output? Below is what I've got on my laptop:

```
Read: dog lazy the over jumped fox brown quick A
Read: A kcicq brown fox jumped over the lazy dog
Read: A kciuq nworb xor jumped fox brown quick A
Read: A kciuq nworb xor jumped fox brown quick A
...
```

What's going on here? It's a race. We thought **read** and **write** were atomic, or executed one instruction at a time from the beginning till the end. However the kernel is a concurrent beast, and it can easily reschedule the process running the kernel-mode part of the **write** operation somewhere inside **reverse_phrase()** function. If the process that does **read()** is scheduled before the writer is given a chance to finish, it will see the data in an inconsistent state. Such bugs are really hard to debug. But how to fix it?

Basically, we need to ensure that no **read** method can be executed until the **write** method returns. If you ever programmed a multi-threaded application, you've probably seen synchronisation primitives (locks) like mutexes or semaphores. Linux has them as well, but there are nuances. Kernel code can run in the process context (working "on behalf" of the userspace code, as our methods do) and in the interrupt context (for example, in an IRQ handler). If you are in the process context and a lock you need has already been taken, you simply sleep and retry until you succeed. You can't sleep in the interrupt context, so the code spins in a loop until the lock become available. The corresponding primitive is called a spinlock, but in our case, a simple mutex – an object that only one process can "hold" at the given time – is sufficient. A real-world code may also use a read-write semaphore, for performance reasons.

Locks always protect some data (in our case, a "struct buffer" instance), and it is very common to embed them in a structure they are protecting. So we add a mutex ('struct mutex lock') into the "struct buffer". We must also initialise the mutex with **mutex_init()**; **buffer_alloc()** is a good place for this. The code that uses mutexes must also include **linux/mutex.h**.

A mutex is much like a traffic light – it's useless unless drivers look at it and follow the signals. So we need to update **reverse_read()** and **reverse_read()** to acquire the mutex before doing anything to the buffer and release it when they are done. Let's have a look at **read** method – **write** works just the same way:

```
static ssize_t reverse_read(struct file *file, char __user * out,
        size_t size, loff_t * off)
{
  struct buffer *buf = file->private_data;
  ssize_t result;
  if (mutex_lock_interruptible(&buf->lock)) {
    result = -ERESTARTSYS;
    goto out;
  }
```

We acquire the lock at the very beginning of the function. **mutex_lock_interruptible()** either grabs the mutex and returns or puts the process to sleep until the mutex is available. As before, **_interruptible** suffix means the sleep can be interrupted with a signal.

```
  while (buf->read_ptr == buf->end) {
    mutex_unlock(&buf->lock);
    /* ... wait_event_interruptible()  here ... */
    if (mutex_lock_interruptible(&buf->lock)) {
      result = -ERESTARTSYS;
      goto out;
    }
  }
```

Below is our "wait for the data" loop. You should never sleep when holding a mutex, or a situation called a "deadlock" may occur. So, if there is no data, we release the mutex and call **wait_event_interruptible()**. When it returns, we reacquire the mutex and continue as usual:

```
  if (copy_to_user(out, buf->read_ptr, size)) {
    result = -EFAULT;
    goto out_unlock;
  }
  ...
out_unlock:
  mutex_unlock(&buf->lock);
out:
  return result;
```

Finally, the mutex is unlocked when the function ends or if an error occurs while the mutex is being held. Recompile the module (don't forget to reload it) and run the second test again. You should see no corrupted data now.

What's next?

Now you have a taste of kernel hacking. We've just scratched the surface of the topic, and there is much more to see. Our first module was intentionally simple, however the concepts you learned will stay the same in more complex scenarios as well. Concurrency, method tables, registering callbacks, putting processes to sleep and waking them up are things that every kernel hacker should be comfortable with, and now you've seen all of them in action. Maybe your kernel code will end up in the mainline Linux source tree some day – drop us a line if this happens!

Dr Valentine Sinitsyn edited the Russian edition of O'Reilly's *Understanding the Linux Kernel*, has a PhD in physics, and is currently doing clever things with Python.

Debugging kernel code

Perhaps the most common debugging method in the kernel is printing. You can use plain **printk()** (presumably with **KERN_DEBUG** log level) if you wish. However, there are better ways. Use **pr_debug()** or **dev_dbg()**, if you are writing a device driver that has its own "struct device": they support the dynamic debug (**dyndbg**) feature and can be enabled or disabled on request (see **Documentation/dynamic-debug-howto.txt**). For pure development messages, use **pr_devel()**, which becomes a no-op unless DEBUG is defined. To enable DEBUG for our module, include:

```
CFLAGS_reverse.o := -DDEBUG
```

in the Makefile. After that, use **dmesg** to view debug messages generated by **pr_debug()** or **pr_devel()**.

Alternatively, you can send debug messages directly to the console. To do this, either set **console_loglevel** kernel variable to 8 or greater (**echo 8 > /proc/sys/kernel/printk**) or temporarily print the debug message in question at the high log level like **KERN_ERR**. Naturally, you should remove debug statements of this kind before publishing your code.

Note that kernel messages appear on the console, not in a terminal emulator window such as Xterm; that's why you'll find recommendations not to do kernel development in the X environment.

LV OVER TO YOU

Do you want to see some more kernel magic – perhaps intercepting file access or creating your own iptables module? Tell us what you think at forums.linuxvoice.com or letters@linuxvoice.com

LINUX VOICE

MASTERCLASS

Essential Linux tools explained – this month, say hello to the Filezilla FTP client and learn the power of SSH.

GET TO KNOW FILEZILLA

Does you email have a draconian attachment size limit? Never mind – learn Filezilla and get your files moving.

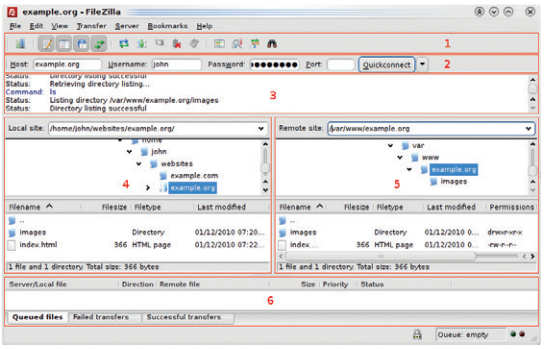
JOHN LANE

Moving files between machines is something that most of us need to do at some point. Whether that's simply as a backup, updating a web server or another task, it helps to have a flexible tool that can handle multiple file transfer protocols. Today, we'll take a look at FileZilla, a cross-platform GUI application released under the GPL version 2. It's implemented using wxWidgets, a cross-platform GUI library that uses GTK on Linux, so it doesn't have many dependencies. You should find it in your distribution's package repository:

```
sudo apt-get install filezilla
```

The first time that you start FileZilla you should take a minute to familiarise yourself with the layout, but the main thing to grasp is that you transfer files between a local directory, displayed on the left, and a remote one, which is displayed on the right.

The main window comprises the toolbar and quick connect bar, a message log, and local and remote file listings. The transfer queue lists the files that have, or will be, transferred, and the toolbar contains buttons that you can use to show and hide the individual areas of the main window.



FileZilla's window has six main areas, but basically your local files are on the left, with romote files on the right.

You can connect to a remote site immediately by filling in the boxes in the Quickconnect Bar, which lies across the top of the window. The default protocol is FTP, but you can also use SFTP or FTPS by prefixing the host name with the protocol.

```
sftp://mysite.example.org
```

You should enter the username and password for the site. See the boxout if you are using SFTP and need to authenticate with a private key instead of a password (leave the password field empty in this case).

Secure FTP

FileZilla also checks and warns about unmatched SFTP server host keys in a similar way to SSH (it internally uses PuTTY code and, if you use that too, then they share the same cache file (which can be found at **~.putty/sshhostkeys**).

Once you are connected to a remote server, its files and directories are displayed in the remote site part of the window. The local site window similarly displays files and directories on your local machine. Navigating around both of these areas is done in the same way, and to copy a file you drag and drop between them. When you do so, the transfer queue updates to reflect

SFTP authentication

Many SSH servers require that you have a key for authentication. If you are connecting to such a server, Filezilla gives you two choices. The first, and probably best, is to add the required key to your ssh-agent and ensure that the SSH_AUTH_SOCK environment variable is set before launching Filezilla.

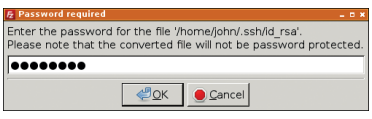
To start an SSH agent and add your key use this command:

```
eval `ssh-agent` && ssh-add
```

The second way is to add your keys into FileZilla. To do this, go to Settings > SFTP to add them. FileZilla natively works with keys saved in PPK format, which is the PuTTY Private Key format (PuTTY is an SSH client that works on Windows as well as Linux).

If, as is most likely, you are using an OpenSSH key, FileZilla will prompt to convert its format. It will save a copy of your key in a new PPK-formatted file of your choosing.

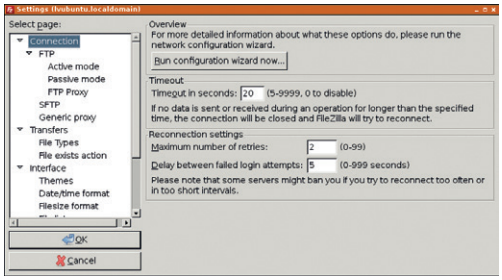
One important point to note is that FileZilla does not work with keys that require a passphrase. If you try to add such a key, you will be prompted to convert it. This will require you to enter your passphrase and will save a copy of your key with its passphrase removed.



If you feel uncomfortable about the security implications of using unprotected private keys, you may prefer not to load them into Filezilla and, instead, use the SSH Agent.

Settings

FileZilla has a lot of options that you can tweak to your taste. The settings dialog, accessed via the Edit menu, enables you to control many aspects of the application, including how connections are made and how the various protocols work. You can also control how FileZilla performs file transfers and customise the user interface.



When you've tweaked it exactly how you want it, your config settings are saved to ~/.filezilla/filezilla.xml.

the transfers that take place. This is separated into three tabs: Queued Files shows transfers yet to take place; Failed Transfers and Successful Transfers are self-explanatory.

The Local and Remote windows each have a pop-up context menu (right-click) that you can use to create and delete files and directories, set permissions and so-on. The options vary depending whether the menu pops up in the context of a file, directory or neither.

Quick connections

FileZilla remembers connections that you make with the Quickconnect bar, as we have done here, and they can be repeated by selecting them from the drop-down beside the Quickconnect button. The Quickconnect bar allows you to build up a history of connections that you can re-use quickly and easily, but there is another way to manage connections if you want more control – the Site Manager.

The Site Manager allows you to organise the sites that you connect to and offers more options to control how those connections operate. It is, however, distinct from the Quickconnect bar, and sites added to one do not appear in the other. You can add a currently connected site to the Site Manager by using File > Copy Current Connection To Site Manager. This opens the Site Manager ready to accept the new site and defaults its fields with the details for the current connection. Press OK to complete adding the site.

There are two ways that you can connect to a site that is saved in the Site Manager. The first way is to open the Site Manager, select the site and click Connect. The other, more direct route, is to right-click the Site Manager toolbar icon and select the site from the drop-down menu that is displayed.

As you use FileZilla to connect to more sites, each new connection can either replace an existing one, which gets closed, or can be opened as a new tab. A tabbed interface is added to the main screen when you have two or more sites open, and each tab contains local remote and transfer windows for one connection.

Instead of connecting to a site, you can perform a manual transfer of a single file. This function, accessed through Transfer > Manual Transfer is a way to move one file to or from a remote site, although connecting and using drag and drop is a more intuitive way to achieve the same result.

LV PRO TIP

For documentation and support resources, go to **http://filezilla-project.org**

Memorable paths

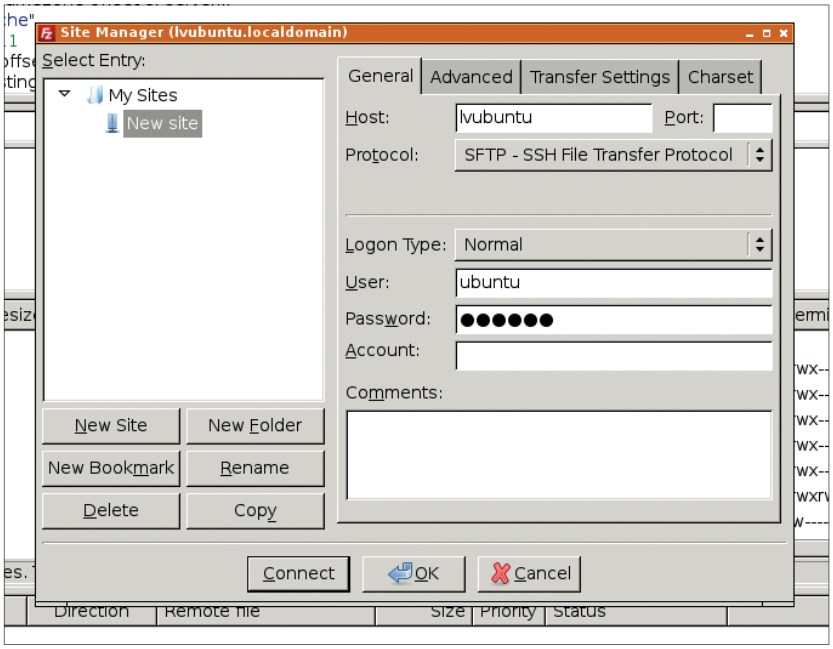
If you find yourself regularly navigating to many local and remote paths, you'll want to check out the Bookmarks function. A bookmark defines either local or a remote path and can be global or site-specific. When a bookmark is selected, the Local and Remote windows change to those paths.

A useful feature that can be enabled on a bookmark or on a site as a whole (through the Site Manager) is synchronised browsing. This selects a local and remote directory path and keeps both in sync as you navigate either the local or remote file views. You'll be warned if you try to navigate to a path on one that doesn't exist on the other; continuing to that location disables synchronised browsing.

"Firefox was one of the first browsers to offer the tabbed browsing model."

It's just as important to realise what synchronised browsing isn't: it is not an automated mechanism to synchronise the contents of local and remote filesystems. Filezilla doesn't do that. If you want automated synchronisation there are other more appropriate tools available, not least rsync. But, for a nice clean GUI-based general-purpose file transfer utility, FileZilla has it covered.

Use the Site Manager to organise the sites that you connect to.



AN INTRODUCTION TO SSH

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

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

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

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

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

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

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

```
ssh remotehost
```

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

```
ssh user@remotehost
```

You will see a serious-looking warning when you connect to a remote host for the first time, but assuming you're happy that what you've connected to is what you asked for, you can enter **yes** to continue the connection.

```
The authenticity of host 'remotehost (192.168.1.15)' can't be established.
ECDSA key fingerprint is 6d:c4:cf:43:75:a5:79:e0:74:a0:b7:22:b7:da:e1:25.
Are you sure you want to continue connecting (yes/no)?
```

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

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

The keymaster

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

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

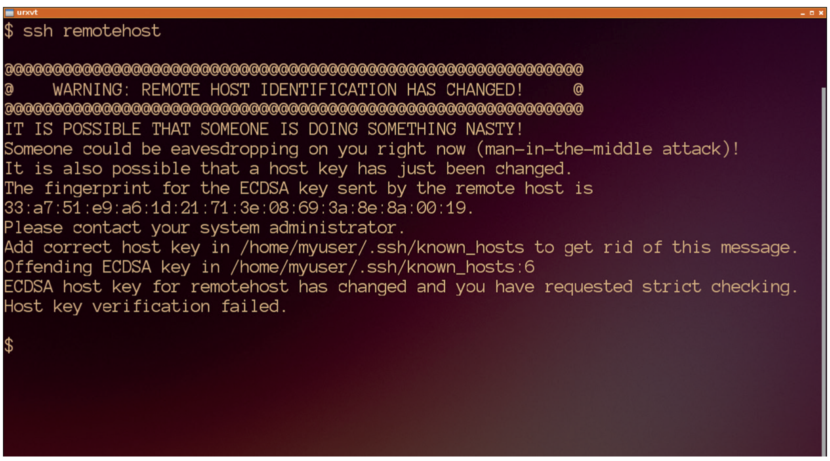
```
ssh-keygen -t rsa
```

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

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

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

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



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

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

Beyond the command-line

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

```
ssh -X remotehost firefox
```

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

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

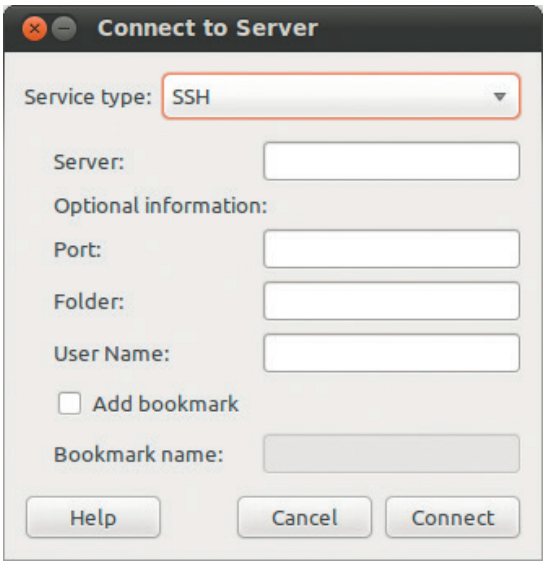
The parameters run SSH as a daemon (**-f**) without executing a remote command (**-N**) because we're tunnelling instead. The **-L 25:remotehost:25** tells SSH to listen on a local port 25 and forward any traffic it receives across the SSH connection to **remotehost** and onwards to **remotemailhost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

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

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

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

```
sftp remotehost
```



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

Server configuration

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

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

After changing the server configuration, reload it with

```
sudo killall -HUP sshd
```

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

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

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

```
sudo apt-get install sshfs
sudo adduser myuser fuse
```

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

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

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

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

Securing SSH

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

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

```
command="/usr/bin/something args..." ssh-rsa AAAAB3Nza-C1yc...
```

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

LV PRO TIP

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

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

LINUXVOICE DVD 002

Distros, videos, applications, games, podcasts and more...

SHOWCASING THE BEST OF FOSS

Welcome to the second Linux Voice DVD. It's been fantastic to hear your feeback about the first issue, so thank you to everyone who got in touch. This issue's DVD, like last month's, is an 8GB dual-layer beast jam-packed with software, videos, podcasts and much more.

We have the latest release of Mageia, a superb desktop distribution that boasts a whopping nine desktop environments and window managers. It's a great way to explore the different interfaces in the Linux and open source world.

Then we have gNewSense, a completely pure distribution (ie without a single byte of non-free software) that even the notoriously picky Richard Stallman doesn't have a beef with.

And then there are videos, apps and other gems to explore – open index.html for the full lowdown. We hope you enjoy the disc, and if you have a digital subscription to Linux Voice, you can get the DVD ISO image from our website at www.linuxvoice.com.

Mike Saunders, Disc Editor
mike@linuxvoice.com

Complete desktop distro

Mageia 4

A first-class Mandriva spin-off with over 4,400 packages.

One of the biggest reasons for distro hopping is trying out a new desktop environment. Sure, most big-name distros have a range of desktops, but they're often not integrated very well, so you have to choose a specific distro to get a good version of, say, KDE or Gnome.

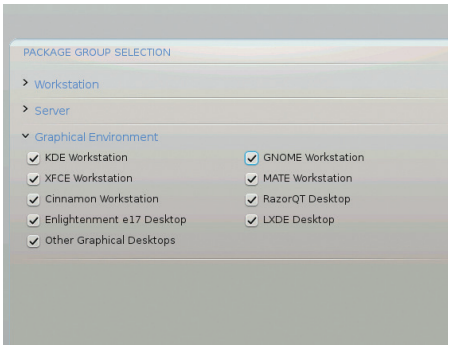
Mageia is different: the version on our DVD includes nine desktops and window managers, all polished and integrated excellently. So you can try the latest versions of KDE, Gnome, Cinnamon, MATE, Xfce, Enlightenment and more from a single distro. We've included the complete 3.7GB edition of Megeia 4 on the disc, and whether you're a long-time Linux user or are new to the operating system, it's well worth a try. The minimum system requirements are:

- **32 or 64-bit x86 CPU**
- **512MB RAM**
- **6GB hard drive space**

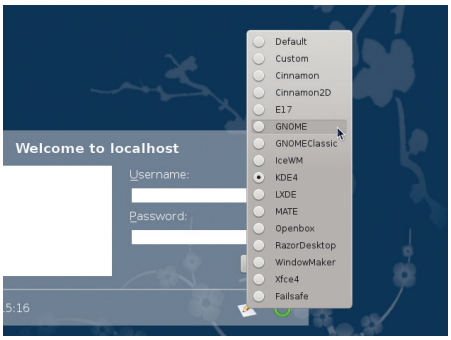
Still, we recommend having 1GB+ RAM for smooth running. To install Mageia, just boot your PC from the Linux Voice DVD – ie start your PC with the disc in the drive. (On some older PCs, you may need to change the boot order in your BIOS so that the DVD drive boots first, so consult your PC documentation or your local Google.)

Choose 'Install Mageia 4' from the boot menu and follow the prompts. If you're completely new to Linux, before installing it's worth loading up **index.html** from the disc in Firefox, scrolling down to the Videos section, and watching our videos about installing and using Mageia – they'll get you prepared before you do the installation yourself.

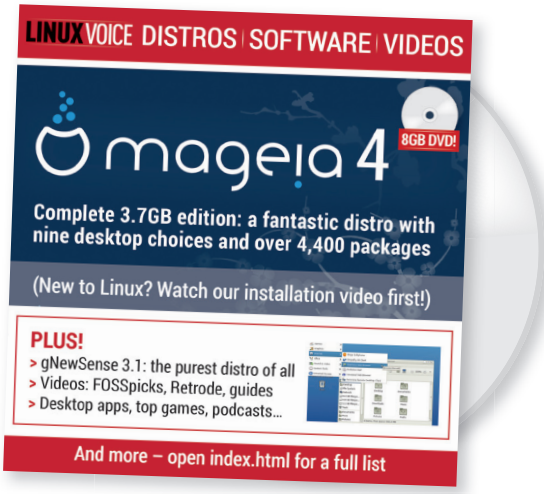
If you want the complete nine-desktop experience, at the Desktop Selection phase of the installation, choose the Custom option. You can then pick and choose which desktops and WMs you want installed. And for more information on Mageia, visit its website at **www.mageia.org**.



Choose Custom during the Desktop Selection step of the installation for a big range of options.



After installation, click the pencil and paper icon in the login screen to choose your desktop or window manager.



RMS-friendly distro

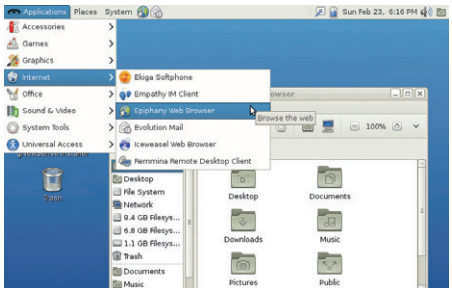
gNewSense 3.1

Only free (as in speech) software – a very pure distro.

We're not big fans of the name, but we really like gNewSense. It's a GNU/Linux distro that only contains completely free software: that is, no binary-only drivers or patent-iffy media codecs that many other desktop distros include. Richard Stallman, creator of GNU and the awesome GPL licence, thinks that most GNU/Linux distributions don't help to advance the cause of software freedom by installing those extras – so he likes gNewSense for its purity.

Now, gNewSense isn't as up-to-date as many other desktop distros doing the rounds, but the new 3.1 release does include various small changes and fixes. We've included this release (the 64-bit version) on the Linux Voice DVD: to install it, just boot your PC from the disc and choose one of the gNewSense options from the boot menu. As well as booting the distro in live mode (which runs from the DVD and has an installer icon on the desktop) you can also perform a text-mode installation, which is ideal if you want to install on older machines or servers.

gNewSense is based on Debian, so any skills you've learnt during your time with Debian and/or Ubuntu will come in useful here. It uses **apt** for package management and all the usual tools are present. There's nothing especially fancy about it – it's just a solid all-round distro with a firm focus on freedom. If you love the spirit and goals of free software, give it a go. And if you need any help or support, pop over to the project's website: **www.gnewsense.org**.



If you hated earlier versions of Gnome 3, it's worth giving the desktop another chance.

Not just software!

Linux Voice videos

Mageia, FOSSpicks, Retrode and newbie guides.

When we started work on the first Linux Voice DVD, we wanted to make it much more than just a compendium of the latest Linux software. We wanted it to be a part of the magazine, website and community as a whole, so we decided to make some videos showcasing things covered in the mag. And we've continued that for this month's DVD.

Open **index.html** in Firefox and scroll down to the Videos section. Click the links to watch the videos – they're in Ogg Theora and Vorbis format, so they will play straight away in Firefox without the need for any extra apps or media codecs.

Firstly there are the aforementioned videos explaining how to install Mageia and explore its KDE desktop. Then we have a look at some apps covered in FOSSpicks, including the rather cool Android-x86. Next there's a quick look at the Retrode as covered on page 22. Finally, Graham made some great "new to Linux" videos explaining what Linux is, how to install it and how to use it, based on Linux Mint 16 from last issue's DVD. They're essential viewing if you're taking your first steps in Linux.

Thanks to the completely free and open Theora+Vorbis video codec, you can watch our videos straight away in Firefox.

And there's more!

Podcasts, software, games, Linux From Scratch...

Legends tell of a faraway group of people, isolated from the rest of the world, with barely any contact with mankind. Little is known about these beings, but one thing is for sure: they're missing out on the most important development in human history. They've never heard the Linux Voice Podcast.

Maybe this issue's DVD will reach their settlements at some point, so we've included the latest episodes for their listening enjoyment. There's lots of Linux banter, lots of discoveries, and lots of off-topicness.

Also on the disc are a bunch of new desktop applications including LibreOffice 4.2 (in Deb and RPM formats), PCManFM 1.2 and Anjuta 3.11.4. We have a selection of hot games such as Ember 0.7.1, along with the latest kernel and GCC code for those who want to live on the bleeding edge. Oh, and if you have a few hours spare and want to get your hands dirty with a new project, try Linux From Scratch: it teaches you how to make your own custom Linux distribution, compiling every little piece yourself. It's ace.

Join the online RPG WorldForge using Ember, and buy pigs from strangers.

/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!

Money is great. You can do all sorts of things with it. It really is rather amazing, and we should all pay due homage to Pheidon, who struck the first recognised coins (in Argos, or possibly Lydia) in about 700BC, since which time it has certainly been a much easier trip to the supermarket.

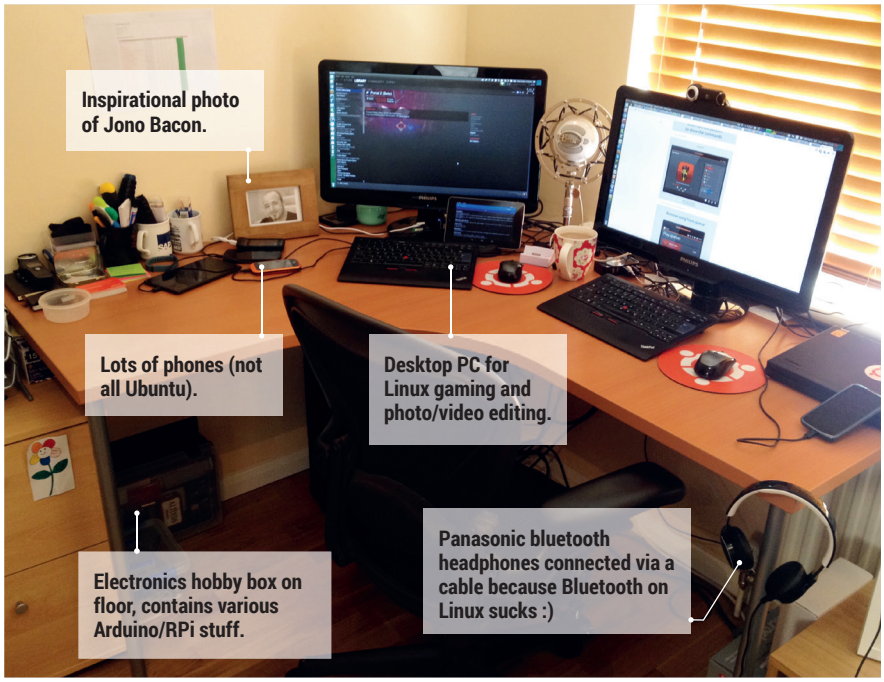
It is therefore a good thing that Linux Voice is giving away 50% of its profits to the needy of open source. Loads of projects, big and small, useful and silly, will no doubt benefit. But, in a terrible paraphrasing of Bob Geldof, "it's not enough". Not that I think 50% isn't enough – I mean money isn't enough.

Many a time, when I have been confronted by a pernicious bug, or some rogue documentation or an unintelligible user interface, I have taken a crisp £20 out of my wallet and waved it at the screen saying "here, for the love of everything that doesn't suck, fix this". Sometimes I have got up to £100, but the screen stares impassively back, taunting my impotence.

As I have already mentioned, money is incredibly useful. Projects can use it for much-needed hardware, or maybe for inescapable ISP costs, attending events or securing an adequate supply of caffeine...

But. What projects often need, sometimes even more than Bitcoins, is vitality, effort and responsiveness. This is usually achieved by having a vibrant, involved community. There have been dozens of articles online and elsewhere on different ways you can help open source software, even if you can't write code.

Paying your way in open source is a good thing, but getting involved is better. If you like software, support it. Get involved. Sometimes little more is required to keep a software project ticking along than just people turning up.



My Linux setup Alan Pope

@uupc podcaster and Canonical chap shows us the office.

Q What version of Linux are you using at the moment?

A Ubuntu Trusty (which will become 14.04) on laptop and desktop. Also, Ubuntu on the Nexus 4's and Nexus 7's. One 2012 Nexus 7 running Android constantly running **irssi-notifier** to alert me of IRC pings.

Q What desktop do you use? If we had to guess, we' say...

A ... Unity on the laptop, desktop and phone.

Q What was the first Linux setup you ever used?

A Red Hat, which I got with a book in around 1996. I had no clue what I was doing, but the terminal seemed cool! Next stop Debian, then Ubuntu where I've stayed for nine years.

Q What Free Software/open source can't you live without?

A Linux itself. While I use the entire stack every day, without the work done by the Linux kernel developers, I can't imagine most of the rest of it would be as advanced and usable as it is.

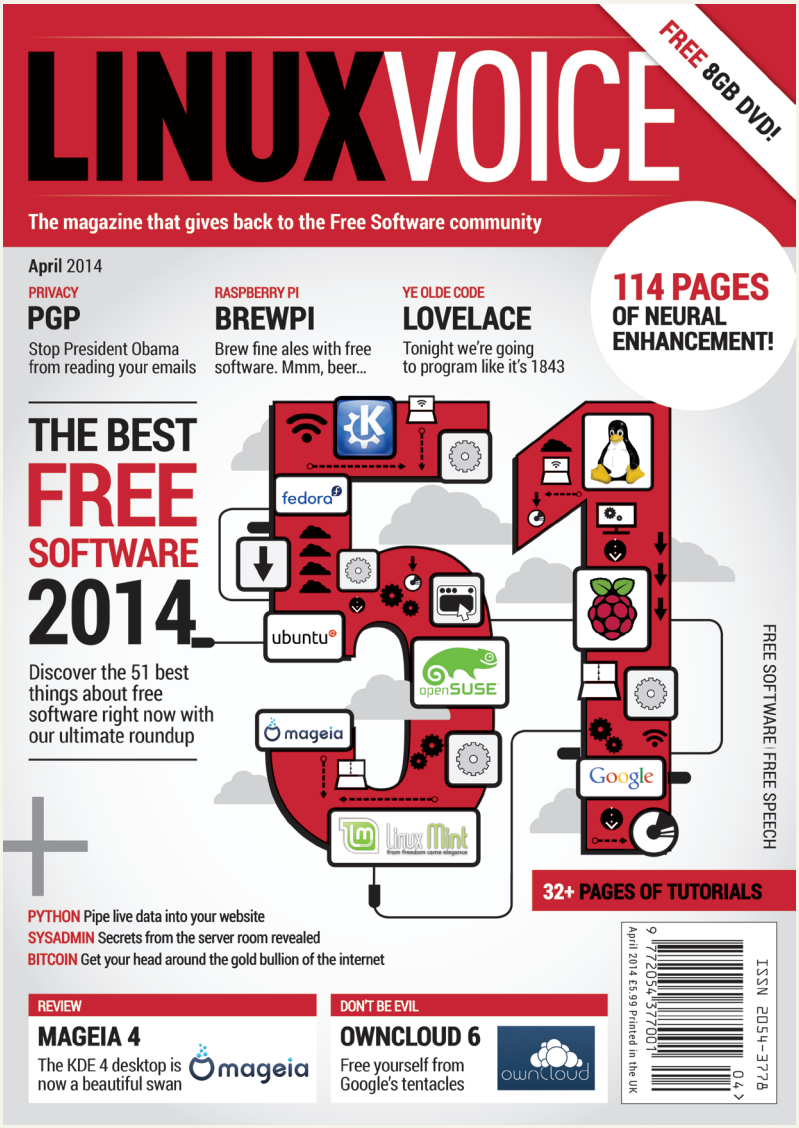
Q What do other people love but you can't get on with?

A KDE. I promised to try it for a full 6 months. Got intensely frustrated with it after a month and gave up, never looked back.

Follow us on Twitter @linuxvoice!

THANK YOU!

Linux Voice would like to say a huge thanks to everyone who made this magazine possible. If you bought the Founder perk during our IndieGogo campaign, and we missed you off the Founders' page in issue 1, we're sorry. Many, many thanks.



Anthony Beavis, Auroraskyes, David Banks, Den Weatherall, Kevan Vautier, Luis Joaquim, Marius Orintas, Mike Eaket, Patrick Allen, Peter Selley, Sally Henderson, Samuel Voss, Sebastian Dixon-Lewis, Stephan Diestelhorst, Tarquin Adams


LV