FREE EXCLUSIVE PROJECT CODE, VIDEOS & SOFTWARE

www.linuxuser.co.uk

Linux User & Developer™

THE ESSENTIAL MAGAZINE FOR THE GNU GENERATION

"OTTO IS THE SUCCESSOR TO VAGRANT"

Mitchell Hashimoto
Vagrant Creator

PLASMA 5 TIPS & TRICKS

Master KRunner, custom keybindings and more

DEVOPS

CREATE SOFTWARE FASTER

Deploy and orchestrate containers like a pro

WIN £750 OF RAS PI PRIZES

COMPILE YOUR OWN SOFTWARE

Get to grips with the GNU Compiler Collection

CONTAINERS ON COREOS

Quickly deploy your clustered environments

MAKE GAMES WITH PYGAME

Code a classic *Breakout* game with the Pygame Zero framework

PLUS Backup solutions ❁ Hack a robot with IR ❁ Fitlet mini PC ❁ MonoDevelop GUIs

ISSUE 159 £5.99

IMAGINE PUBLISHING

9 772041 327002 59 >

Synology®

DS115j AND DS215j

THE PERFECT DATA STORAGE SOLUTION FOR YOUR HOME

Run your own personal cloud from the comfort and security of your home

POWER-SAVING AND RELIABLE

DS215j features a dual-core CPU, supports up to 12TB of data and consumes less than 14w when active while the DS115j offers a stable storage environment for users with no need for raid, consuming less than 11w when active.

ACCESS ANYWHERE, ANYTIME

Synology NAS' and the intuitive DiskStation Manager (DSM) software allow users to sync and share files among multiple devices, including Windows PC, Mac, Linux, iOS, Android and Windows Phone.

MULTIMEDIA CENTRE

Audio Station, Photo Station, Video Station and Media Server transform your Synology NAS into a centralised multimedia hub.

Where to Buy

amazon.co.uk

The electronics specialist maplin

Synology apps available on

iOS

Synology's Media Accolades

PC PRO Excellence AWARDS 2014 SYNOLOGY BEST NAS DEVICE BRAND

Welcome

to issue 159 of Linux User & Developer

Your team of Linux experts...

Jon Masters is a Linux kernel hacker who has been working on Linux for some 19 years, since he first attended university at the age of 13. Jon lives in Cambridge, Massachusetts, and works for a large enterprise Linux vendor. You can find his brilliant Kernel Column on pages 16-17 this month.

Richard Smedley has been helping businesses and community organisations of all sizes to move to GNU/Linux since the 1990s and doesn't expect those he works with to share his love of Emacs. Well, maybe a little. This issue, Richard shows us how to code games with Pygame Zero (p.62) and shares more great FOSS (p.88).

Nitish Tiwari is a software developer by profession and an open source enthusiast by heart, and he helps firms set up and use open source software. Nitish takes on DevOps this month, walking us through Docker, Puppet and Vagrant (p.18), and also explaining how to run container clusters on CoreOS (p.38).

Mihalis Tsoukalos is a UNIX system administrator with expertise in programming, databases and maths. He has been using Linux since 1993. This month, Mihalis continues his two tutorial series: systems programming (p.28) and computer science (p.50). His Coding Column can be found on page 11.

Alexander Tolstoy has used Linux for years. Though brevity is the soul of wit, Alexander loves writing on desktop Linux in details. He has been doing it since he installed his first distro from a floppy disk. Alexander shows us how to get the perfect Plasma 5 desktop this month (p.42) and tests the best backup solutions (p.81).

Gareth Halfacree is our resident news reporter and brings us the latest developments from all over the open source world, starting over on page 6. This issue, Gareth reviews CompuLab's Fitlet iA10, a new model in its family of miniature, fanless PCs that can be customised and ship out with Linux Mint (p.86).

This issue

- Start doing DevOps today
- Make games in Pygame Zero
- Find the best backup solution
- Win a FUZE workstation plus robot arm

Welcome to the latest issue of Linux User & Developer, the UK and America's favourite Linux and open source magazine.

Embracing DevOps practices to speed up your software delivery is far easier than you think. It's a buzzy term and the definitions can be hazy, but the benefits of this modern, agile way of working – improving cohesion between the development and operations teams – can't be ignored. This month, we explain just what DevOps means to you and give you a running start with three core tools: Docker, Puppet and Vagrant. Plus, we show you around the Docker-based distro CoreOS, which makes managing a fleet of containers simple, and we speak to Vagrant creator Mitchell Hashimoto to find out more about HashiCorp's newest automation software, which is set to replace Vagrant once it matures: Otto.

The systems programming and computer science series both continue this month, and there are more great tutorials including a detailed GCC guide and some key tips for getting the perfect Plasma 5 desktop.

Over in the Pi section, it's all about the games this month. As well as the new Pygame Zero library, we're also playing with FUZE BASIC again – plus, you can enter our competition to win a FUZE workstation of your own (page 78). Enjoy the issue!

Gavin Thomas, Editor

Get in touch with the team:

linuxuser@imagine-publishing.co.uk

Visit us online for more news, opinion, tutorials and reviews:

www.linuxuser.co.uk

LinuxUser & Developer

Contents

OpenSource

Features

Tutorials

FileSilo.co.uk

Reviews


luckyBackup


Back In Time


Déjà Dup


KBackup

Join us online for more Linux news, opinion and reviews www.**linuxuser**.co.uk

DOMAINS | MAIL | HOSTING | eSHOPS | **SERVERS**

1&1 CLOUD SERVER

TEST THE BEST!

TOP-PERFORMER CLOUD SPECTATOR

Powered by intel Cloud Technology

Easy to use – ready to go

The 1&1 Cloud Server offers unbeatable performance in terms of CPUs, RAM and SSD storage! Implement your cloud projects with the perfect combination of flexibility and powerful features.

- ✓ **Load balancing**
- ✓ **SSD storage**
- ✓ **Billing by the minute**
- ✓ **Intel® Xeon® Processor E5-2660 v2 and E5-2683 v3**

1 month free!

Then from £4.99 per month*

1 TRIAL TRY FOR 30 DAYS

1 CLICK UPGRADE OR DOWNGRADE

1 CALL SPEAK TO AN EXPERT

0333 336 5509

1&1

1and1.co.uk

* 1&1 Cloud Server 1 month free trial, then from £4.99 per month. No minimum contract period. Prices exclude 20% VAT. Visit 1and1.co.uk for full offer details, terms and conditions. Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and/or other countries. 1&1 Internet Limited, Discovery House, 154 Southgate Street, Gloucester, GL1 2EX.

CONFERENCE

Developers to meet face-to-face at new UbuCon Summit

Above From 2016, Ubuntu devs will be able to meet in person once again

New conference fills hole left by cessation of physical summits

Canonical has announced that it is to hold a new physical event, the UbuCon Summit, in January 2016 following complaints of a lack of face-to-face gatherings; a result of the shift of Ubuntu Developer Summits from physical meet-ups to virtual conferences.

The UbuCon Summit, the company has explained, builds on the successes of volunteer-organised local UbuCon events and scales it up, using the existing team behind the UbuCon SCALE event in Pasadena, California as a base.

"In discussions with the Community Council, and after the participation of some Ubuntu team members at the Community Leadership Summit a few months ago, one of the challenges that we identified our community is facing is the lack of a global event to meet face-to-face after the UDS era," explained Canonical's David Planella. "While UbuCons continue to thrive as regional conferences, one of the conclusions we reached was that we needed a way to bring everyone together on a bigger setting to complement the UbuCon fabric: the Summit."

The first UbuCon Summit will be a two-day event, structured as both an organised conference and an ad-hoc unconference running side-by-side. Two tracks will be available to the attendees: Users, which will concentrate on the day-to-day uses of Ubuntu and how to act as an advocate for uptake of the popular Linux distribution; and Developers, which will offer technical sessions including app development, Internet of Things (IoT) and cloud topics, in addition to the new convergence functionality that Canonical is aiming to build within its Ubuntu mobile ecosystem.

> "The UbuCon Summit builds on the success of volunteer-organised local UbuCon events and scales it up"

The UbuCon Summit will mark the first large-scale official gathering of Ubuntu developers and users since the Ubuntu Developer Summits switched from twice-yearly physical events to virtual events hosted on Google's Hangouts service in 2013. To mark the occasion, Canonical founder Mark Shuttleworth has been confirmed as keynote speaker, with more details on the schedule promised in the coming weeks.

"The Summit is the expansion of the traditional UbuCon: more content and at a bigger scale," claimed Planella. "But at the same time maintaining the grass-roots spirit and the community-driven organisation that has made these events successful. All in all, the idea is to provide a space to showcase, learn about and discuss the latest Ubuntu technologies, but also to focus on new and vibrant parts of the community and talk about the challenges (and opportunities!) we are facing as a project."

The UbuCon Summit is to be held at the Pasadena Convention Centre on 21 and 22 January 2016 as part of the Southern California Linux Expo (SCALE). Organisers include Richard Gaskin and Nathan Haines of the Ubuntu California LoCo group, who have been heavily involved in previous SCALE-hosted UbuCon events, and Gareth Greenaway and Ilan Rabinovitch of SCALE itself.

More information on the event is available on the official UbuCon website at **ubucon.org**, or via SCALE at **socallinuxexpo.org**.

AUTOMATION

Red Hat acquires Ansible technology

Boosts hybrid cloud deployment and management

Red Hat has announced the acquisition of Ansible, best known for its agentless automation products, with a view to making the company's Red Hat Linux platform the go-to offering for hybrid cloud use.

While the financials behind the acquisition have not been formally disclosed, industry sources have suggested the deal values the company at around $100 million, with Red Hat stating that it expects its operating expenses for the next financial year to increase by around $6 million as a result of its new subsidiary.

"Our customers already use Red Hat solutions in conjunction with various IT automation tools. With this acquisition, we want to offer that type of integration through the world-class Red Hat support and certification that makes open source consumable for the enterprise, exactly the same way we do for OpenStack and every other product in our portfolio," claimed Red Hat's Alessandro Perilli, general manager of cloud management strategy, of the deal.

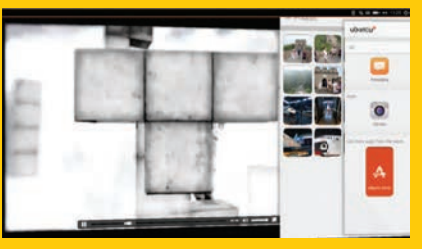
"We're thrilled that Red Hat, a global leader in open source, has chosen Ansible to tackle the future of IT automation and systems management," co-founder and chief executive Saïd Ziouani said of the acquisition of Ansible, founded in 2013, by the company for which he spent ten years working. "This is a strong validation that Ansible's simplicity, enterprise customer base and robust community is winning in enterprise IT automation, from computer to networking to cloud to containers."

"Ansible is a clear leader in IT automation and DevOps, and helps Red Hat take a significant step forward in our goal of creating frictionless IT," added Joe Fitzgerald, vice president of Red Hat, of the acquisition. "Red Hat is transforming IT management, driving innovation that is 100% open source, built on an open management platform, and relentlessly focused on reducing cost and complexity through ease of use and automation. I am thrilled to welcome Ansible to Red Hat to help us expand that commitment."

Since its founding as AnsibleWorks in 2013, Ansible's software has grown to 1,200 contributors, been heralded as the most popular open source automation tool on GitHub, and has seen a claimed 20% of the Fortune 100 working with the company.

TOP FIVE

Unity 8 features – tech preview



Above Although Unity 8 failed to make it into Ubuntu 15.10, there is a technology preview

1 Unity 8 technology

Canonical's new desktop environment, Unity 8, wasn't quite ready in time for the release of Ubuntu 15.10. However, all is not lost because it is being made available as a technology preview, with the promise that it's possible to try it out and cleanly revert back to the default Unity 7 if any problems are encountered.

2 Snapcraft tool

Designed for use with the embedded- and cloud-centric Ubuntu Core spin, Snapcraft has been designed to make the building of Snappy packages as simple as possible for the user, whether they are doing so from scratch or by taking advantage of existing deb packages.

3 Ubuntu Phone updates

Those running the Ubuntu Phone spin will be pleased to discover that they no longer need to manually update their operating system, with updates now being delivered automatically and directly to the handset, just like rival mobile platforms – a big step for the OS.

4 OpenStack Autopilot

Ubuntu 15.10 sees the general availability of OpenStack Autopilot, designed to make the deployment and management of Ubuntu-based OpenStack clouds as easy as possible. It also boasts in-place upgrade support between major releases in addition to input from the OpenStack Interoperability Lab.

5 LXD bundled as standard

The machine container hypervisor LXD is now included in Ubuntu Server by default, enabling any and all Ubuntu Server installs to host hundreds of other Linux guest containers without additional software. And if that has left you wanting more, there is a technology preview of a new RESTful API that is also available.

OPINION FREE SOFTWARE

Hacked off

"Access to computers – and anything which might teach you something about the way the world works – should be unlimited and total"

When Edward Snowden joined Twitter, he gained over a million followers in one day. He is notorious to some, a hero to others, and yet his greatest contribution has been to reveal the extent to which hacking, in its pejorative sense, is the preserve of government agencies. These agencies claim to be working for 'our own good', but are instead busy setting malign standards for everyday surveillance of our citizenry.

The pejorative use of the word 'hacker' probably originated in the tabloid coverage of the exploits of Kevin Poulsen (who hacked his way into the US Department of Defense's ARPANET, the precursor of the Internet, when he was 17 and was later labelled "the Hannibal Lecter of computer crime") and Kevin Mitnick (who broke into the North American Defense Command (NORAD) systems in 1982 when the Cold War was at its height, a feat that inspired the 1983 film *War Games* and provoked any number of nightmares for the morally insecure).

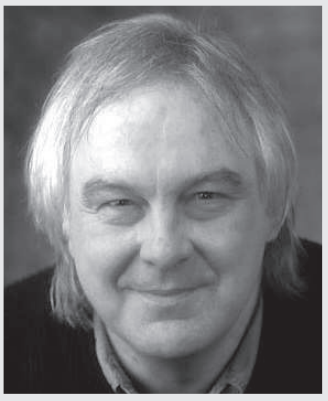
However, the terminology of hacking grew out of the community that evolved around the unlikely and much more benign setting of the TMRC (Tech Model Railroad Club), and its offshoot The Midnight Requisitioning Committee, at MIT (The Massachusetts Institute of Technology) in the late 1950s and early 1960s. The hackers of the TMRC defined a hack as "a project undertaken or a product built not solely to fulfil some constructive goal, but with some wild pleasure taken in mere involvement". They also defined a hacker as a person "who enjoys learning the details of programming systems and how to stretch their capabilities, as opposed to most users who prefer to learn only the minimum necessary", or one who "programs enthusiastically or who enjoys programming rather than just theorising about programming" (from the original jargon.txt at the AI Lab at MIT).

The Midnight Requisitioning Committee was so named because, in the words of Steven Levy, "when TMRC needed a set of diodes or some extra relays, to build some new feature" into the model railroad system, some of them would "wait until dark and find their way into the places where those things were kept. None of the hackers, who were as a rule scrupulously honest in other matters, seemed to equate this with stealing." The allegiances of the hackers transferred from under the model railroad layout tables to the computer when a TX-0 computer arrived in building 26 of the MIT complex and the members of TMRC discovered that the best time to gain access to the TX-0 was at night, "when no person in his right mind would have signed up for an hour-long session on the piece of paper posted every Friday beside the air conditioner in the RLE lab".

From this point onwards, "the TMRC hackers, who soon were referring to themselves as TX-0 hackers, changed their lifestyle to accommodate the computer," and became the core members of the AI (Artificial Intelligence) Group at MIT, later funded by the Defense Advanced Research Projects Agency (DARPA) of the US Department of Defense. Former TMRC hackers developed the first LISP machine, the first computer workstation, the first computer games, the first music software and the first display hacks. Hackers were the heroes of the computer revolution and the hacker tradition always included a healthy disrespect for authority and a reciprocal love of pranks. Every idea and every limitation was another boundary to be explored, a means to an adventurous end that might upset the applecart or translate into a better idea; especially if a higher moral reason could be found to justify the ends, and that reason, as defined in The Hacker Ethic, was:

"Access to computers – and anything which might teach you something about the way the world works – should be unlimited and total. Always yield to the hands-on imperative!"

Free software can also trace its origins to the hacker subculture of the AI Lab at MIT. The greatest hack of all was probably the hack on copyright that gave us copyleft and the GPL. Hackers are and were the good guys of computing. Someone who steals into our computers to pry or take our identities is a 'cracker'. After the revelations during the last decade of the 'hacking of phones' by our popular press, it is probably too late to rescue the original meaning entirely, but it is probably worth remembering that a 'hack' is also a journalist without any principles. ■



Richard Hillesley
writes about art, music, digital rights, Linux and free software for a variety of publications

"Hackers were the heroes of the computer revolution and the hacker tradition always included a healthy disrespect for authority and a love of pranks"

DATABASE

PostgreSQL 9.5 boosted by €1 million EU investment

2ndQuadrant demonstrates positive growth

The development of PostgreSQL, recently released as version 9.5 Beta, has benefited from EU funding thanks to 2ndQuadrant's dedicated work and coordination of a major research project called AXLE. 2ndQuadrant facilitates the development of PostgreSQL and is a world leader in PostgreSQL support services.

The research initiative received €1 million in R&D funding from an EU FP7 grant and has served as a catalyst for the development of PostgreSQL. As the three-year project comes to an end this month, the team at 2ndQuadrant will have contributed more than 10 years of effort towards enhancing PostgreSQL through the AXLE project.

AXLE (Advanced Analytics for Extremely Large European Databases) has brought together a diverse group of researchers, all focused on solving the needs of extremely large data analysis. With PostgreSQL at the heart of the project, the goal was to produce positive outcomes that could benefit many – in this case, several new – advanced features, which will make their way into PostgreSQL in the next releases. Specific to PostgreSQL, the outcomes include atomic locking, Block Range indexes, DDL triggers and a TABLESAMPLE feature.

AXLE has also breathed life into Postgres-XL 9.5, a scalable PostgreSQL-based database cluster that allows both business intelligence and transactional workloads. Postgres-XL is the first open source project to offer these capabilities and is developed communally by 2ndQuadrant, Huawei, NTT and a growing community.

INTERNET OF THINGS

Dell IoT Edge Gateway gets Ubuntu Core support

First units due to ship in December 2015

Dell has announced a new Internet of Things (IoT) gateway product, the Dell Edge Gateway 5000 Series, which brings support for Ubuntu Core and its Snappy management architecture.

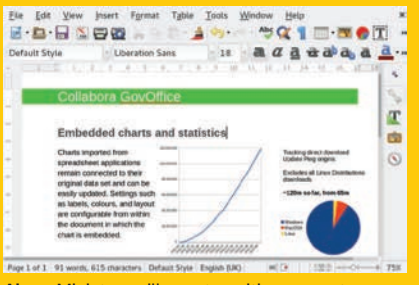
Announced back in 2014, Ubuntu Core is Canonical's cloud- and embedded-centric variant of the Ubuntu Linux distribution. One of the key changes is its use of Snappy, a new management system which uses atomic transactions to allow any changes to be rolled back on demand.

It's this that has attracted Dell to the platform. "Snappy Ubuntu Core's ability to automatically roll back to the last known working version of the OS is invaluable to our IoT customers," explained Dell's director of IoT strategy and partnerships Jason Shepherd at the announcement. "Should an OS update fail to boot due to any reason, Ubuntu Core will automatically revert back to its last known good OS version and keep our customers' information flowing to the cloud."

The Intel Atom-powered Dell Edge Gateway 5000 family, launching in December for commercial use and March 2016 for industrial users, is designed to collect data from IoT sensor networks and perform analytics before transmitting to remote servers for analysis.

OPEN SOURCE

Collabora signs UK Gov cloud agreement



Above Ministers will soon transition across to open, cloud-based productivity suites

Buyers for the UK Government have signed a commercial agreement with Collabora Productivity that will see open source productivity software used on desktops, mobile devices and cloud servers, in a bid to save money on proprietary equivalents.

Set up to offer LibreOffice products to enterprise users, Collabora is shortly to launch a cloud-based, browser-accessible version of the package dubbed CloudSuite. This, and a customised version of LibreOffice created specifically for governmental use – GovOffice – will be available to all non-profit-making governmental organisations.

"Moving to the cloud, adopting open IT standards and saving the taxpayer money are three key Government objectives achieved by today's agreement," claimed Michael Meeks, Collabora general manager. "Collabora will work with The Crown Commercial Service to raise awareness of the benefits of the Open Document Format (ODF) as well as the more general benefits and cost savings of open source client solutions," Meeks added. "An integral part of this agreement reflects the level of support and commitment Collabora will provide in the deployment of the Open Document Format."

The UK Government has been shifting away from proprietary formats since a scathing report from UK Cabinet Office Minister Francis Maude. Its current guidelines state that documents should be provided in open and accessible formats, including HTML, ODF and plain text, and that proprietary formats are to be avoided.

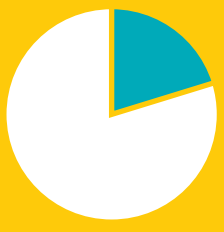
DISTRO FEED

Top 10

(Average hits per day, 2 October – 26 October)

1.	Linux Mint	▼	2,892
2.	Debian	▼	2,094
3.	Ubuntu	▲	1,790
4.	openSUSE	-	1,519
5.	Fedora	▼	1,175
6.	Manjaro	▲	1,115
7.	Mageia	▼	1,084
8.	Android-x86	▼	849
9.	CentOS	▼	831
10.	Arch Linux	▼	773

This month



Stable releases **(27)**
In development **(6)**

Ubuntu has enjoyed a surge of interest this month thanks to the release of 15.10 Wily Werewolf alongside official spins.

Highlights

Ubuntu 15.10
Ubuntu absolutely dominated this month's release figures, largely thanks to the simultaneous launch of its Desktop, Server, and Core variants alongside the various translated, task-focused and desktop-environment-specific official spins.

Android-x86 5.1 RC1
While Android-x86's Distrowatch traffic was down this week, larger drops in CentOS and Arch saw the port of Google's Android Open Source Project (AOSP) hit the top ten chart.

Linux From Scratch 7.8
Linux From Scratch is a freely-distributed guide to building your own distribution by Bruce Dubbs. The latest release includes an alternate version based around systemd.

GRAPHICS

Intel releases OpenSWR rasteriser

Promises up to 51-fold performance increase

A small team of engineers at Intel has released an open source software rasteriser, which it is claimed offers a 29- to 51-fold performance improvement over systems like llvmpipe.

Dubbed OpenSWR, the tool acts as a virtual graphics processor and interfaces with Mesa3D for API and state tracking layers. It's this project to which the team has contributed the code, targeting users working with large geometry models but who do not have access to hardware graphics processing capabilities – remembering, of course, that Intel produces CPUs but not dedicated GPUs.

"Our customers prefer open source, and allowing them to simply download the Mesa source and enable our driver makes life much easier for them," Intel's Tim Rowley explained of the decision to release OpenSWR under the permissive Mesa MIT licence. "It's easier to work with the Mesa community when the source we're working with can be used as reference."

Intel has stated that the company is targeting applications based on the Visualisation Toolkit, and that OpenSWR passes its OpenGL 3.2 backend rendering tests at 99 per cent, and that it is looking to improve conformance to other suites.

OpenSWR is compatible with any x86 processor featuring either AVX or AVX2 extensions, though Rowley has admitted that tests have not been carried out on rival AMD's processors. Details and the source are available at github.com/OpenSWR.

CAREERS

Microsoft to hire Linux experts

We've come a long way from the Halloween memos

Microsoft's Mark Russinovich has placed a call-out for Linux professionals to send their CVs to the company, as his Azure cloud computing division eyes expansion.

Microsoft has traditionally competed against open source software in general and Linux specifically, with a series of leaked memos in October 1998 – known as the 'Halloween Memos' – proving particularly aggressive. In recent years, however, the company has revised its stance.

Speaking during a keynote at All Things Open, Azure chief technology officer Russinovich pointed to considerable growth in the use of Linux on Azure and his company's need to support this.

Jokingly asking any Linux experts in the crowd to pass up their résumés, Russinovich claimed that the company is eager to fully support the open source community – a far cry from its traditional approach of 'embrace and extend,' to which critics appended 'extinguish'.

A slide accompanying Russinovich's speech claimed that there are 500 Linux-specific job openings at Microsoft, and a further 330 relating to open source projects.

Arbitrary integer size

Learn the problems associated with the simple addition of two numbers and then follow along with our two solutions

The problem that will be examined this month is how to add integers that are of arbitrary size. The programming language that will be used is C, but the techniques that will be presented are directly applicable to almost all modern programming languages.

Two crucial questions will come up sooner or later: firstly, how do you deal with overflows? Secondly, will you support signed integers? Neither of them are difficult to answer, but the point is that you should think about them before you start developing your program because changes at a later stage will take more time.

A rational choice would be to use an array to hold all the digits of an integer. If you really want to support infinite digits, you can use a text file for storing the digits of each integer. So two implementations will be presented. The first one will use an array and the second a text file to store each integer. Each approach has its advantages and drawbacks. The first one (**add.c**) is easier to implement and quicker to execute as it does not have to read or write to a file. The latter (**addFile.c**) is more flexible, uses less memory and can deal with overflows without the need for any extra code; you just write more data to a file. **AddFile.c** will not deal with how to organise your files that contain the integers; it will just read two big integers stored in two text files and save the result of their addition into a new file. All filenames will be given as command line arguments.

The main decision that you will have to make in **add.c** is about the size of the array. You can start with a relatively big array size and see how it goes before increasing it. This is not a major issue, however you will need to recompile **add.c** in order to increase the size of the array. For reasons of simplicity, **add.c** will deal with unsigned integers only. So if you give two integers with a size of n digits as the input of **add.c**, it will automatically store the result of their addition into an array that also has n places. Currently, **add.c** adds two randomly generated integers.

In order to understand the way **add.c** stores an integer into an array, look at the following example:

- You want to store 123 into an array.
- You have an array with 10 places that you will have to initialise by putting a zero everywhere.
- The integer will be stored in the array as [0,0,0,0,0,0,0,1,2,3].

Please note that this method will unfortunately only work for fixed size arrays.

Once you have this arrangement for the way you store your integers, everything is simple because you only have to add the elements that can be found at the same index number of the array. In case of overflow, you can just print a message on the screen.

The most challenging point that **addFile.c** must resolve is how to store each integer. The most convenient way of storing an integer is in reverse order. You will understand why when you realise that it is easier to implement integer addition in **addFile.c** when the integers are in reverse order. So **addFile.c** will just read the two files character by character and then save the result into a new file using the same way.

The three calls of the **system()** function use a Perl one-liner to reverse the input and present the digits of all integers in their correct order. You can remove it if you want and replace it with a simple **/bin/cat** command.

If you are a supporter of simple solutions then you should definitely go with the **add.c** implementation. However, if your purpose is to have the 'perfect' solution then you should choose **addFile.c**. Your choice of the approach should also be based on your input.

At this point you may well be asking: what about subtraction, multiplication and division? The approach for each one of them should be similar to the integer addition. However, there are three tricky points that you should consider beforehand:

- If you divide two integers, the result could be a floating-point number, which does not happen with the other three operations.
- The multiplication of two integers might quickly grow to be quite large because the result of a multiplication of two integers with N digits each can have as many as N+N digits.
- The subtraction of two integers can have a signed integer as a result.

You can find the source code to follow this article at **github.com/mactsouk** and at **filesilo.co.uk**. ■

Mihalis Tsoukalos
is a UNIX administrator, a programmer, a DBA and a mathematician. He has used Linux since 1993

> If you are a supporter of simple solutions then you should go with the **add.c** implementation. However, if your purpose is the 'perfect' solution, then choose **addFile.c**

INTERVIEW MITCHELL HASHIMOTO

Otto: Meet the new Vagrant

Mitchell Hashimoto, founder of HashiCorp, is a developer obsessed with automation. He is the creator of Vagrant, Packer, Serf, Consul, Terraform, Vault, Nomad and Otto. He is a key DevOps community figure, a technical author, and a seasoned engineer with enviable experience in the fields of operations and research

Otto and Atlas

Atlas is HashiCorp's only commercial product, funding its eight open source projects. Mitchell tells us that Otto to Atlas is like Git to GitHub – it gives you a core workflow, all the primitives, a lot of the complex logic. Eventually, it will give you a full UI to Otto. Atlas will also be the collaboration platform for Otto, checking that nobody else is deploying a particular application at the same time as you, and enabling you to control which applications people can deploy, and even the times at which they can deploy them. Atlas will also be able to run a lot of the Otto commands for you, enabling you to offload activities and, for example, close your laptop while an **otto deploy** is running.

Mitchell Hashimoto integrates development and deployment in Otto, the next generation of Vagrant. We asked how it will simplify software production for you

What is Otto, in a nutshell?

Its goal is to do everything as well as, if not better than, Vagrant could do from a developer perspective. And then we also added deployment. That's the feature difference, but there are also philosophical differences.

The major philosophical difference is what we call fossilisation versus codification. If you take a Vagrantfile from five years ago and run **vagrant up** today, it'll probably still work and bring up that environment exactly how you set it up five years ago. That's because a Vagrantfile is a fossil – a snapshot in time. For some use cases you want a fossil, but what we've determined over the past few years is that what you *really* want is codification. So if you look at the Appfile, which is Otto's configuration format, instead of being prescriptive on how to do something, it's declarative on what you want to get done. So you're saying, 'I want to develop a PHP application', but you're not saying how to install PHP, you're not saying what OS to use, and that's because the knowledge of how to do that is centralised in Otto. So if you run Otto today and you run Otto five years from now, it's likely that different things will happen because we're going to use current best practises.

The main difference is that Appfile describes the application and Vagrantfile describes the machine.

The website says that Otto is built for microservices – what do you mean by this?

We went through a phase where there were a lot of monoliths and now we're coming back to service-oriented, but with lots of little services versus a handful of bigger services. This has made development difficult because you need to know how to install a handful of services as well as your application, which might be in different languages – completely ignoring deployment. And so we tried to solve that with Otto, with the Appfile that you use to describe and configure Otto.

You put in the pointers to URLs where the dependencies are, and Otto reaches out to those, downloads them and reads their Appfiles, which describe how to install them for development. It puts

Above Otto uses a declarative approach, which makes life much easier for developers

Otto's Appfile

Configuration can be minimal, leaving it all to Otto's auto-detection, or quite detailed. You can also share blocks between Appfiles

Appfiles are completely optional. If no Appfile exists, Otto inspects the project to detect what kind of application it is. It then generates an internal Appfile, which is not saved externally. Individual blocks within an Appfile are also optional. If an Appfile exists and a block such as "application" is missing, Otto will just merge in the detected data.

Fragments from other Appfiles can be imported – this is the recommended way of sharing configurations for Appfiles, especially the project and infrastructure blocks (see line 1).

If the application name and type properties (lines 5 & 6) are not included in the Appfile, Otto auto-detects the name of the directory the Appfile is in for the name, and uses filename pattern matching to determine the type – for example, the presence of a Gemfile indicates a Ruby application.

Otto recognises GitHub URLs (line 8) and turns them into the proper Git repositories, and also accepts generic Git repos. It also recognises BitBucket URLs and turns them into Git or Mercurial repos, as well as accepts generic Mercurial repos. Finally, you can specify local file paths.

There can be multiple customisation blocks (line 13). The name of the block is the component to customise.

As with the application block, the name and infrastructure properties (lines 19 & 20) in the projects block can be left out and Otto auto-detects them.

Infrastructure (line 23) is currently hardcoded to be AWS with the 'simple' flavour. The other available flavour for AWS (line 25) sets up a more production-like infrastructure that includes a VPC with both public and private subnets, and more besides.

Sample Appfile:

```
001 import "github.com/myorg/otto-shared/go" {}
002
003 # Application block
004 application {
005     name = "otto example"
006     type = "go"
007
008     dependency { source = "github.com/hashicorp/otto/examples/mongodb" }
009 }
010
011 /*
012 Let's disable this for now using a multi-line comment
013 customization "go" {
014     go_version = "1.4.2"
015 }
016 */
017
018 project {
019     name = "test project"
020     infrastructure = "production"
021 }
022
023 infrastructure "production" {
024     type = "aws"
025     flavor = "vpc-public-private"
026 }
```

See the Otto docs for more info on setting up your Appfiles: **bit.ly/1SiXMFN**

the burden of installing a service for development as a dependency onto the service creator. That makes things a lot simpler, and the developer of the service has a lot more flexibility; completely changing the language won't affect your downstream developers.

Microservices are still new. I don't think that they're mainstream yet – we're just trying to plan for the future.

Why was there a need to release Otto as a new product, rather than improve Vagrant?

Using programming languages as an example; if you wrote in an object-oriented language and suddenly felt that functional programming languages were the future, I would claim it would be unwise to bolt on features to an object-oriented language because they don't get along. In the same way, using the machine versus app example of Vagrant, I took a look at a Vagrantfile and thought that it just didn't fit to put something higher-level there, and also Vagrant is very widely used – it gets millions of downloads a month – and there's a lot of backwards compatibility concern. If you want something that's six years old, battle-hardened and definitely going to work, then Vagrant's there. If you're using microservices, you want the cutting edge, then Otto is there – it's going to be rock-solid eventually, but it's still very much 0.1.

What's the next step for Otto?

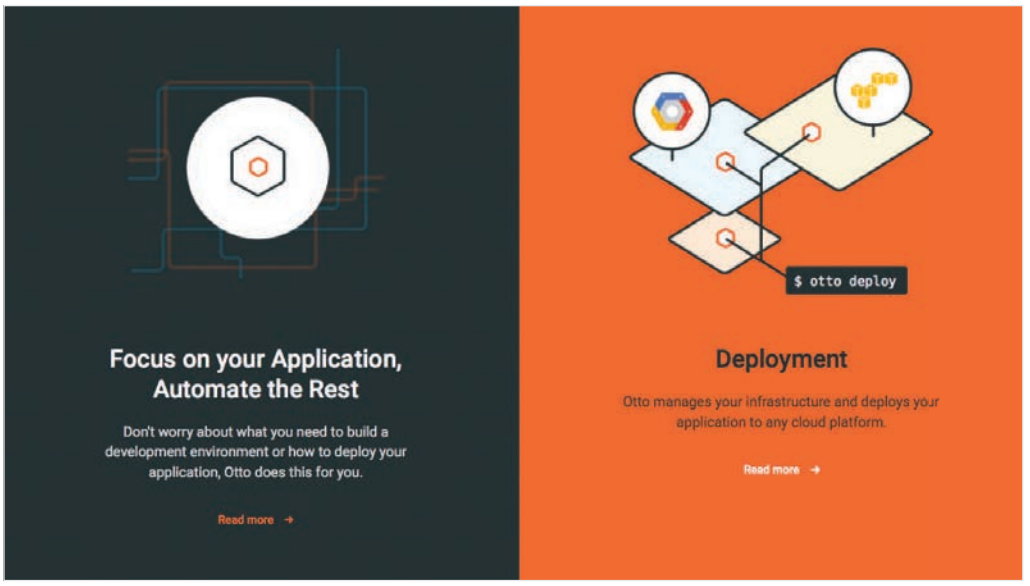
I'm not sure how many people reading this will have used Vagrant 0.1 six years ago, but Vagrant 0.1 was, self-admittedly, very bad. It required Ubuntu, it only worked with VirtualBox, it only worked with Chef as a provisioner – weird things like that. When I look at Otto, I feel very much the same way – I think that it's restrictive right now, for the purpose of showing you what it's capable of. But over the next few versions we're going to eliminate a lot of those restrictions. So in 0.2, for example, the big feature is the **otto dev** command, which is the equivalent to the **vagrant up** command. The **otto dev** command for basically any application type takes about five or ten seconds, versus Vagrant where it might take about a minute. So we're working really hard to make this fast, to make it really enjoyable.

> "After we make the development experience really good over the next couple of versions, the focus is going to be making the deployment experience production-ready"

We're focusing on development first. There's a disclaimer on the Otto website that while Otto *can* deploy, we don't recommend doing that yet. **Otto deploy** is mostly a demo of what it's capable of but it's not quite production-ready. After we make the development experience really good over the next couple of versions, the focus is going to be making the deployment experience production-ready. That'll take us through the next six months.

What were the limitations of Vagrant that led you to create Otto?

First, we noticed that everybody's development environment within a certain language or framework was very similar – almost identical except for one or two things. While everyone likes to believe that their development environment is special, that's not what we were seeing across the hundreds of thousands of users we were looking at. So we can build you a very common Ruby environment, a very common PHP environment,



Right It's not quite there yet, but Otto will soon be ideal for deployment as well as development

How to use Otto

The following is an example use case for Otto, which illustrates both the development and deployment steps for a Rails application on AWS.

1 Start your Rails project
Initialise a new Rails application with out-of-the-box defaults:
```
$ rails new
```

2 Compile and configure
Otto's compile step automatically detects the type of application as Rails and then prepares Otto to manage your app's lifecycle. You can customise nearly every aspect of Otto's behaviour by writing an Appfile.
```
$ otto compile
```

3 Develop locally
Otto builds and runs a local development environment, so you can easily test and make changes to your application:
```
$ otto dev
```

4 Construct infrastructure
Otto creates a base infrastructure according to industry best practises:
```
$ otto infra
```

5 Build your application
Otto packages your application and prepares it for deployment onto your infrastructure:
```
$ otto build
```

6 Deploy
Your Rails application is converted to running infrastructure – your app is now readily available to the world:
```
$ otto deploy
```

and the idea is that you only need to customise the little differences above that. You don't need to recode the common ground.

The second thing was deployment. Even years ago, at Vagrant 0.1 or 0.2, we got a feature request which was, 'Can I **vagrant up** to production?' We tried over the years to make that possible, and ultimately we found that the Vagrantfile isn't a good description of how to get something to production, because the Vagrantfile describes, for one, a lot of machine-level things that are hard to translate to something like AWS, but in addition it just describes a lot of development things you don't want in production. So we tried a bunch of things but the one that felt the best was just recreating a whole new file – you would have your Vagrantfile for development and your other configuration for production, and it just felt weird. The main thing we're doing to address it is moving up an abstraction layer. By describing the application rather than the machine, I think we have the right configuration to do both development and production.

The third thing was microservices. Like I said, this isn't mainstream yet, but it's fast-growing. It's hard to describe all your dependencies in one Vagrantfile, and using multi-VM was too heavy for a microservice. If you have a half-dozen microservices then you're running six virtual machines, and that's going to take down a lot of basic development machines – multi-VM was never made for microservices anyway.

A lot of people turned to Docker, right?

Yeah, so we saw that a lot of people were trying to use Docker for development for microservices. Docker is great at packaging microservices – wrapping microservices up in a container is a great way to make it easy to run them. The main issue we saw, from a Vagrant perspective, was that using Docker didn't really solve all the problems. You had the packaging problem solved, but the burden of what services to install and how to configure them was still on the end application developer. So if you're developing an app, you'd have to know all the dependencies you had – which you still need to know – but then you would also have to know all *their* dependencies. The approach we took with Otto is that you only need to tell Otto your immediate dependencies – Otto will figure out your transitive dependencies and then it's up to that dependency to tell Otto how to install, run and configure it.

What do you think the learning curve will be like?

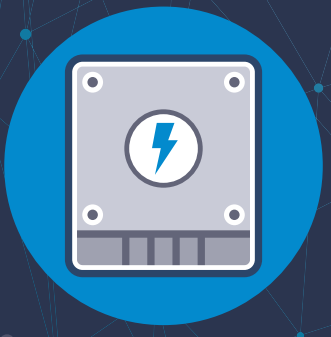
I think, from the development side, it'll be very familiar for Vagrant people. The Appfile itself is different, so that will take some getting used to, but **otto dev** is like **vagrant up**; **otto dev ssh** is like **vagrant ssh**; **otto dev destroy** is like **vagrant destroy**; there are a lot of similar things. At the same time, from the development side, Otto is going to do a lot of things for you that Vagrant developers are really going to like. It automatically finds a non-conflicting IP address and gives the machine a static IP, so you no longer need to configure networks. There's a command to get that address – **otto dev address** – so you don't need to remember it like you had to do with Vagrant.

The deployment side is new for us, so we've tried to build a workflow that's as easy as Vagrant. I think, though, that ultimately the learning curve for Otto is going to be low. The cost of this simplicity is that Otto makes a lot of decisions for you – we fully expect that's going to make people uncomfortable. But if you're uncomfortable, and it's a legitimate reason, then we'd love to hear from you so we can encode that knowledge into Otto and make it do what you're comfortable with, if that is the best choice for the industry. ■

Cluster

Our revolutionary NEW Web Hosting platform

100% guaranteed uptime!



Smart SSD storage & intelligent load balancing

Dedicated SSL certificates

Web Hosting from:

£1.99

per month ex VAT charged at 20%

Call **0333 0142 708**

or visit **fasthosts.co.uk/hosting**

SERVERS · WEB HOSTING · DOMAIN NAMES · EXCHANGE EMAIL

fasthosts

OPINION

The kernel column

Jon Masters summarises the latest happenings in the Linux kernel community, as Linux 4.3 is released and work continues toward 4.4

Jon Masters
is a Linux-kernel hacker who has been working on Linux for some 19 years, since he first attended university at the age of 13. Jon lives in Cambridge, Massachusetts, and works for a large enterprise Linux vendor, where he is driving the creation of standards for energy efficient ARM-powered servers

Linus Torvalds announced the release of version 4.3 of the Linux kernel, noting that things had seemed to be a bit busy with fixes towards the end of the development cycle, but that "doing the actual numbers shows that that really was just my subjective feeling, probably due to the kernel summit and travel back home from Korea". The latest kernel includes several exciting new features – the PIDs controller and user space page faults are among them – as well as the removal of the venerable ext3 filesystem (though readers shouldn't worry about the latter, as we will explain).

The PIDs controller feature in Linux 4.3 is based upon a simple enough idea: prevent the 'fork bomb' attack that used to be common on older Unix systems. In a fork bomb, the attacker runs code that tries to create as many new processes as possible, which ties up system RAM and computer resources. As a result of this, Linux (and Unix) systems, have long instituted seemingly arbitrary limits upon the maximum number of processes allowed (known to the kernel as tasks). Indeed, on a Fedora laptop, the **ulimit** command informs that the author's user account is (by default) restricted to creating a maximum of 1,024 processes. But working at the per-user level for limits is very coarse grained. The new controller group approach enables a group of processes (such as those started as part of a container) to be limited in their forking of further processes.

Support for userspace page faults in Linux 4.3 will bring new capabilities to diverse areas, including that of virtualisation. Virtual machines, when run under KVM on Linux, leverage the existing 'host' kernel to provide most of the needed supporting drivers and infrastructure, while abstracting the running VM as a regular process. That process has a userspace component based upon a modified QEMU that provides supporting capability for emulating hardware (and other resources) made available to the VM. Now, it will become possible for that QEMU to also handle page faults on behalf of the virtual machine during live migration activity, allowing for background migration of a running machine while tracking any modifications it makes to its memory during the migration process in a very lightweight fashion, ultimately speeding up live migration and reducing the latency impact on the VM.

With the release of Linux 4.3 came the opening of the merge window (the period of time – usually just shy of two weeks in duration – during which disruptive changes are allowed) for what will ultimately become Linux 4.4. At this time, one of the more interesting features being tracked for 4.4 inclusion is that of richacls (aka native, not mapped to POSIX, NFSv4 ACLs on filesystems), which will be of interest to networked and enterprise users in particular. We will have a full summary of the features that make it into 4.4 in next month's issue. Meanwhile, it's exciting also to see that support for the ultra low-cost C.H.I.P. "$9 computer" (based upon a sunxi 32-bit ARM chip) is probably going to land in 4.4 in time for the first batch of shipments.

Finally this month, let's spare a thought for Jaccon Bastiaansen, who posted a lengthy message entitled "Possible race condition in kernel futex code", describing his use of an older 3.12.42-rt58 kernel on an 8-core 64-bit "Real Time" (hence the -rt58 patchset reference) x86 system. His setup leverages 50 threads, a custom memory allocator and various other complexity described in the initial message. The interesting part of the story is the level to which he (and perhaps some colleagues hinted at in the message) obviously went to in order to root out the fundamental cause of his problems.

Jaccon describes how the Linux 'pthreads' (POSIX Threads) code was falling over, claiming that a deadlock condition was arising with two threads simultaneously acquiring the same futexes (a construct used to represent a fast user mutex – hence "futex" – otherwise known as a lock). He wound up discovering that the compiled version of his kernel (when run through a disassembler) showed the kernel was checking the lock using unsafe and non-atomic byte-level accesses where it should have been using a word (on Intel, that means 4-byte) check. The problem was reproducible with certain compiler versions on his older kernel, but it had been fixed in newer kernels, as Thomas Gleixner noted. The thread makes for good reading for its depth: **https://lkml.org/lkml/2015/10/5/400**.

Farewell, ext3

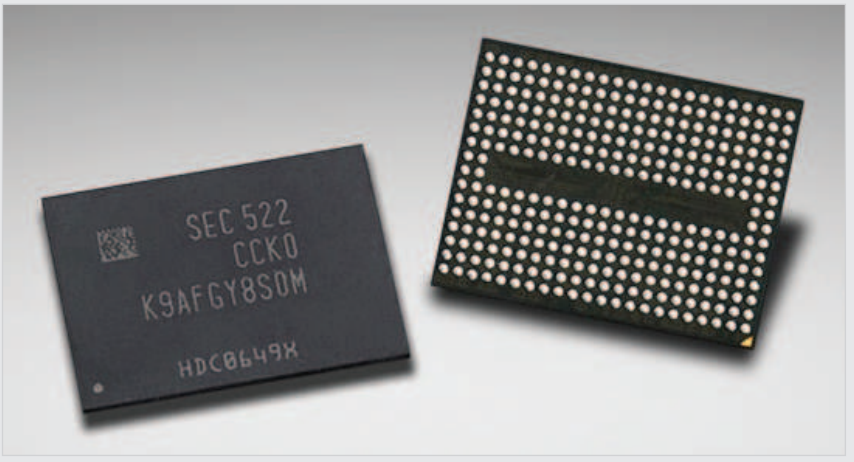
The year was 2001. Linux version 2.6 was still a future aspiration, and kernels came in an odd/even cycle of multi-year, odd-numbered 'development series' releases, followed eventually by a new 'stable series'. This was long before Git (Linus had not become frustrated enough to write it yet), and before the contemporary two-month development cycle with a merge window that has obviated the need for separate development series kernels. In 2001, a major new kernel feature could languish for years in obscurity waiting to land in the next stable series kernel. It was, then, with some considerable relief that ext3 support was added to Linux 2.4.15 without waiting for the (2003!) release of the first 2.6(.0) kernel.

Ext3 brought something new to Linux that it had not had before: mainline support for filesystem journaling without applying many 'out of tree' (unofficial) patches. Journaled filesystems are more resilient against power failures because they separate their actions into transactions that are first written to a 'journal' before the on-disk filesystem data structures are updated. If power fails during this process, the filesystem has enough internal state to recover its metadata without forcing the user to run a special filesystem checking program. To end users, this means that a power failure during filesystem activity (that is to say, nearly any time) need not result in a lengthy use of the **fsck** (filesystem check) utility.

Developers found the occasional need to run **fsck** frustrating, while consumers could be forgiven for being highly confused about the scary sounding messages printed on their screen about "inconsistent" filesystem state. Indeed, this author recalls in his very early days of Linux use (several decades ago), being so terrified by such confusing messages (in a time before Google) that he wound up reinstalling a system after an unclean shutdown rather than figure out **fsck**. Like many, then, it was a relief to hear about the work that Stephen Tweedie had done to add journal support to the venerable ext2 (extended second) filesystem.

Journaling was not a new concept. At the time of the introduction of ext3 into Linux, Microsoft Windows users had had the benefit of built-in support for NTFS (which was arguably far more sophisticated) for some years. Yet unlike NTFS (which had an awkward upgrade path from FAT32 for users), the design of what became ext3 was deliberately based upon an evolutionary approach. It might have been more efficient to throw out the design of ext2 entirely (and follow the path of others, such as XFS, which came from SGI's Unix and was added to Linux finally in 2004) but ext3 had the distinct advantage of supporting in-place upgrades of existing ext2 filesystems.

The new filesystem added a journal to an existing ext2 filesystem, and indeed for some time it was possible to boot systems with the journal disabled (treating them as ext2) in order to mix older and newer kernels. Eventually, the long heritage of ext3 caught up with it, and in 2006, Ted Ts'o (who had long since taken over as maintainer of ext3) introduced what would become ext4. Eventually, the ext4 driver gained support for older ext3 filesystems (rather than needing the older ext3 driver). And that's why, in 2015, the older ext3 driver has finally been removed. Yet those who are still using ext3 can continue to do. Linux will transparently run any remaining ext3 filesystems using the modern ext4 driver. Eventually, it is believed btrfs will largely replace ext4, although far from transparently. ■



Above Btrfs will become important as next-gen storage tech, like Samsung's 256Gbit 3D V-NAND flash memory, drastically increases SSD sizes and storage pooling becomes more popular among users

"Linux will transparently run any remaining ext3 filesystems using the modern ext4 driver. Eventually, it is believed btrfs will largely replace ext4"

More on DevOps
Otto's on p12 and CoreOS is on p38

CREATE SOFTWARE FASTER

Deploy and orchestrate containers like a pro

DevOps is the latest buzzword to hit the software industry – the practise promises quick delivery to customers with the least friction between various teams. As is to be expected, everyone wants to implement DevOps. But just like medicines that should be taken in a controlled manner and fixed doses, DevOps should be adopted in a phase-wise manner, keeping in mind major factors like human resources. Not doing so will only lead to chaos and misunderstanding among teams.

The first step in order to properly plan DevOps implementation is to understand DevOps itself, and then the various factors related to it. In this feature we will do just that. We'll begin with an introduction to DevOps, and best practices to make it work well, and we'll follow with some tutorials on DevOps-related tools.

On the tools side, we will cover Docker, Puppet and Vagrant. As you may know, Docker is a containerisation engine. It helps create containers that encapsulate software and its required dependencies in an efficient way. Puppet is a configuration management tool. It lets you describe system resources and their state, either using Puppet's declarative language or a Ruby DSL in files. Vagrant helps in automatically provisioning multiple virtual machines, with each having its own configurations managed by Puppet. Vagrant also has a Docker provider, which means you can manage several Docker containers with Vagrant.

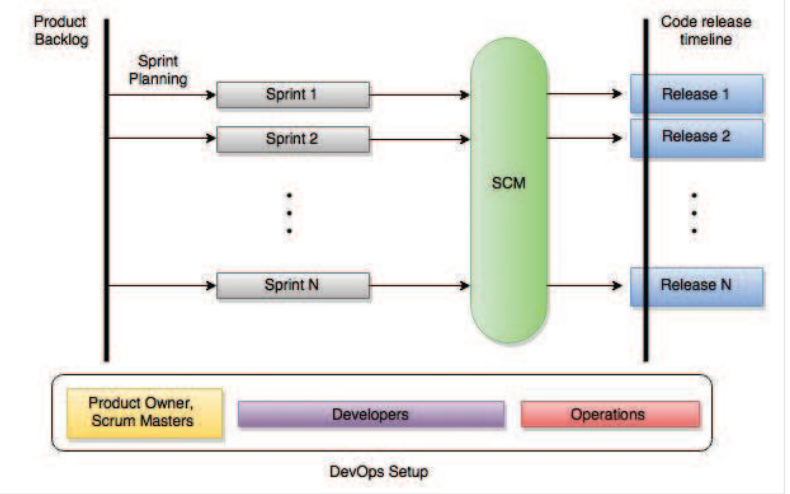
UNDERSTAND DEVOPS

Before implementing DevOps, let's understand the core DevOps concepts and typical scenarios where DevOps can help

So, what exactly is DevOps? Experts say that if you ask this of 10 people, you may get as many different answers. This is because of the lack of a standardised DevOps definition. In contrast, all the other major software development and deployment methodologies are based on specific definitions. But DevOps is based on the practical, real-life needs of modern organisations, rather than a study performed by researchers. Modern organisations need to be Agile in their operations; they need to increase the rate of production releases and reduce human intervention in redundant tasks like testing and continuous integration. They need to have a clear picture of the development and delivery pipeline. DevOps has the promise to achieve all of this and more.

It is probably futile to devise a definition for DevOps; rather it would be better to clearly understand the central ideas – automation; standardised infrastructure across development, testing and operations teams; all the team members having an overview that spans entire delivery pipeline, etc. All these actually point to just one direction – minimising delays due to unplanned infrastructure issues and faster, reliable deliveries to customers. Now, it is important to understand here that different organisations have different customers and requirements, and it is seldom possible to adopt a one-size-fits-all approach, so it becomes your responsibility to understand the delivery bottlenecks in your organisation and then adopt DevOps practices to sort them out.

Let's take a scenario of a typical software product development company – the team came up with a killer new feature and scheduled it to go to production in a week. The development team worked hard and released the code on time. The code was pushed to the development server, tests were executed and everything seemed to be working fine.



Above DevOps reduces the delay between a completed sprint and a code release inside an Agile workflow

Finally, the team got the sign-off and moved the code to the production server and the feature went live. It went well for a few hours before the users learned about the new feature. But as soon as the traffic grew, the dreaded 'Internal Error 500' page started to appear for most of the users. It was later learned that a critical server update was not executed on the production server. While the server could handle a few users without the update, it was impossible to handle huge traffic. The reason for this mishap was miscommunication among the development and the infrastructure teams. You get the point here – had the organisation adopted DevOps principles, there would have been clearer communication and this occurrence probably could have been avoided.

HOW TO IMPLEMENT DEVOPS WITH AGILE PRACTICES

Agile enlists clear guidelines about software development processes, but sometimes they prove insufficient for the needs of modern organisations. Let's take a look at a few simple tips and tricks to help you implement DevOps principles while staying within the Agile framework.

Product backlog: A simple way to enable better communication among teams is to add extra fields related to different teams – such as development, testing, infrastructure – in the product backlog. This makes sure everything is documented and there is minimal scope for miscommunication.

Scrum of scrums: Scrums are an integral part of Agile methodology. A meeting of scrum masters of different teams, ie the scrum of scrums, can make sure people from different teams are in sync.

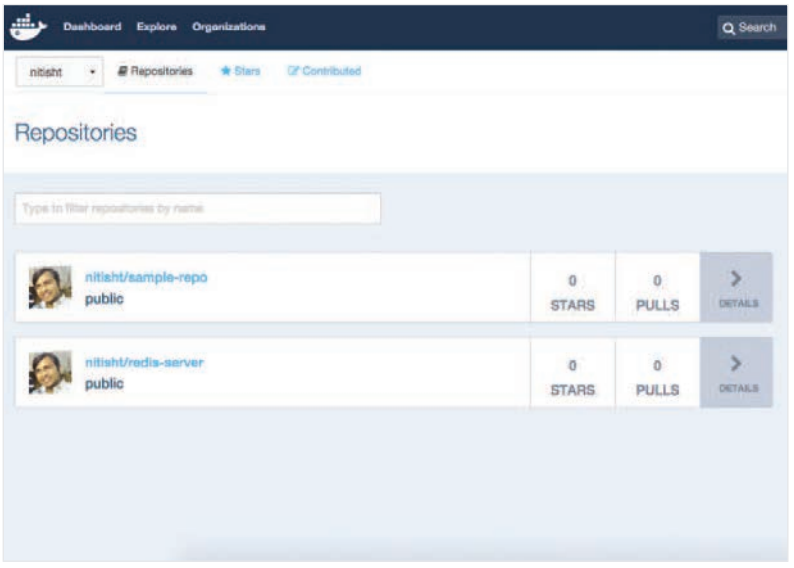
Done definition: Adding operations-related items to the done definition of a sprint is another handy approach to make sure operational issues are addressed well.

Focus on employees

DevOps suggests many changes in the way teams work. While these changes look simple from the outside, for employees who are supposed to change their way of working this can mean a lot of confusion. It is very important to understand that it is the employees who will actually make or break your DevOps dreams. So you should take them into confidence and train them well before implementing new DevOps tools.

EMPLOY DEVOPS WITH DOCKER CONTAINERS

Docker offers a great way to manage infrastructure across teams. Let's see how it works and how to use its features to implement DevOps



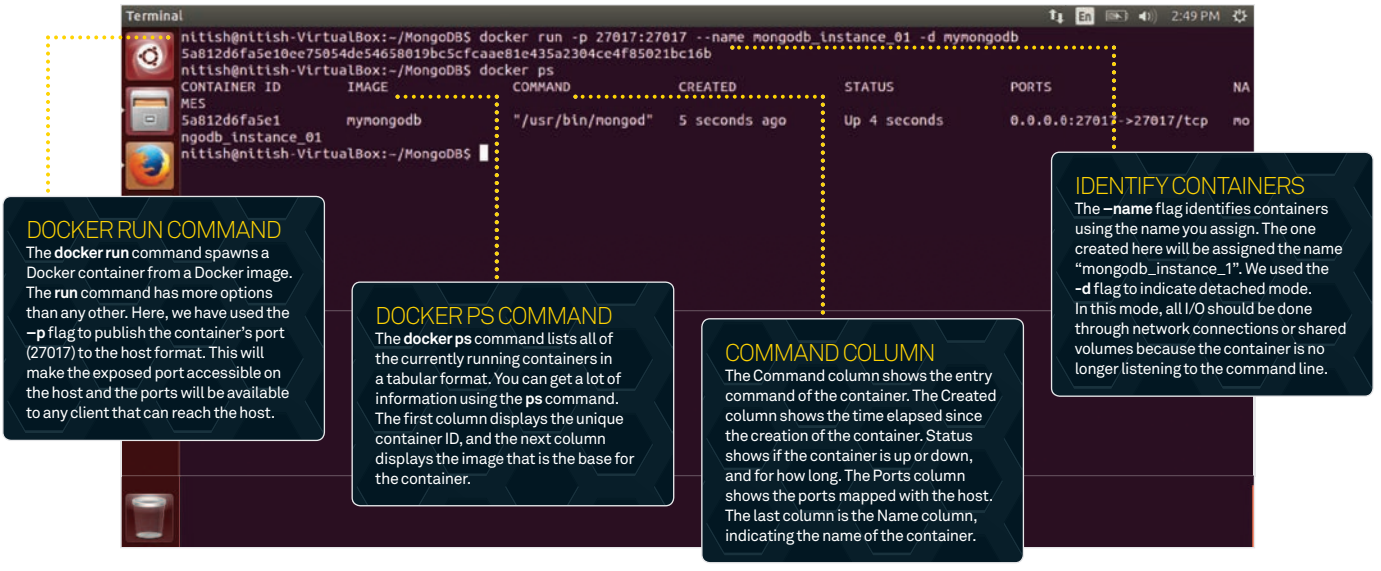
Above The Docker home page displays an overview of your repository activity

Docker and its uses

Containers offer a great way to run self-contained software units, handling any dependencies software may need internally. Docker takes this further with simple and easy-to-create Dockerfiles that serve as the building blocks of Docker containers. Add to this the comfort of a union file system approach, which enables files and directories of separate file systems – known as branches – to be transparently overlaid to form a single coherent file system, and you can overlay a container on top of another. So, how does all this add up and let different teams easily collaborate on the lines of DevOps? Here are a few use cases:

- Need that specific OS version across all the teams? Just create the Dockerfile for that OS version and distribute it. That's it! Anyone can now easily create the container from the Dockerfile and use it to execute the code.
- Need to execute tests on your code before shipping it? Create the Docker container for your code and deploy multiple containers of same system. You can now run concurrent tests on the same system.

MANAGE CONTAINERS IN DOCKER



DOCKER RUN COMMAND

The **docker run** command spawns a Docker container from a Docker image. The **run** command has more options than any other. Here, we have used the **–p** flag to publish the container's port (27017) to the host format. This will make the exposed port accessible on the host and the ports will be available to any client that can reach the host.

DOCKER PS COMMAND

The **docker ps** command lists all of the currently running containers in a tabular format. You can get a lot of information using the **ps** command. The first column displays the unique container ID, and the next column displays the image that is the base for the container.

COMMAND COLUMN

The Command column shows the entry command of the container. The Created column shows the time elapsed since the creation of the container. Status shows if the container is up or down, and for how long. The Ports column shows the ports mapped with the host. The last column is the Name column, indicating the name of the container.

IDENTIFY CONTAINERS

The **–name** flag identifies containers using the name you assign. The one created here will be assigned the name "mongodb_instance_1". We used the **-d** flag to indicate detached mode. In this mode, all I/O should be done through network connections or shared volumes because the container is no longer listening to the command line.

CREATE A DOCKERFILE

We've talked about how easy it is to create a Docker container for applications and deploy the container. Let us now create a container from scratch to understand the steps involved. As an example, let us create a container for MongoDB, an open source cross-platform NoSQL database, used very frequently by enterprise applications. To start, create a file named as Dockerfile in the directory where you want to create the image:

```
$ nano Dockerfile
```

Then open the file in your favourite editor and start adding the instructions. First, set the base image as Ubuntu using FROM ubuntu:latest. Then declare the maintainer of the Dockerfile:

```
Maintainer Name <email@address.here>
```

Let's now complete the installation prerequisites:

```
RUN apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 7F0CEB10
RUN echo "deb http://repo.mongodb.org/apt/ubuntu "$(lsb_release -sc)"/mongodb-org/3.0 multiverse" | tee /etc/apt/sources.list.d/mongodb-org-3.0.list
```

Then install MongoDB:

```
RUN apt-get update && apt-get install -y mongodb-org
```

MongoDB also requires a data directory. Create it using:

```
RUN mkdir -p /data/db
```

Finally, expose the relevant port from the container to the host and set the entry point as the MongoDB daemon:

```
EXPOSE 27017
ENTRYPOINT ["/usr/bin/mongod"]
```

Save this file. We can now move forward to build the image from this Dockerfile. To build the image, type this into the terminal:

```
$ docker build --tag mymongodb .
```

This tells Docker to build an image from the Dockerfile present in the current directory (note the . at the end) and tag it as mymongodb. You have the MongoDB image ready to run now. Let's spawn a container from the image:

```
$ docker run -p 27017:27017 --name mongodb_instance_01 -d mymongodb
```

This creates a container called mongodb_instance_01 and executes it. You can check the currently running containers using the command:

```
$ docker ps
```

Finally, you can connect to the MongoDB container by running:

```
$ mongo --port 27017
```

In-browser terminal

Wetty (Web+tty) is a JavaScript-powered in-browser terminal emulator. You can use the Dockerised Wetty to run a Linux shell in your browser; the Docker image is already available, in the DockerHub, so you just need to type:

```
$ docker run
-p 3000:3000 -d
nathanleclaire/wetty
```

Note that Wetty currently works in the Chrome browser only.

- Need to automate the testing process? Use tools like Deis that take the code from repository and then deploy it in the form of a Docker container on predefined hardware nodes.

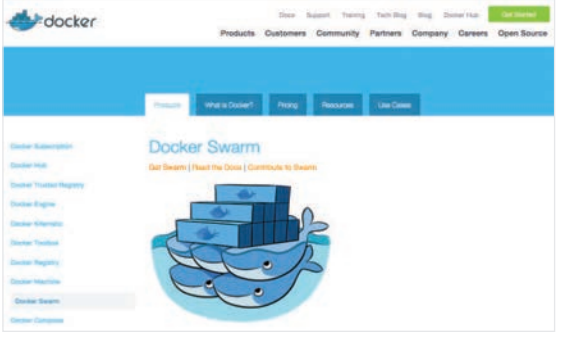
Docker provides great flexibility in implementing new ways to successfully implement DevOps. Let us now take a look at some key Docker features that make it the flexible tool it is.

Union file system

One of the lesser known features of Docker is the union file system. Though not explicitly visible to the end users, the union file system works in the background to enable the building of new containers on top of old ones, saving you from repeating things. Let's visualise the process. Every Docker image starts from a base image. As you add instructions in the Dockerfile, each instruction creates a new layer in the resulting image. When you finally run the image, the union file system combines all the layers into a single image, which is then run as a container. This layered approach means you can also use an existing image as the base for a new image.

Docker Registry

All of us know about GitHub. Docker Registry can be thought of as something similar for Docker images. Though not a part of your Docker installation, Docker Registry is tightly integrated with Docker and allows you to easily push/pull



Left If you need to manage many Docker images, make use of the Docker Swarm tool

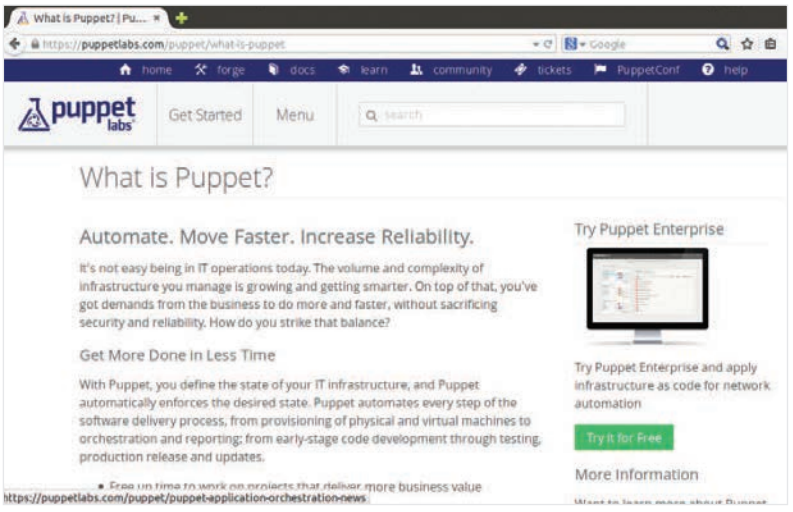
Docker images. From a DevOps point of view, this offers a great deal of connectivity. Even if teams are sitting across locations, they can easily share and distribute Docker images. If you are interested in the cloud-hosted Docker Registry provided by Docker itself, head over to **hub.docker.com**.

Docker Swarm

To be able to deploy and manage multiple Docker images on a cluster, Docker provides a tool called Docker Swarm. The prerequisites to start using Swarm are: VirtualBox installed on the local system, all the nodes have access to Docker to pull images, and each node is associated with a discovery service like etcd, Consul, ZooKeeper, etc.

AUTO-CONFIGURE NODES WITH PUPPET

Puppet enables you to manage the state of your IT infrastructure. Here we'll look at the Puppet installation process, then the master/agent setup



Above You can learn more about the uses of Puppet at **puppetlabs.com**

Automation is one of the key factors in implementing DevOps. The more you reduce human effort in repetitive tasks, the more you can get out of your workforce. IT infrastructure configuration is one such field that involves repetitive tasks – provisioning physical and virtual machines, orchestration, reporting and other similar tasks. Here are some advantages a configuration management tool offers:

- Code commit log shows you the whole history of change on the infrastructure
- Reproducible across systems
- Scale easily
- Coherent and consistent server setups
- Aligned environments for development, testing, and production nodes

So, a tool that lets you handle all this and more, without having you get your hands dirty in coding can help in going a long way on the DevOps path. Puppet is one such tool. Simply put, Puppet lets you manage the state of your IT infrastructure. In an agent/master setup, the master is the central Puppet server, from which all of your configuration data will be managed and distributed, and all your remaining servers will be Puppet agent nodes, which can be configured by the Puppet master server. In standalone Puppet, every node periodically uses the **puppet apply** command to compile and apply its own configuration, using a full set of Puppet modules and manifests. In other words, each node requires the same files and data that the Puppet master requires in agent/master Puppet. In this tutorial we will focus on the agent/master Puppet setup.

Declarative language

Puppet uses a declarative language to help you describe the state of the system. This means you don't have to think about building a LAMP stack by installing Apache, MySQL and PHP, then tweaking php.ini, then enabling mod_php, and so on. You can just describe the state you want – for example, "a web server with Apache, MySQL and Wordpress" – and then let Puppet make it happen.

01 Pre-installation steps

Once you decide on the setup type, ie agent/master or standalone, the next step is to identify the master (if you are planning to use the agent/master setup). Note that a Windows machine can't be a master. Also, the master should be a dedicated machine with a minimum of two processor cores and at least 1GB RAM. It must also be reachable at a reliable hostname. You can reduce the setup time on your agents by making sure the master is available at the default hostname of **puppet**. The Puppet master should allow incoming traffic on port 8140 and agent nodes should be able to connect to the master. Puppet master serves as the certificate authority and so it should have its system time set up accurately, or it may mistakenly issue agent certificates from the distant past or future, which other nodes will treat as expired. Once all these are taken care of, you can move on to the next step of Puppet installation.

02 Installation on the master

Time to install Puppet on the master. First enable the Puppet Labs package repository; once this repository is enabled, we can install Puppet. Type:

```
$ wget https://apt.puppetlabs.com/puppetlabs-release-trusty.deb
$ sudo dpkg -i puppetlabs-release-trusty.deb
$ sudo apt-get update
```

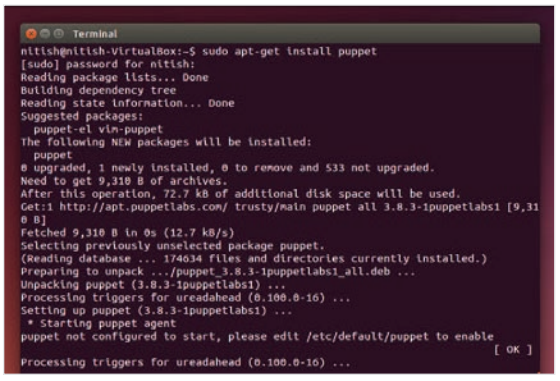
Next, on the Puppet master node, type either this:

```
$ sudo apt-get install puppetmaster-passenger
```

... which is recommended by PuppetLabs itself, as it installs Puppet and its prerequisites, and automatically configures a production-capacity web server; or instead, type in:

```
$ sudo apt-get install puppetmaster
```

... which will only install Puppet, its prerequisites and an init script (/etc/init.d/puppetmaster) for running a test-quality Puppet master server.



03 Installation on agents and post-installation

On all the agent nodes, you now need to run the following command:

```
$ sudo apt-get install puppet
```

This will install Puppet and an init script (/etc/init.d/puppet) for running the Puppet agent daemon. With this, we complete the installation process but Puppet is not started yet. Let's now configure the master and the agents, so they can talk to each other as and when required. The first step in this process is to set the master's name and certificates. You will need a DNS server that can resolve the names to corresponding IP addresses, then in the main section of the master's puppet.conf file, set the dns_alt_names setting to a comma separated value of each hostname that master can have. The default name to which an agent will try to connect is **puppet**. So, if you use "puppet" as the hostname for master, it will save you some effort. This is one of the most important steps and should be done carefully to avoid communication issues at later stages.

04 Activate certificates

The next step is to activate the certificates on the master. If this is the only Puppet master in your deployment, or if it will be acting as the CA server for a multi-master site, you should now run the following:

```
$ sudo puppet master --verbose --no-daemonize
```

This will create the CA certificate and the Puppet master certificate, with the appropriate DNS names included. As soon as the terminal output displays:

```
Notice: Starting Puppet master version 3.8.3,
```

... kill the process by typing Ctrl+C, because we don't want the master to run. Now put all the Puppet manifests and modules in place and then configure a web server if you are planning to use Puppet to manage production loads.

05 Configure the agent

First of all, you will need to configure the puppet.conf settings in each of the agent nodes so that they are able to connect to your puppet master server. Also, certain environment-specific factors should be changed according to your infrastructure requirements. Then you will need to start the Puppet agent service. You can either configure it to start on boot or set up a cron job, so that it runs at specific intervals. To start the Puppet service, type:

```
$ sudo puppet resource service <NAME>
ensure=running enable=true
```

Then configure the cron job using:

```
$ sudo puppet resource cron puppet-agent
ensure=present user=root minute=30 command='/usr/
bin/puppet agent --onetime --no-daemonize --splay'
```

The final step is to sign the certificates for each of the agent nodes. Note that in an agent/master deployment, an admin must approve a certificate request for each agent node before that node can fetch configurations. Agent nodes will request certificates the first time they attempt to run. To approve the requests, log on to the Puppet master server and view outstanding requests using:

```
$ sudo puppet cert list
```

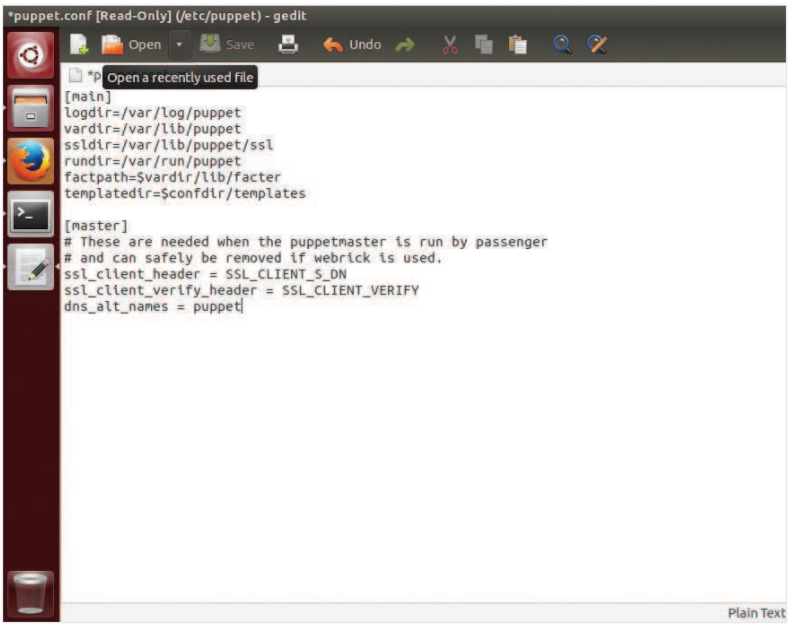
Then sign a request by typing:

```
$ sudo puppet cert sign <NAME>
```

You can also sign all the requests in one go using:

```
$ sudo puppet cert sign --all
```

With this, the agent nodes are now ready to communicate with the master. Puppet also enables a higher level of configuration by letting you classify the nodes. With this, you can determine which agent nodes receive which information from the master.



Above Most of the configuration settings listed at **bit.ly/1GYjvl7** can be set in puppet.conf

Run when file updates

You can run a command automatically whenever a file changes. Just use an exec resource with refreshonly set to true – for example:

```
file { "/etc/bind":
source => "/dist/apps/
bind" }

exec { "/usr/bin/ndc
reload":

subscribe   => File["/
etc/bind"],

refreshonly => true

}
```

MANAGE YOUR VIRTUAL MACHINES WITH VAGRANT

Vagrant makes it easy for you to manage several virtual machines in one go. Here we will learn how to install and get started with Vagrant

Before diving further into what Vagrant can do for you, let's first understand what Vagrant is. In simple terms, Vagrant provides an easily configurable, reproducible and portable work environment, built on top of virtualisation technology. The work environment built by Vagrant can be controlled by a single consistent workflow in order to help maximise the productivity and flexibility.

To achieve its magic, Vagrant stands on the shoulders of giants like VirtualBox, VMware, AWS, or other providers. You can then use provisioning tools – such as shell scripts, Chef or Puppet – to automatically install and configure software on the machine. So, once you create a single Vagrantfile, you just need to **vagrant up** and everything is installed and configured for you to work with. Other members of your team can create their development environments from the same configuration, so whether you're working on Linux, OS X or Windows, all your team members are running code in the same environment, against the same dependencies, all configured the same way.

Installing Vagrant

Vagrant can be easily downloaded and installed on your system. Go to the Vagrant downloads page (**vagrantup.com/downloads.html**) and download the package appropriate for your system. You can then install it by following the standard installation procedure for your platform. Also, the installer automatically adds the **vagrant** command to your system path, so you can start using it now. If you see the "command not found" error, just log out and back in, and it should work.

Vagrantfile

A Vagrantfile describes the type of machine required for a project, and how to configure and provision these machines. Note that the actual filename for the file is simply Vagrantfile. Vagrant runs with one Vagrantfile per project, and the Vagrantfile is supposed to be committed to version control. This enables other developers involved in the project to check out the code, run **vagrant up** and be on their way. Vagrantfiles are portable across every platform Vagrant supports.

Environment variables

You can use environment variables in Vagrantfile, by calling ENV['variable_name']. For example:

```
config.vm.box =
"#{ENV['boxname']}"
```

VAGRANTFILE EXPLAINED

```
File  Edit  View  Search  Terminal  Help
# -*- mode: ruby -*-
# vi: set ft=ruby :
# Vagrantfile API/syntax version.

VAGRANTFILE_API_VERSION = "2"

#Check if you have the good Vagrant version to use docker provider...
Vagrant.require_version ">= 1.6.0"
Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|

# All Vagrant configuration is done here. The most common configuration
# options are documented and commented below. For a complete reference,
# please see the online documentation at vagrantup.com.
ENV['VAGRANT_DEFAULT_PROVIDER'] = 'docker'

# Share an additional folder to the guest VM. The first argument is
# the path on the host to the actual folder. The second argument is
# the path on the guest to mount the folder. And the optional third
# argument is a set of non-required options.
# config.vm.synced_folder "../data", "/vagrant_data"
# Provider-specific configuration so you can fine-tune various
# backing providers for Vagrant. These expose provider-specific options.
# Example for VirtualBox:
#
config.vm.provider "docker" do |d|
d.build_dir = "."
d.has_ssh = true
end
config.ssh.port = 22
end
~
~
~
~
"Vagrantfile" 31 lines, 1079 characters
```

VAGRANTFILE VERSION CHECK

Docker is only supported on Vagrant version 1.6 and above, so before anything else, you need to confirm if the installed Vagrant has the proper revision. You can do this easily, as shown here.

VAGRANT DEFAULT PROVIDER

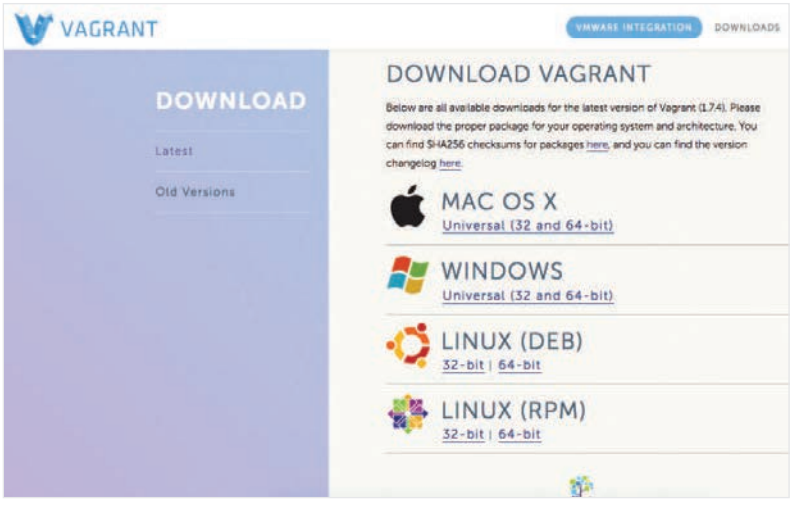
Setting up the Vagrant default provider saves having to specify at every Vagrant command that the provider is Docker. The rest of the file has options that Vagrant will use to build the Docker image and run a container.

PROVIDER-SPECIFIC CONFIGURATION

Once the provider is set, make configurations specific to the provider. This is done in Vagrantfile using a loop. This way, multiple config.vm.provider blocks can exist to configure multiple providers.

SSH SETTINGS

The settings within config.ssh set how Vagrant will access your machine over SSH. The default is fine, and even if this line were not added, SSH would have used its default port 22. But, this shows you can change the port.



Above Installation is simple – just head to the downloads page and pick the one for your system

Vagrantfiles follow Ruby syntax, but knowledge of Ruby is not necessary in order to make modifications to the Vagrantfile, since it is mostly simple variable assignment.

Note that when you run a **vagrant** command, Vagrant climbs up the directory tree looking for the first Vagrantfile it can find, starting first in the current directory. So if you run **vagrant up** in /home/user/projects/foo, it will search the following paths, in order, for a Vagrantfile until it finds one:

```
/home/user/projects/foo/Vagrantfile
/home/user/projects/Vagrantfile
/home/user/Vagrantfile
/home/Vagrantfile
/Vagrantfile
```

Boxes

Instead of building a virtual machine from scratch, which generally takes lot of time and effort, Vagrant uses a base image to quickly clone a virtual machine. These base images are known as boxes in Vagrant, and specifying the box to use for your Vagrant environment is always the first step while you are creating a Vagrantfile. Boxes are added to Vagrant with:

```
$ vagrant box add hashicorp/precise32
```

This stores the box under a specific name so that multiple Vagrant environments can re-use it. Note that HashiCorp is the company behind Vagrant, and the above command will download the box called "hashicorp/precise32" from HashiCorp's box catalog. Once a box has been added to Vagrant, you can configure a project to use it as a base. Open the Vagrantfile and change the contents to the following:

```
Vagrant.configure("2") do |config|
config.vm.box = "hashicorp/precise32"
end
```

You can search through HashiCorp's catalogue of Vagrant boxes at: **atlas.hashicorp.com/boxes/search**.

Port forwarding

In some cases, you may want to access a port on your host computer and have your request forwarded to a port on the guest virtual machine. Port forwarding can be set in Vagrant by adding one line to the Vagrantfile.

In the following example, accessing **localhost:8080** will access port 80 on the guest machine, and **localhost:8443** will access port 443 on the guest machine:

```
config.vm.network
:forwarded_port,
guest: 80, host: 8080
```

```
config.vm.network
:forwarded_port,
guest: 443, host:
8443
```

SET UP DOCKER CONTAINERS

Vagrant provides a method for creating repeatable development environments across a range of operating systems. It uses providers to spin up isolated virtual environments. The default provider is VirtualBox, but since Vagrant 1.6, Docker-based development environments have been supported too. Compared to other tools like boot2docker, which can help run Docker on non-Linux platforms, Vagrant has some important advantages:

- Configure it once and run it everywhere – Vagrant is just a Docker wrapper on systems that support Docker natively, while it spins up a 'host VM' to run containers on systems that don't support it. Users don't have to bother themselves with whether Docker is supported natively or not; the same configuration will work on every OS.
- Docker hosts are not limited to boot2docker (a VirtualBox image of Tiny Core Linux) – Debian, Ubuntu, CoreOS and other Linux distros are supported too.
- Vagrant can orchestrate Docker containers. This means you can run multiple containers concurrently and link them together.

A Vagrant box for boot2docker is available on the default box catalog. You can add that Vagrant box to your system by simply running the following:

```
$ vagrant box add mitchellh/boot2docker
```

Vagrant will prompt for the provider and then download the appropriate version of the box for the selected provider. Then, unless you tell it otherwise, Vagrant will spin up an instance of this boot2docker box anytime it needs to perform Docker provider operations. If you haven't already added the Vagrant box, the first time you use the Docker provider, it will download the box automatically. This is the easy way to set up Docker with Vagrant.

But what if you don't want to use boot2docker? What if you'd rather use Ubuntu or CentOS, so that what you do in Vagrant more closely matches what you might do in the data centre?

You can change this behaviour by adding a line in the Vagrantfile that spins up your Docker containers. The specific line would look something like this:

```
docker.vagrant_vagrantfile="path/to/host/VM/
Vagrantfile"
```

So, you need to create a Vagrantfile that spins up a VM running the Linux distribution of your choice, and then reference that Vagrantfile in the one that creates your Docker containers. Of course, since Vagrant expects the filename to be Vagrantfile, the file defining the host VM must be in a different directory to the file defining the Docker containers. So, you need to specify the path to the host VM Vagrantfile.

Although Vagrant expects Docker to be running inside this host VM, it does provide a way to simplify that by allowing you to provision (install) Docker into the VM when you run **vagrant up** (much in the same way Vagrant can provision other things into a VM when it is first instantiated). This is done with a simple command in the host VM Vagrantfile:

```
config.vm.provision "docker"
```

SAVE UP TO 50% ON A GIFT SUBSCRIPTION THIS CHRISTMAS*

JUST £29.99

10 ISSUES, SAVE 50%

GADGET
Packed with the latest, coolest and most exciting tech
12 issues, save 50%

HOW IT WORKS
The action-packed science and technology magazine
13 issues, save 46%

ALL ABOUT HISTORY
Bringing history to life for the whole family
13 issues, save 49%

HISTORY OF WAR
The stories, strategies, heroes and machines of historic conflicts
12 issues, save 50%

DIGITAL PHOTOGRAPHER
Inspiration, tutorials and tools for enthusiasts and professionals
12 issues, save 50%

RETRO GAMER
The number one magazine for classic gaming
12 issues, save 50%

ALL ABOUT SPACE
Discover the wonders of the solar system and beyond
13 issues, save 49%

SCIFINOW
The number one magazine for sci-fi, fantasy and horror fans
12 issues, save 50%

ORDER HOTLINE 0844 856 0644**
ONLINE AT www.imaginesubs.co.uk/xmas151

SUBSCRIBE TO ANY MAGAZINE FOR JUST £29.99

- Exclusive **Christmas offer**
- Save **up to 50%** on the shop price*
- Never miss an issue of your **favourite magazine**
- **Free delivery**, direct to your door
- **Ideal Christmas gift** - that lasts all year!
- **Free e-card** to send when you buy as a gift

BUY AS A GIFT OR TREAT YOURSELF!

Use code XMAS151 for this extra-special price.

Order by 2nd December to start your subscription the first issue after Christmas.

** Calls will cost 7p per minute plus your telephone company's access charge.

BY POST

Send your completed form to:
Imagine Subscriptions, 800 Guillat Avenue, Kent Science Park, Sittingbourne, Kent ME9 8GU

YOUR DETAILS

Title_____First name_____
Surname_____
Address_____
_____
_____
Postcode_____Country_____
Telephone number_____
Mobile number_____
Email address_____
This subscription is ☐ for me ☐ a gift

DELIVERY DETAILS (IF DIFFERENT FROM ABOVE)

Title_____First name_____
Surname_____
Address_____
_____
_____
Postcode_____Country_____
Telephone number_____

PAYMENT DETAILS

UK £29.99 Magazine name_____

FOR MULTIPLE ORDERS PLEASE CALL THE ORDER HOTLINE **0844 856 0644****

CHEQUE

☐ I enclose a cheque for £_____
(made payable to Imagine Publishing Ltd)

CREDIT/DEBIT CARD

☐ Visa ☐ Mastercard ☐ Amex ☐ Maestro

Card number ☐☐ ☐☐ ☐☐ ☐☐ ☐☐ ☐☐ ☐☐ ☐☐

Expiry date ☐☐ ☐☐

Issue number (Maestro) ☐☐

Signed _____

Date _____

All subscribers receive a monthly e-newsletter of news and offers.

Please tick if you would prefer not to receive any promotional material from Imagine Publishing:

by post ☐ by telephone ☐ by email ☐

Please tick if you would prefer not to receive any promotional material from other companies

by post ☐ by telephone ☐ by email ☐

Please tick if you DO wish to receive such information by email ☐

TERMS & CONDITIONS

* Titles are priced at £29.99 saving up to 50% on the shop price. For the full list of titles and issue details please visit **www.imaginesubs.co.uk/XMAS151.** This is a UK-only offer, for overseas offers please visit **www.imaginesubs.co.uk/XMAS151** or call **+44 (0)1795 592 869.** Gift subscriptions ordered by 2nd December will begin with the first issue on sale January 2015 – orders after this date will start with the following issue. You have the option of sending a digital gift card so please include your email address in the form above. Non-gift subscriptions will start with the next available issue.
Offer ends 31st December 2015. Promo code XMAS151

Mihalis Tsoukalos
is a Unix administrator, a programmer (for Unix and iOS), a DBA and also a mathematician. He has been using Linux since 1993

Resources
Text editor
GCC compiler
Perl interpreter
Go compiler
Haskell compiler

Systems programming: Files and directories

Find out how to work with files, directories and symbolic links using system calls to write tools

This tutorial will present the system calls and explain the techniques that will enable you to deal with files and directories in Linux. The **stat** family of functions and the stat structure that holds all important data are the centre of this tutorial. The majority of the other system calls here enable you to change most of the attributes kept in the stat structure. Although the stat family has four functions; **stat**, **fstat**, **lstat** and **fstatat**, only the first three will be explained here. However, all of them return the same information, which is a stat structure, and have the same man page, which should be the first and the last place you will need to look for additional information.

As always, most topics come with the relevant C code that clarifies the tricky parts of each individual concept. Nevertheless, you should try to experiment and write your own code. The best way to do so is by trying to implement simple versions of existing Linux command line utilities.

In addition to the expected C code, you are going to see code in Perl, Go and Haskell; a scripting language, a general purpose programming language and a functional programming language, respectively.

Before continuing, you should always remember that when you talk about files in UNIX or Linux, it could mean everything from a network device, a network socket, a whole hard drive or a printer to a plain text file.

The stat() system call

The following code shows a simple program (fstatEx.c) that uses both **stat()** and **fstat()** and gives you information about files and directories. The main difference between **stat()** and **fstat()** is that the former needs a pathname as input whereas the latter wants a file descriptor as input. Therefore, **fstat()** is more appropriate when you want to examine a file that is already open. The stat functions are mostly used by the **ls** utility.

```
#include <stdio.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
int main(int argc, char** argv)
{
```

A little Perl

It would be useful to know how Perl deals with files and directories. Download and inspect find.pl, which shows the implementation of find.c in Perl. As usual, the Perl version uses fewer lines of code than C to do the same task.

As Perl has many modules, it is recommended to use one when it helps you do your job. Nevertheless, this implementation does not use any Perl modules in order to be closer to the C implementation.

Each time find.pl discovers a new directory, the directory is added to an array in order to process it later on – a very clever approach, indeed. The program matches the pattern, which is the second command line parameter of the program, against the current filename. If it is a match then the filename is printed; otherwise, the program goes to the next file.

At line 21, the program opens a directory for reading, reads all the contents of the directory at once and stores them in an array. Then (line 22), it closes the directory before processing the contents of it using a handy **foreach** loop (line 23). Line 33 makes the program exclude the current directory (.) and its parent directory (..) from the output using a regular expression. Line 35 matches a filename against the given pattern. If it is a match, it continues by printing the actual number of blocks allocated for the file; otherwise, it goes to the next iteration of the **foreach** loop and processes the next file in the **@files** array. Line 36 shows that Perl uses the stat structure as an array.

```
    if (argc != 3)
    {
        printf("Please use exactly two arguments!\n");
        return -1;
    }
  char *file1 = argv[1];
  char *file2 = argv[2];
  int fd = open(file1, O_RDONLY);

  // Open the second file
  if (fd < 0)
  {
    printf("Error reading %s\n", file2);
    return -1;
  }

  // Get file permissions using stat()
  struct stat fileStat;
    if(stat(file2, &fileStat) < 0)
        return 1;

  printf("The file permissions of %s are:\t", file1);
  printf( (S_ISDIR(fileStat.st_mode)) ? "d" : "-" );
  printf( (fileStat.st_mode & S_IRUSR) ? "r" : "-" );
  printf( (fileStat.st_mode & S_IWUSR) ? "w" : "-" );
  printf( (fileStat.st_mode & S_IXUSR) ? "x" : "-" );
  printf( (fileStat.st_mode & S_IRGRP) ? "r" : "-" );
  printf( (fileStat.st_mode & S_IWGRP) ? "w" : "-" );
  printf( (fileStat.st_mode & S_IXGRP) ? "x" : "-" );
  printf( (fileStat.st_mode & S_IROTH) ? "r" : "-" );
  printf( (fileStat.st_mode & S_IWOTH) ? "w" : "-" );
  printf( (fileStat.st_mode & S_IXOTH) ? "x" : "-" );
  printf("\n");
```

```
struct stat {
  dev_t   st_dev;     /* ID of device containing file */
  ino_t   st_ino;     /* inode number */
  mode_t  st_mode;    /* protection */
  nlink_t st_nlink;   /* number of hard links */
  uid_t   st_uid;     /* user ID of owner */
  gid_t   st_gid;     /* group ID of owner */
  dev_t   st_rdev;    /* device ID (if special file) */
  off_t   st_size;    /* total size, in bytes */
  blksize_t st_blksize; /* blocksize for filesystem I/O */
  blkcnt_t  st_blocks;  /* number of 512B blocks allocated */

  /* Since Linux 2.6, the kernel supports nanosecond
    precision for the following timestamp fields.
    For the details before Linux 2.6, see NOTES. */

  struct timespec st_atim;  /* time of last access */
  struct timespec st_mtim;  /* time of last modification */
  struct timespec st_ctim;  /* time of last status change */

#define st_atime st_atim.tv_sec    /* Backward compatibility */
#define st_mtime st_mtim.tv_sec
#define st_ctime st_ctim.tv_sec
};
```

Note: the order of fields in the stat structure varies somewhat across architectures.

Left The information returned by the stat family should look like this

```
  // Get inode number using fstat()
  struct stat anotherFileStat;
    if(fstat(fd, &anotherFileStat) < 0)
        return -1;
  printf("File %s inode number is \t%ld\n", file2,
anotherFileStat.st_ino);

  // Close file2
  close(fd);
  return 0;
}
```

For the fstatEx.c program, the **stat()** system call returns the file permissions whereas the **fstat()** system call returns the inode number. An inode is a data structure on a filesystem that keeps the information about a file except its name and data. Directories are implemented as lists of names assigned to inodes. Therefore, a directory contains an entry for itself, its parent directory and each child, which can be files or directories. Please don't forget the fact that inodes do not contain filenames; only other file metadata.

The top image shows the stat structure that holds the information returned by the stat family. Although the actual definition of the stat structure might differ among UNIX variants, most of the presented information should be there.

The fstatEx.c file is the first place to go when you want to remember how **stat()** works. You can verify the results of fstatEx.c using the **ls** command. Converting the information you get from **stat()** into the octal format is not a trivial task, although you only have to write this code once. On the other hand, getting the inode number is straightforward.

The umask() function

The **umask()** function sets the file creation mask that is connected with every process. The file creation mask comes into play every time a process is going to create a new file or a new directory.

The main usage of **umask()** is creating temporary or permanent files that cannot be read or written by mistake. Normally, you do not have to deal with the **umask** value unless you want to use a different **umask** value than the one read from the shell. Changing the **umask** of a process does not affect the **umask** value of its parent (usually the shell).

The chmod() function

The **chmod(path, mode)** function sets the file permission of the file specified by the pathname path to mode. The **fchmod()** system call does the same using a file descriptor instead of a path. Typically, you will not need to use any of these functions unless you are writing software that deals with security issues. As it is a bit tricky to use **chmod()**, the following C code should make things easier to understand:

```
char mode[]="0777";
int i;
i = strtol(mode, 0, 8);
if (chmod (file, i) < 0)
    printf("Error using chmod");
```

Define the desired permissions as a string. Then use **strtol()** to convert the string into a long integer in the octal numeral system. Finally, use **chmod()** to set the desired permissions to the file. The mentioned C code can be found in fileTimes.c.

Symbolic links

The **lstat()** function gives information about symbolic links whereas **stat()** returns information about the file the link references. The only attributes returned from an **lstat()** call that refer to the symbolic link itself are the file type (**S_IFLNK**), size, blocks and link count.

Another issue is that if you want to read a symbolic link, you cannot do it with **open()** because **open()** follows the symbolic link. There are two system calls for opening the actual symbolic link: **readlink()** and **readlinkat()**.

Study the following code (countSymLinks.c) and the image below for more about dealing with symbolic links:

```
#include <unistd.h>
#include <stdio.h>
#include <sys/stat.h>
#include <dirent.h>
#include <string.h>

int totalSymbolicLinks = 0;
void printDir(char *dir)
{
  DIR *dp;
  struct dirent *entry;
  struct stat statbuf;

  if( (dp = opendir(dir)) == NULL )
  {
    fprintf(stderr,"cannot open directory: %s\n", dir);
    return;
  }
  chdir(dir);
  while((entry = readdir(dp)) != NULL)
```

```
iMac:code mtsouk$ diff countSymLinks.c find.c
7,9c7
< int totalSymbolicLinks = 0;
<
< void printDir(char *dir)
---
> void printDir(char *dir, int depth, char *desiredName)
13a12
>       int spaces = depth * 4;
28a28
>
30c30
<                       printf("%s/\n", entry->d_name);
---
>                       printf("%*s%s/\n",spaces,"",entry->d_name);
32c32
<                       printDir(entry->d_name);
---
>                       printDir(entry->d_name, depth+1, desiredName);
34,35c34,35
<               else if ( S_ISLNK(statbuf.st_mode) )
<                       totalSymbolicLinks++;
---
>               else if ( (strcmp(desiredName, entry->d_name) == 0) || (strcmp("anything",
desiredName) == 0) )
>                       printf("%*s%s\n",spaces,"", entry->d_name);
46c46
<       if ( argc != 2 )
```

Right Here we use the **diff** command to compare our countSymLinks.c and find.c files

> The main usage of **unmask()** is creating temporary or permanent files that cannot be read or written by mistake

```
  {
    lstat(entry->d_name, &statbuf);
    if ( S_ISDIR(statbuf.st_mode) )
    {
      // Ignore . and ..
      if (strcmp(".", entry->d_name) == 0 ||
strcmp("..", entry->d_name) == 0)
        continue;
      // Print the current directory
      printf("%s/\n", entry->d_name);
      /* Recurse at a new indent level */
      printDir(entry->d_name);
    }
    else if ( S_ISLNK(statbuf.st_mode) )
      totalSymbolicLinks++;
    else
      continue;
  }
  chdir("..");
  closedir(dp);
}

int main(int argc, char** argv)
{
  char *rootDir = "";
   if ( argc != 2 )
  {
    printf("Please provide exactly 1 command line
argument.\n");
    return -1;
  }
  rootDir = argv[1];

  printf("Scanning %s for symbolic links!\n", rootDir);
  printDir(rootDir);
  printf("Found %d Symbolic Links!\n",
totalSymbolicLinks);
  return 0;
}
```

The various times of a file

Each file has three time fields: the modification time (**st_mtim**), the changed-status time (**st_ctim**) and last-access time (**st_atim**). There is a subtle difference between the modification time and the changed-status time. The modification time indicates a change to the data of the file whereas the changed-status time indicates a modification to the metadata of the file, which then results in a modification to the inode of the file.

Systems programming in Haskell

Haskell is a functional and compiled programming language that can be used for systems programming. Haskell is a perfect choice for performing filename matching operations. The only problem with the Haskell way is that different operating systems use different and specialised modules.

The **System.Directory** module contains many functions that can be used for getting information from the filesystem, including getting a list of files in a directory, renaming or deleting files, copying files, changing the current working directory (**setCurrentDirectory**) and creating new directories. The good news is that **System.Directory** is portable and works on any platform where GHC (Glasgow Haskell Compiler) itself works. The following output shows the advantages of using Haskell:

```
> :module System.Directory
> getDirectoryContents "/tmp"
["update-cache-2a113cac","upload_progress_
cache",".","update-extraction-2a113cac","drush-
env",".font-unix",".htaccess","ghc14750_0",".XIM-
unix","..","hsperfdata_root",".ICE-unix",".X11-unix",".
Test-unix"]
> getDirectoryContents "/tmp" >>= return . filter
(`notElem` [".", ".."])
["update-cache-2a113cac","upload_progress_
cache","update-extraction-2a113cac","drush-env",".font-
unix",".htaccess","ghc14750_0",".XIM-unix","hsperfdata_
root",".ICE-unix",".X11-unix",".Test-unix"]
```

The first **getDirectoryContents** command returns all the contents of the /tmp directory. The second command does the same with a little difference: it excludes the special values of "." and ".." using pattern matching.

Depending on the function used to access a file, some or all of the time fields can change. Also, modifying, deleting or adding a directory entry will most certainly change some or all of the times of the parent directory.

Now let's look at fileTimes.c. You will need to translate the time values you get into a format that makes more sense to users – this is done using the **ctime()** function. As usual, the first thing you should do is calling either **stat()** or **fstat()** to get the necessary data. After getting the data you should extract it and print it on screen, which is a direct and painless process. As you already learned, you then change the file permissions of the text file using the **chmod()** function. Last, you append a little text to the file. Compile and execute the program to see how the various times of a file change.

Working with directories

Underneath the surface, UNIX machines implement and treat directories as regular files; there exist only two differences between them. The first one is that the kernel does not allow writing on directories and the second one is that they have a special bit set in their inode.

Although knowing how to work with individual files is good, most of the time you are going to work with entire directories or directory trees. The two most useful system calls for operating with directories are **opendir()** and **readdir()**.

You can see both **opendir()** and **readdir()** in action by looking at the C code presented in find.c. Please note that

S_ISDIR(statbuf.st_mode) helps you to make sure that you are dealing with an actual directory instead of a regular file or a network card!

Traversing a directory structure

The purpose of the presented program (find.c) would be to traverse a directory structure in order to find files with a given filename – like a simplistic version of the find utility. If the desired filename is "anything" then find.c prints every file and directory as it finds it. It is always good to add some extra functionality to programs – this is implemented at line 34.

Download and inspect the find.c file. The Perl version that is presented in find.pl is more advanced in the sense that it uses a regular expression instead of exact matches; however, there are more similarities than differences between find.c and find.pl. As you might have guessed, the C version is bigger than the Perl version.

Please note that comparing strings for equality in C can be done either by comparing them character by character or by using the **strcmp(3)** function, which is a better solution. The main reason for having a separate function for printing the contents of a directory, which is named **printDir()**, is that the program operates recursively; as **main()** cannot call itself, you need a separate function!

The program uses the **chdir(2)** function to change the current working directory to the provided root directory (line 19). This is not mandatory but it makes things simpler for the programmer. Each time **printDir()** finds a directory entry, it visits that directory. When it finishes examining a directory, it goes back to the parent directory (line 39).

Please note that find.c uses **lstat()** instead of **stat()** in order to be able to skip symbolic links – this is not mandatory, but bear in mind that symbolic links are not very safe as they can even make your program jump to a different filesystem. Additionally, multiple symbolic links can create a loop that will make your program run forever! Fixing a symbolic link loop programmatically is not an easy task; therefore, using **lstat()** instead of **stat()** and performing the required checks is a safe and secure choice.

Calling C code from Haskell

As if Haskell was not good enough on its own, it can also call C code! Please note that Haskell is used as an example here as many other languages can also call C code.

The following program (pid.hs) gets the process id using C code wrapped in Haskell code:

```
{-# LANGUAGE ForeignFunctionInterface #-}
module Main where
import System.Posix.Types
foreign import ccall unsafe "getpid" c_getpid :: IO CPid
main = do
  print =<< c_getpid
```

Although you might not be familiar with the Haskell code, you should understand how extraordinary what just happened is: you used a C system call in a totally different programming language to get the information you wanted. Put simply, the **c_getpid** Haskell function uses **getpid(2)** to get the process ID. Using a slightly different approach, you can call your own C functions from Haskell code.

Counting symbolic links

countSymLinks.c implements another command line utility that searches a directory tree for symbolic links and counts them. You should already know how to traverse a directory structure from the code in find.c, which is going to be the base for countSymLinks.c. Two things should change in countSymLinks.c. The first is that it should use a global variable to keep the number of symbolic links found so far. The second is that it must use **S_ISLNK()** instead of **S_ISDIR()** to ensure the examined entry is indeed a symbolic link (line 34). Please check countSymLinks.c for the full C code.

The image on the previous page shows the output of the **diff** command that compares countSymLinks.c and find.c. As most of the desired functionality is already there, you only need to see the modifications. Although the use of **lstat()** is optional in find.c, this is not true for countSymLinks.c: the use of the **lstat()** system call in countSymLinks.c is required because, as you already know, **stat()** follows symbolic links!

Searching the PATH

Path.c shows the full C code of the path.c file that searches the PATH environment variable for an executable file like the **which** utility. You should be familiar with most of the presented code but it would be good to quickly review it.

First of all, the program reads the PATH variable and stores its contents. Then, it separates the various parts of the PATH variable – that are directories – to use each part independently. Last, it checks if the requested executable is located in one of the directories of the PATH environment variable. The program lists all instances of the executable found instead of just the first one.

The tricky thing with the PATH variable is that it may contain the current working directory, which is specified with a dot: ".". The **realpath(3)** function resolves the canonicalised absolute pathname for any given path. Therefore, the **opendir()** system call opens the canonicalised absolute pathname of every directory found in the PATH environment variable. Small details make a huge difference so always pay attention to them!

Please note that if the program you want to search for is in the current PATH but it is not an executable file, path.c shows it. This kind of checking happens with the help of the **access()** function at line 38. The **access(2)** function checks whether the calling process can access the specified file, which is its first argument; its second argument specifies the accessibility tests that are going to be performed. The **F_OK**

> "The tricky thing with the PATH variable is that it may contain the current working directory, which is specified with a dot"

value tests for the existence of the file, whereas the X_OK value tests whether the file has the execute permissions.

Remember that having the current working directory in your PATH environment variable might be a security risk.

Finding out the actual type of 'file'

The final part of the tutorial will present a handy tool that prints the file type of each one of its command line arguments. The implementation of filetype.c is fairly short and easy, mainly because the provided macros declared in the sys/stat.h header file greatly simplify the code. Just for your reference, the S_ISDIR() macro is defined as follows:

```
#define __S_ISTYPE(mode, mask) (((mode) & __S_
IFMT) == (mask))
#define S_ISDIR(mode) __S_ISTYPE((mode), __S_IFDIR)
```

The vital C code of filetype.c is the following **if/else** block:

```
if (S_ISREG(myStat.st_mode))
    printf("Regular file");
else if (S_ISDIR(myStat.st_mode))
    printf("Directory");
else if (S_ISCHR(myStat.st_mode))
    printf("Character special file");
else if (S_ISBLK(myStat.st_mode))
    printf("Block special file");
else if (S_ISFIFO(myStat.st_mode))
    printf("Pipe or FIFO file");
else if (S_ISSOCK(myStat.st_mode))
    printf("Socket");
else if (S_ISLNK(myStat.st_mode))
    printf("Symbolic Link");
else
    printf("Unknown file type!");
```

The S_ISREG() macro finds regular files; S_ISDIR() finds directories; S_ISCHR() discovers character special files; S_ISBLK() finds block special files; S_ISFIFO() finds if the examined file is a pipe or a FIFO; S_ISSOCK() finds sockets and the S_ISLNK() macro finds symbolic links.

Once again, the vital part of the C code is the stat structure because the lstat() system call is used for each command line argument, for reasons that should be clear by now.

The C code that processes all command line arguments one by one is the following:

```
for ( i = 1; i<argc; i++)
{
    if (lstat(argv[i], &myStat) < 0)
    {
        printf(" Error reading it with lstat()!\n");
        continue;
    }
}
```

This is a common technique for processing multiple command line arguments that do not contain any options. The reason for starting with argv[1] is that argv[0] always holds the name of the executable file. The **continue** keyword causes control to pass to the end of the enclosing **for** statement and makes the program to process the next file.

Look at filetype.c for the complete implementation. Until you implement a task, you don't know if it is simple or difficult. Also, most of the code of Linux command line utilities is about error checking, command line argument processing and exceptions – filetype.c did not do too much error checking but when you are writing production code, you cannot get away with dealing with error checking and abnormal situations.

Processing the output of filetype.c with AWK gives interesting summary data about the /dev directory:

```
$ ./filetype /dev/* | awk -F: '{print $2}' | sort
| uniq -c | sort -rn
  113  Character special file.
   43  Block special file.
   11  Directory.
    7  Symbolic Link.
    1  Pipe or FIFO file.
```

Similarly, examining the contents of the /tmp directory gives the next output:

```
$ ./filetype /tmp/* | awk -F: '{print $2}' | sort
| uniq -c | sort -rn
   77  Directory.
   10   Pipe or FIFO file.
    6  Regular file.
    3   Socket.
```

Once again, you see that the true power of UNIX command lines utilities comes when you are combining them to create more complex commands.

Use Go for files and directories

As it is always good to know that you have many alternatives for doing systems programming, another example in a different programming language will be presented; this time it will be Go. Go has many packages that can deal with files and directories that will make you write less code, which also means fewer bugs.

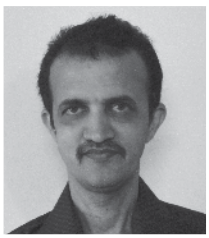
The presented program (walk.go) implements the core functionality of find.c – which is traversing a directory structure and visiting all its directories and files.

The logic is more or less the same as in find.c. It is interesting to notice that there are many lines of code for error checking – when writing systems software it is truly important that the programmer is checking the return values of all functions and making sure that everything is working as expected before continuing. The **filepath.Walk(root string, walkFn WalkFunc)** function walks a file tree by considering its root to be root (the first variable) and by calling the **walkFn** for each file or directory in the tree, including root. The **walkFn** does the necessary error checking before printing the name of the visiting file or directory – you can see that the Go approach is very convenient and makes the job of the programmer easier. If the **walkFn** function returns **SkipDir** when invoked on a directory (line 23), Walk skips the contents of the directory entirely.

You can try to improve the program by getting the root directory as a command line argument and by implementing the full functionality of find.c. Go code will make you feel right at home because it looks a lot like C code.

Compile software with the GNU Compiler Collection

Get a handle on the GCC essentials and learn to work with its powerful error and warning flags



Swayam Prakasha
has a master's degree in computer engineering. He has been working in IT for years, focusing on areas like operating systems, network security and electronic commerce

Resources
GCC
bit.ly/1QsGYe9

When we are developing software in the Linux environment, we often use C or C++ as the programming languages. The programs we write are normally referred to as the source code, and there is always a need to convert these source files into another computer language. Compilation is the process of converting the source code into a target language having a binary form, the purpose of which is to create an executable program. Compilation is often referred to as the translation of the source code from a high-level programming language to a lower-level language. A compiler is a specialised program that converts source code into machine language (also called object code or machine code) so that it can be understood directly by a CPU (central processing unit). An excellent C compiler is included in the GNU Compiler Collection (GCC), one of the most important components of most modern Linux distributions. Consequently, GCC is considered the standard C compiler for the Linux environment, and is also used as a standard C++ compiler. Please note that GCC is portable and runs on many operating platforms, and is currently available on all Unixes. For more details on GCC, you can visit **gcc.gnu.org**. By typing **man gcc** at the console, you can obtain all the related information about this popular compiler.

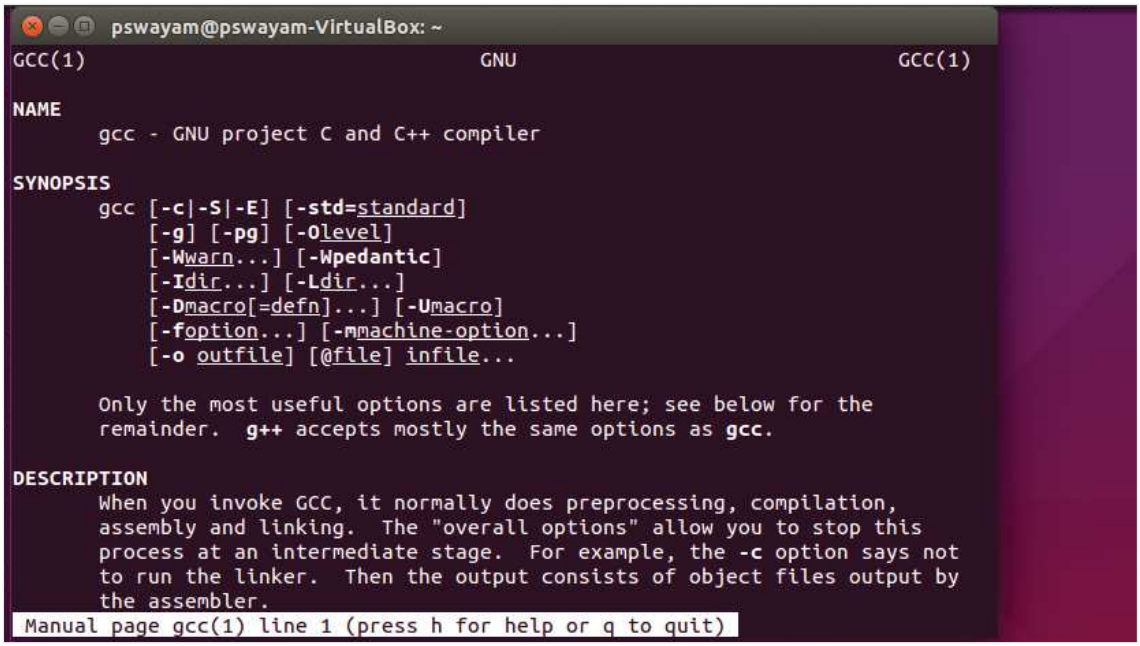
The easiest way to install GCC is to install a package made for your operating system. Note that all GNU/Linux distributions include packages for GCC. Once you have GCC on your Linux box, you can find out its version by using the **gcc --version** command. More details can also be obtained via the **-v** option.

Let us now see how you can use GCC to compile a program written in the C language. Using any one of your favourite text or code editors, type up the following program and then save it as 'sample_prog1.c'.

```
#include <stdio.h>
main()
{

printf("This is a sample program to understand GCC\n");

}
```

In its simplest form, GCC can be used like this:

```
gcc sample_prog1.c
```



Right The man page for GCC explains the various options that can be used with it

As is to be expected, that command executes the entire compilation process and generates an executable by the name of 'a.out'.

If you want to run the program, you need to execute the following command:

```
./a.out
```

This will print the message "This is a sample program to understand GCC" on your screen. We have now successfully complied the C program.

If you want to have a specific output file name for the generated executable, you can use the **-o** option as shown:

```
gcc -o sample_prog1 sample_prog1.c
```

Here, instead of 'a.out', the executable generated will be 'sample_prog1'. In other words, we can see that the **-o** option sets the output filename for GCC.

Another important option that developers normally use with GCC is **-Wall**. When we compile a program, the compiler will also give information about the warnings that are encountered in the compilation process. These warnings are controlled by the **-W** option of GCC. There is a way to enable all of the common warnings by passing **-Wall** as one parameter to GCC. For our earlier sample program, sample_prog1.c, we can use the following command where we have passed **-Wall** as one parameter.

```
gcc -Wall -o sample_prog1 sample_prog1.c
```

Please note here that even though there were warnings, GCC was able to compile our program. Only when there are errors will GCC abort the compilation.

It is important to also note that GCC has a variety of specific warning flags, some of which are enabled by default and all of which have negative forms. The **-Wall** flag will turn almost all of the warning flags, but the **-Wextra** flag enables even more warning condition flags. In order to understand which warning flags are enabled by **-Wextra**, it is worth running **man gcc** and having a quick read through.

Let us consider the following program (save this with the name 'sample_prog2.c').

```
#include <stdio.h>
int main(void)
{
printf("This is the second sample program to
understand GCC options\n");
}
```

When we try to compile the above program using the **-Wall** option, we will get the following warning:

```
sample_prog2.c: In function 'main':
sample_prog2.c:5:1: warning: control reaches end of
non-void function
```

If we look at the above example, we can note here that even though the main function is declared to return an integer value, the return value is undefined. And our compiler immediately displays a warning corresponding to this. This problem can be fixed by a developer using the following method (sample_prog3.c):

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
printf("This is the third sample program to
understand GCC\n");
return EXIT_SUCCESS;
}
```

When we compile the above program, we will not have to worry about any warnings.

Another useful option with GCC is **-Werror**, as this causes GCC to treat all warnings as errors. When we use **-Werror**, GCC will not finish the compilation if any warning is detected during the process. Whenever we are dealing with large projects, it is common to use this option and having the compilation aborted in this manner can be beneficial. We can use **-Werror** in the following way:

```
gcc -Wall -Werror -o sample_prog2 sample_prog2.c
```

Warnings generated by the compiler when we set the **-Werror** option may not be very serious ones, but as you write complex programs, this option can be extremely useful for tracking down the bugs. Consequently, it is good practice to include this option in your process.

Another useful feature with the modern development system is the availability of some of the powerful debugging tools. With the help of these debugging tools, a developer will have powerful ways of tracing the execution of their programs. And with this, you will be able to isolate the annoying problems and then take corrective measures to fix the issue. GNU Debugger (GDB) is one such powerful debugging tool.

Before we use GDB effectively in our programs, we need to compile our programs with debugging symbols. When we do this, GCC inserts extra information to the object files and executable files that it generates. This extra information will enable GDB to determine the relationship between the compiled code and the lines of code in the source program.

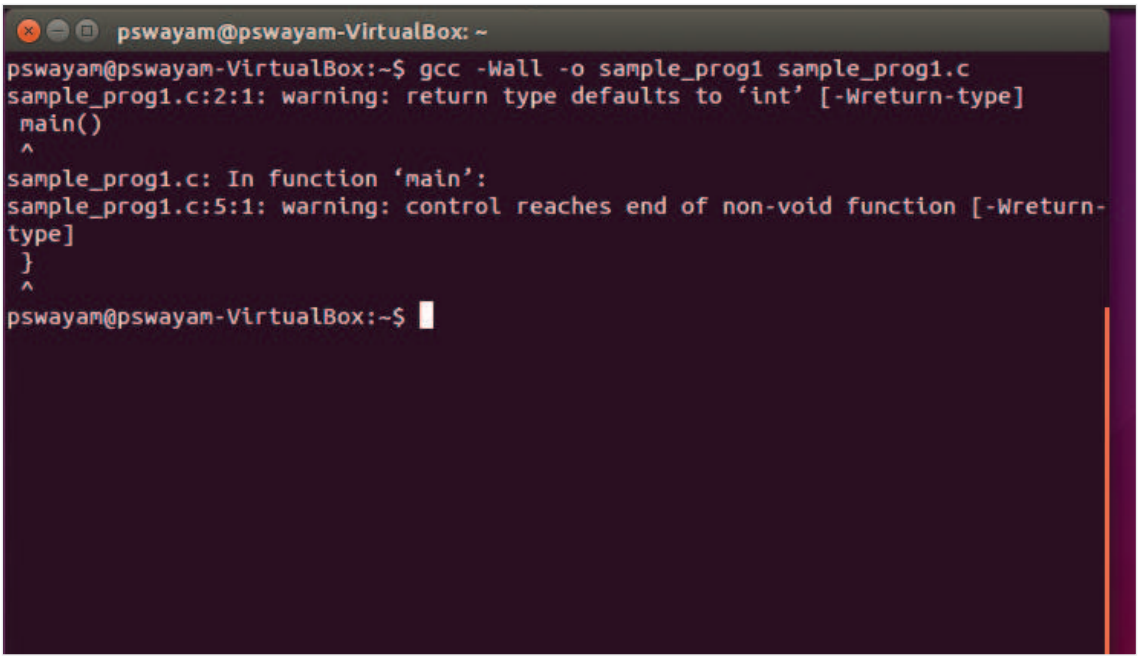
Please note that debugging symbols are not compiled into your programs by default. When we have the debugging symbols, they increase the size of the executables. Also note that in order to generate the debugging symbols, we need to use the **-g** option with GCC. In its simplest form, it generates a default set of debugging options and they will normally be enough for the debugging purpose. This can be done by executing the following command:

```
gcc -g -Wall -o sample_prog2 sample_prog2.c
```

It is also possible to enable more debugging information, and that can be useful in some scenarios. If we are going to use the GDB debugger then we may need to execute a command similar to the following:

Save time with GCC

The most common use of GCC is when there are multiple program files to be compiled. If we compile each and every source file separately then the time required to build the executable is significant. However, if we compile all the files at once, we can achieve a significant time saving. If we have some 1,000 files and have modified only one file, then rather than compiling 1,000 files, we just need to compile one file and link it with the object files to update the binary. This saves huge compilation time. You can go through a Makefile to understand how this can be achieved.



Right GCC will still finish compiling even if it throws you warnings – it only halts the process if there is an error

Using compilation

Compilation is an important aspect of the software development process and compilers are extensively used to create executables whenever we are working on huge code base. In this tutorial, we take a look at the compilation process and how to use popular compilers (GCC for compiling C programs and G++ for compiling C++ programs). We also explain compiler optimisation and learn how to use various optimisation levels that will significantly improve the performance of our programs.

GCC and G++ are the most widely used compilers, partly because of the fact that both of these happen to be open source compilers.

```
gcc -ggdb -Wall -o sample_prog2 sample_prog2.c
```

The **-ggdb** part of the above command instructs the GCC to generate debugging symbols with GDB extensions. In order to get the maximum possible debugging information, you can use **-ggdb3**. Now let's take a quick look at some of the widely used GCC options.

- **-E option** The output of the pre-processing stage can be produced using this option
- **-S option** This can be used in order to produce the assembly level output
- **-c option** A developer can use this option to produce only the compiled code (without any linking)
- **-save-temps option** Through this option, output at all the stages of compilation is stored in the current directory
- **-l option** This option can be used to link with share libraries
- **-v option** Using this option, you can print out verbose information on all the steps GCC takes while compiling a source file

When we are talking about the compiler, we normally come across a concept known as compiler optimisation. We can use the optimiser as a part of the compiler and it is capable of examining your code, identifying the areas that are suboptimal and rewriting them using code that does the same thing in less space or with much better performance. The GNU compiler GCC has a powerful optimiser and it can be applied to your programs very effectively.

In GCC, we normally enable the optimisations by using the **-O** option. You can specify different levels of optimisation for GCC; if we use **-O**, it will be taken as **-O1** (ie level 1). You can normally go up to three levels (or **-O3**). To use the basic optimisations, you can execute a command similar to the following one:

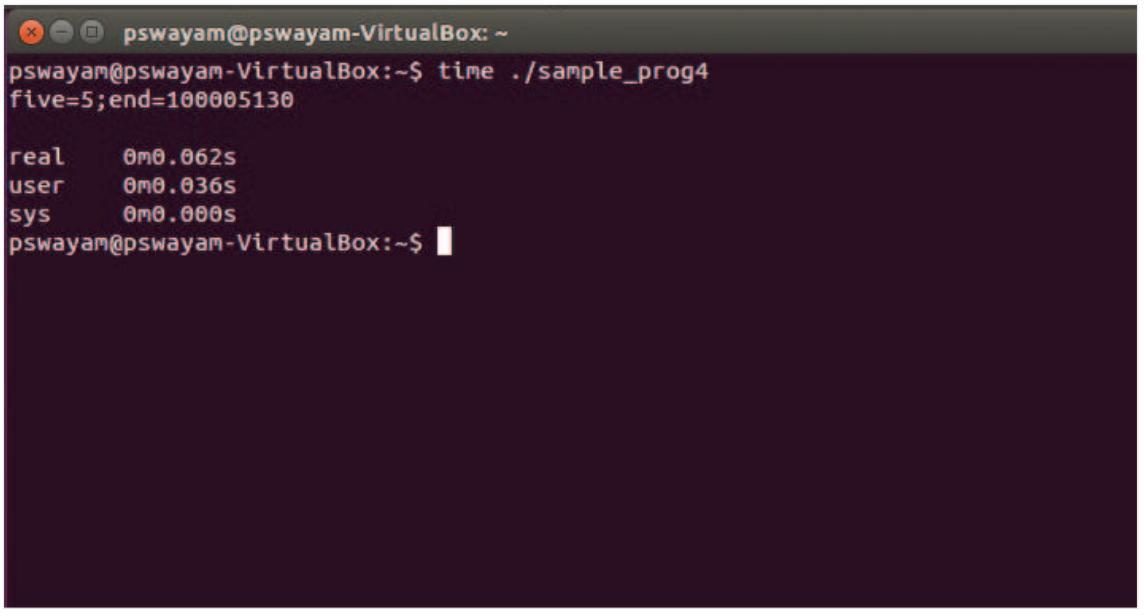
```
gcc -Wall  -O1 -o sample_prog3 sample_prog3.c
```

As mentioned earlier, we can use **-O** instead of **-O1,** or alternatively use **-O2** or **-O3**. These options basically control how aggressive GCC's optimiser is. As expected, more aggressive optimisation means that our code will run faster.

We are all aware that optimisation offers tremendous gains to the performance of your programs, but at the same time there are some drawbacks with the optimisation techniques. Some people prefer to compile their programs without any optimisations. This is mainly because the more aggressive GCC becomes with optimisations, the longer it takes for the program to finish compiling. It is always good to skip optimisations during the day-to-day development work, but make sure that you enable optimisations when it is the time to release the code. As mentioned before, we can have **-O1**, **-O2** or **-O3** as optimisation levels, but the **-O3** option can increase the size of the generated program. Even though this difference is not that significant, it becomes pretty important in some programs. If a program uses more RAM then swapping may occur on the machines on which it runs – this in turn can hurt the performance more than the gain.

Another disadvantage with the optimisation technique is that debugging can be very difficult when we enable optimisation. This means that tracing the execution of the programs can be difficult. Consequently, it is best to always skip the optimisation techniques as much as possible when we need to debug our programs.

The general rule is that the larger the number, the more optimisations are performed while compiling the code. Many developers prefer to compile their programs with **-O2** level of optimisation. This often provides the best compromise between optimisation strength, compile time and code size. Let us illustrate the optimisation with a code sample (our fourth sample program – sample_prog4.c):



Left These are the **time** results for our optimised file. Previously, we had 0.310s for real and 0.264s for user

```
#include <stdio.h>

int main(void)
{

    int cnt;
    int end;
    int temp;
    int five;
        for (cnt = 0; cnt < 2* 100000000 * 9/18 +
 5131; cnt += (5-3)/2)
        {
        temp = cnt / 15302;
        end = cnt;
        five = 5;
        }
        printf(" five = %d; end =  %d\n", five, end);
        return 0;
}
```

Let us first compile this program without any optimisation:

```
gcc -Wall -o sample_prog4 sample_prog4.c
```

The compilation will come through without any errors. Normally, we execute the program by using **./sample_prog4** at the command prompt. Since now we are interested in some statistics, we need to do it in a slightly different way. We have a command – **time** – and this command reports resource utilisation of your program when it finishes. This command basically indicates how much time the program takes to complete the execution:

```
time ./sample_prog4
```

Now let us try to execute the same program with one of the optimisation levels enabled:

```
gcc -Wall -O1 -o sample_prog4 sample_prog4.c
```

Execute the program by using the following command:

```
time ./sample_prog4
```

If we compare the two outputs, we can see a significant improvement in the execution time.

The main function of the above program was to illustrate the value that optimisation brings to the table and, purposely, we have written this sample code very inefficiently. Subsequently, in the above example, we saw that an optimiser has a lot of improvements to offer. If we are able to write code that is precise and efficient or a code with a very good flow, then an optimiser may be able to do even more for you (from a performance perspective) or you may not even need any optimisations at all. It is thereby worth noting the following two points here:

GCC's optimiser can significantly improve the performance of your programs.

You can always put more efforts into increasing the program speed of the optimiser.

We also have a GNU C++ compiler called G++ and it performs the same functions for C++ programs as GCC does for C programs. Strictly speaking, GCC can also compile C++ code. However, the result may not always be correct without specifying additional options. So it is always good to compile C++ programs using G++ compiler. Usually, we have .C or .cpp as an extension to C++ programs.

If we have the program my_testprog.C, we can use the following command to compile this C++ program:

```
g++ -Wall -o my_testprog  my_testprog.C
```

The executable can be run in the usual manner:

```
./my_testprog
```

Run complex container clusters on CoreOS

Nitish Tiwari
is a software developer by profession and an open source enthusiast by heart. As well as writing for leading open source magazines, he helps firms set up and use open source software for their business needs

Resources

CoreOS
coreos.com

Read on and discover how CoreOS uses containers to manage installed software simply

Containers are the standard for running software there days, so it was high time an operating system based on containers appeared. With CoreOS, this gap has been filled very well. Let's dig a little deeper to try and understand why a new Linux distro is needed at all when the current ones handle containers well.

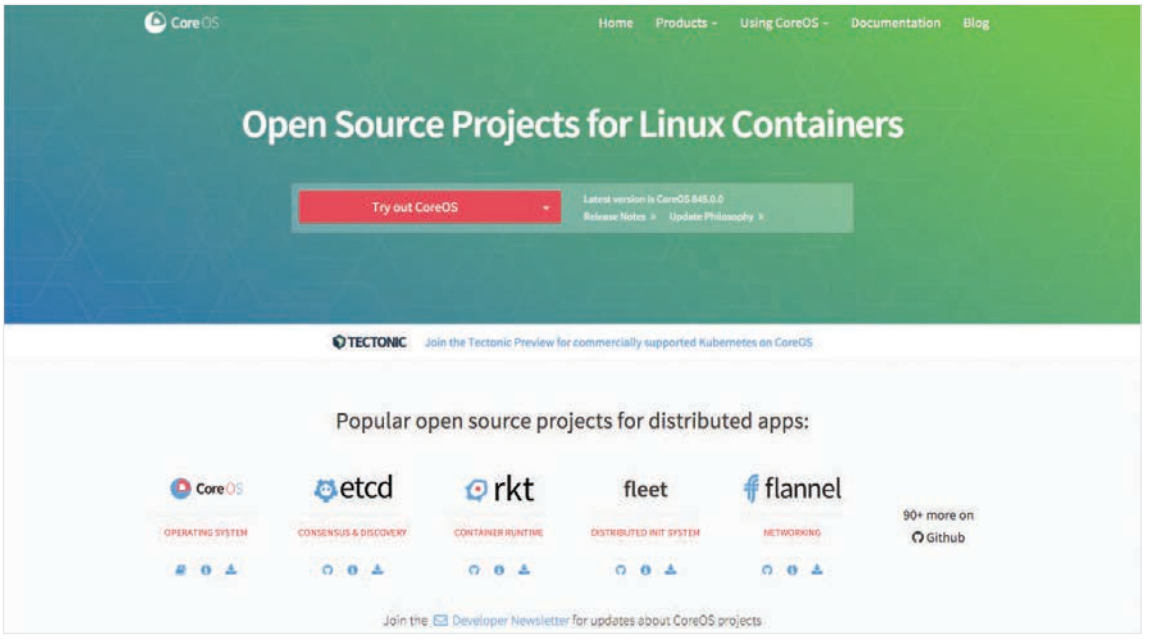
The problem is the fact that, while it is easy to use containers on current Linux distros, they don't play well when you need to manage (launch/destroy) several containers or link containers to create different back-end services. The new wave of containerisation, led by Docker, has made things lot easier – applications can now be easily deployed on almost any system. You just need the appropriate Dockerfile and Docker installation. But, if there are several hundreds of containers and you need to up/down-scale them frequently, current distros feel insufficient. CoreOS helps you address such scenarios.

Designed for security, consistency, and reliability, CoreOS is a Linux-based operating system for clusters. It is important to mention here that CoreOS doesn't ship with a package manager at all; rather, it comes with Docker pre-installed. You can use CoreOS to manage your services (using containers) instead of installing packages. So, for every service that you need, just create and use a container.

01 Preconditions

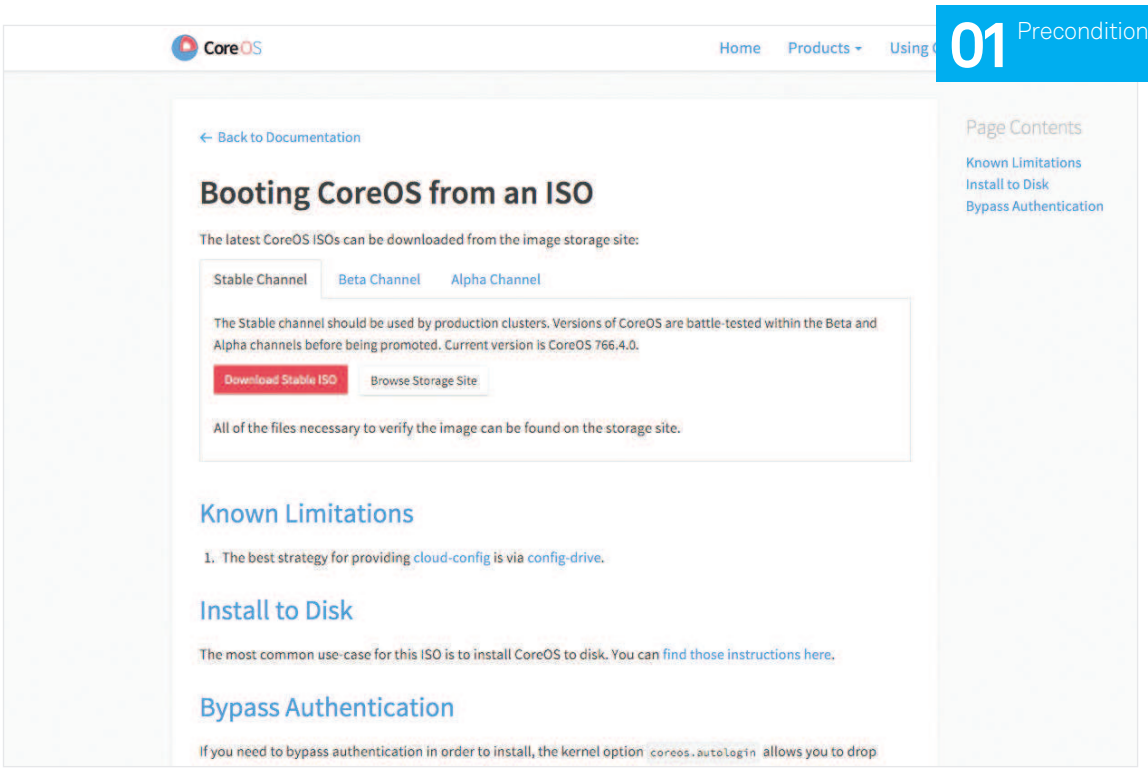
CoreOS is supported on almost all the major computing platforms – cloud providers like Amazon EC2, Azure, DigitalOcean, Google Compute Engine; virtualisation platforms like VMware, VirtualBox; cloud-based operating systems like OpenStack; bare metal servers and even your own personal computer. But, since CoreOS disk installer deletes everything on the target disk before installation, installing CoreOS on your current computer is a big ask – not everyone would want to abandon their current setup to install a new OS. It would be easier to get started with CoreOS by first trying it out on a virtual machine and then using it in its full glory by installing it on a cloud server. So, we will focus on installation of CoreOS as a virtual machine on your laptop (using VirtualBox) to learn its features.

The first precondition is having VirtualBox installed on your system. You can download the installer for your current OS from **virtualbox.org**. VirtualBox is available for all the major operating systems. The next thing is the CoreOS ISO file. This is the file we will use to mount CoreOS media on VirtualBox. You can get the latest stable revision of CoreOS from: **coreos.com/os/docs/latest/booting-with-iso.html**.



Right CoreOS integrates well other with distributed apps like etcd, rkt, fleet, flannel, etc

01 Preconditions



Left The ISO downloads are linked inside the CoreOS documentation

02 Cloud-config file

CoreOS offers a great way to manage OS-level items such as user accounts, network configurations and more, using the cloud-config file. Here is how this works – every time the system boots up, the coreos-cloudinit program looks for the cloud-config file and then configures the OS as per the settings in the file. Note that the cloud-config file is processed during each boot – if it is somehow corrupted then the OS will still boot, but you may not be able to log in with your credentials because the cloud-config file stores all of those user credentials.

It is also important to note here that this file is not auto-created – you'll need to create it yourself and add user account details for starters. Later on, as and when you need extra configurations, you can add them to the file. The cloud-config file uses the YAML file format that uses whitespaces and new lines to delimit lists, associative arrays and values. CoreOS also provides a command line tool to validate a cloud-config file – just run **coreos-cloudinit -validate** from the command prompt.

Now that we know few things about the cloud-config file, let us proceed with the installation process. We will also create the cloud-config file during the installation process.

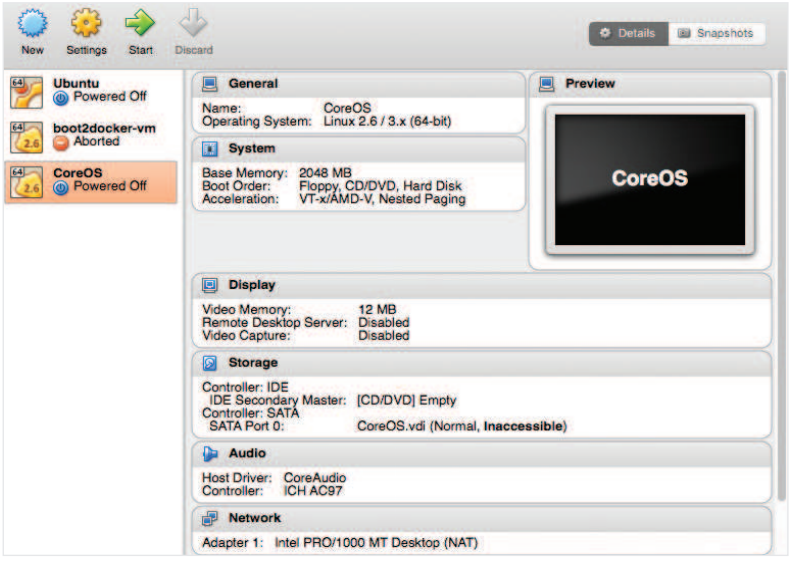
> "Since there is no package manager, any software that you'd like to run on your CoreOS installation should be run as a container"

03 Create virtual machine on VirtualBox

As discussed, we will install CoreOS as a virtual machine on VirtualBox. Start by firing up VirtualBox on your system. Click on the New button up in the top-left corner to create a new virtual machine. In the next page that follows, assign a proper name for the virtual machine, set the Type as Linux and the Version as Linux 2.6 / 3.x (64 Bit). The next page asks you to set the RAM for the virtual machine; anything above 2048MB should be fine for our CoreOS installation. The next step is to set the hard drive file type as VDI (Virtual Disk Image) and then storage on the physical hard drive as dynamically allocated. We set the disk size to be 8GB, which is the minimum requirement. Based on the free memory on your system and your needs, you can set it to anything above 8GB. Finally, hit the Create button. This creates a virtual machine with the specifications that have been set.

What is flannel?

Docker assigns an IP address to each container so it can communicate with other containers on the same host, but applications running inside containers can't advertise their external IP and port on the network. Flannel solves the problem by giving each container an IP that can be used for container-to-container communication. It uses packet encapsulation to create a virtual overlay network.



Above With the virtual machine created, you need to sort out the mount

04 Boot CoreOS for the first time

Now the virtual machine is created, we need to mount the CoreOS ISO to it and boot the live version of CoreOS. Only when the live version is booted can we create the cloud-config file and then install CoreOS on the disk.

To mount the CoreOS ISO, right-click on the CoreOS virtual machine we just created and select Settings. Then click on the Storage tab, and associate the CoreOS ISO file to the IDE controller. Now, go back to the list of virtual machines, select the CoreOS virtual machine and click Start (on the top bar). This boots up the CoreOS virtual machine using the ISO file attached to it as the boot media. After the boot messages, you should see the CoreOS command prompt with the default core user logged in. This confirms that the CoreOS ISO successfully booted and you can now create the cloud-config file before installing CoreOS to disk. Note that the core user automatically logs in for the first time, but it is mandatory to create a user (with the cloud-config file) in order to log in when CoreOS boots up the next time.

05 Create the cloud-config file

The cloud-config file follows the YAML file format and has specific entries. We need to follow the file structure to make sure that CoreOS recognises and processes the file. First is the header: it should be either **#cloud-config** or **#!**. The header should be followed by an associative array with zero or more of the following keys – **coreos**, **ssh_authorized_keys**, **hostname**, **users**, **write_files** and **manage_etc_hosts**. We'll focus on the **users** entry for now.

The users entry has three sub-fields: **name**, **passwd** and **groups**. These fields are self-explanatory, except for one catch – the **passwd** field holds the password hash and not the actual password, which is good for security but adds another step while you create the cloud-config file. You need to create the hash manually, then add it to the cloud-config file. One option is to use the **openssl** utility (available with CoreOS) to create the hash. Just type:

```
$ sudo openssl passwd -1
```

The system prompts you for your password and then asks you to verify it. Once successfully completed, you get the hash for your password. Add this hash into the password field of the cloud-config file. Note that while logging in, you need to key in the plaintext password. Once done with the fields, save the file with the .yml extension.

06 CoreOS disk installation

Now that we have the cloud-config file ready, let us install CoreOS on the disk so that we are independent of the mounted CoreOS ISO file. The CoreOS install process is pretty straightforward. You just need to use the **coreos-install** command. To install CoreOS to the disk, just type:

```
$ sudo coreos-install -d /dev/sda -C stable -c
cloud_config.yml
```

This will install CoreOS to /dev/sda device from the stable channel, using the cloud-config file cloud_config.yml. Note that you need to replace **cloud_config.yml** with the name of the cloud-config file you created in the previous step. Once you enter the command, CoreOS will be downloaded and installed to your virtual machine. Check the installation by unmounting the ISO image from the IDE controller and rebooting the virtual machine. If everything's correct, the virtual machine should reboot and you should be able to log in using the username and password set in the cloud-config file.

07 Docker on CoreOS

Docker containers form the main building block of CoreOS. Since there is no package manager, any software you'd like to run on CoreOS should be run as a container – so it's no surprise that Docker is pre-installed on CoreOS!

Using Docker on CoreOS is same as on any other Linux distro, but other services like systemd, fleet, etcd (more on them later), which blend well with containers and make connecting and managing them easy, make a huge difference. To refresh your Docker command knowledge, let us launch an Ubuntu container and then spawn the bash prompt in the container. Just type:

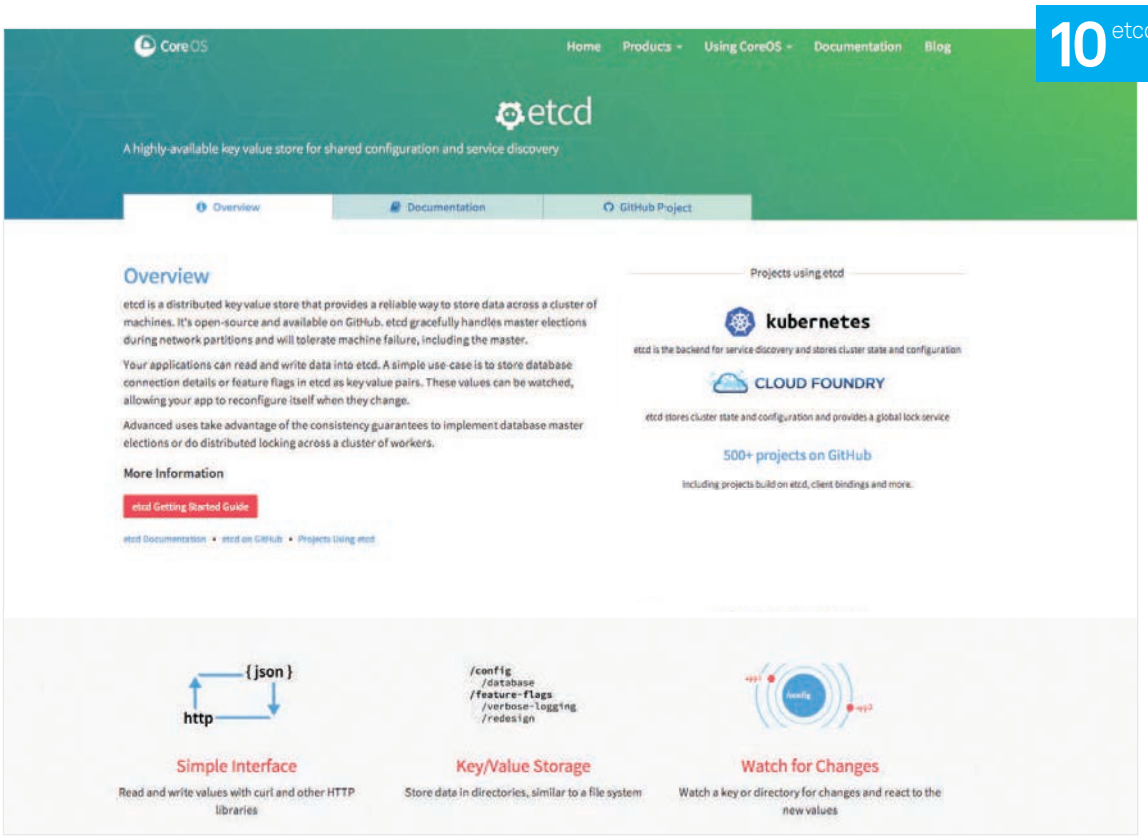
```
$ docker run -t -i ubuntu /bin/bash
```

08 Systemd

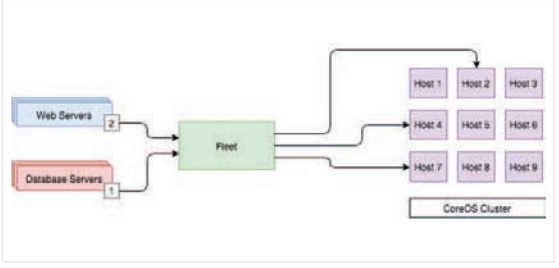
Systemd is the init system used by CoreOS. It is at the core of the distributed init system, fleet. There are two main concepts related to systemd. The first is a unit, which is basically a configuration file describing the properties of the process that will be run using systemd. The second is a target, a grouping mechanism that enables systemd to start up groups of processes at a time. Each target is actually a collection of symlinks to the unit files, required for that target. This is specified in the unit file by the field:

```
WantedBy=multi-user.target
```

It is worth mentioning here that systemd is the first process on CoreOS; once started, it reads different targets and starts the processes specified which enable the OS to start. In CoreOS the unit files are located at /etc/systemd/system.



Left Take advantage of etcd to store data when using a cluster of machines



09 fleet

Cluster management is one of the major USPs of CoreOS, and fleet is the service behind this. From the end user point of view, fleet lets you treat the whole CoreOS cluster as a shared single init system. This means developers don't need to worry about the cluster at all, and they can write applications as small ephemeral units that can be easily migrated around. Meanwhile the DevOps team can focus on running containers that make up a service, without having to worry about the individual machines each container is running on. For example, if an application consists of five containers, fleet will guarantee that they stay running somewhere on the cluster.

High availability is achieved by ensuring that the service containers are not on the same machine, or even the same availability zone in remotely distributed systems. You can use the **fleetctl** command to manage your cluster. Further, fleet units are segregated in two types – global units that run on all the nodes in the cluster when started, and standard units that are scheduled onto a single machine.

10 Etcd

Etcd is a distributed key-value store that provides a reliable way to store data across a cluster of machines. Applications running on the CoreOS cluster, irrespective of the node they are running on, can write their data to etcd and you can rest assured that they will get the data back whenever they need it. Common examples are storing database connection details, cache settings, feature flags and more. Note that etcd runs on each machine in a cluster and can gracefully handle loss of nodes and even the current master's loss.

Reading and writing to etcd is done via HTTP-based APIs. The **etcdctl** utility can be used directly from a CoreOS machine to set key and corresponding message. Just type the following into the command prompt:

```
$ etcdctl set /message Hello
```

Here, 'message' is the key and 'Hello' is the value. To read a value back, just pass the key to etcdctl. You can type:

```
$ etcdctl get /message
```

To delete the key you can type:

```
$ etcdctl rm /message
```

You can also access etcd from within a container, but must use the IP address that was assigned to the docker0 interface on the CoreOS host.

Rkt introduction

Rkt (Rocket) is an implementation of the App container specification. This is an open specification that defines several aspects of how to run applications in containers, like an image format, runtime environment and discovery protocol. The image format defined by App container specification is called ACI (Application Container Image) and the basic unit of execution is called a pod, which is basically a grouping of one or more app images (ACIs).

Make your perfect Plasma 5 desktop

Alexander Tolstoy
has spent years fine-tuning his Linux system to make it blazingly fast, but he keeps on searching for more tweaks. This kind of addiction seems to be a good one

Use hidden tricks to reveal the inner power of this bleeding-edge advanced desktop environment

The holy wars between KDE and Gnome users seem to be long over, perhaps because of a massive migration of the latter to Cinnamon, but many of the rest still love their KDE installation for being fast, configurable and feature-rich. But KDE 4, first released seven years ago, is moving aside to free the way for the shiny new Plasma 5, the next-generation hardware-accelerated desktop with a sleek modern look, yet retaining the much valued 'Swiss Army knife' approach, which always inspired KDE users and developers. Just like the aging KDE 4, Plasma 5 is a modular desktop made of plasmoids that can be rearranged in any way. The desktop itself is a converged shell, which adapts to different target devices on the fly. Together with modern Qt5-based applications, it makes the experience consistent and helps ensure the workflow is more elegant.

The new Plasma 5 desktop may not yet be mature enough, but it's been closing the gap very quickly in the past months, so that many people already use it on a daily basis. Kubuntu, OpenSUSE, KaOS, Manjaro and many others offer fine-tuned distributions with Plasma 5 out of the box. The steps here cover tips and tricks that work for both KDE 4 and 5, though some are Plasma 5-specific.

Resources

Plasma 5 desktop

Start and stop

The most obvious reason for restarting the graphical shell is applying settings that require more than just hitting the Apply button. Most people would simply log out and then log back in, but in this case all launched applications will be shut down, which is not preferable for many.

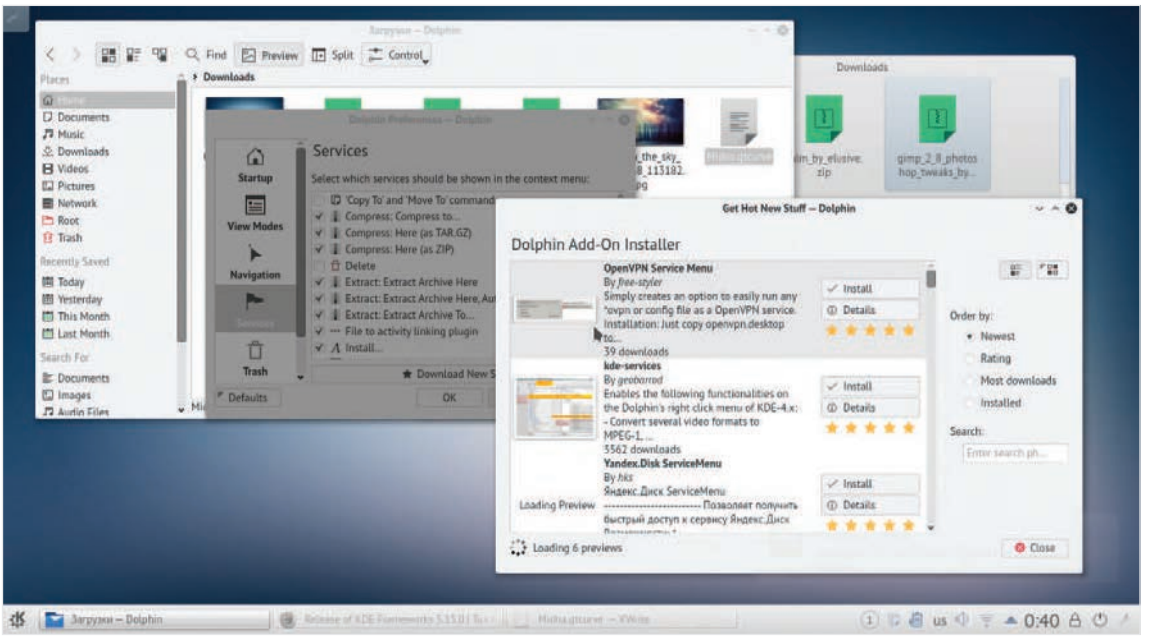
The Plasma 5 desktop is modular and it means that we can kill the shell without disturbing anything else, be it a background process or a graphical application. Open Konsole, or maybe hit Alt+F2 to show up the KRunner heads-up prompt, and type:

```
$ kquitapp5 plasmashell
```

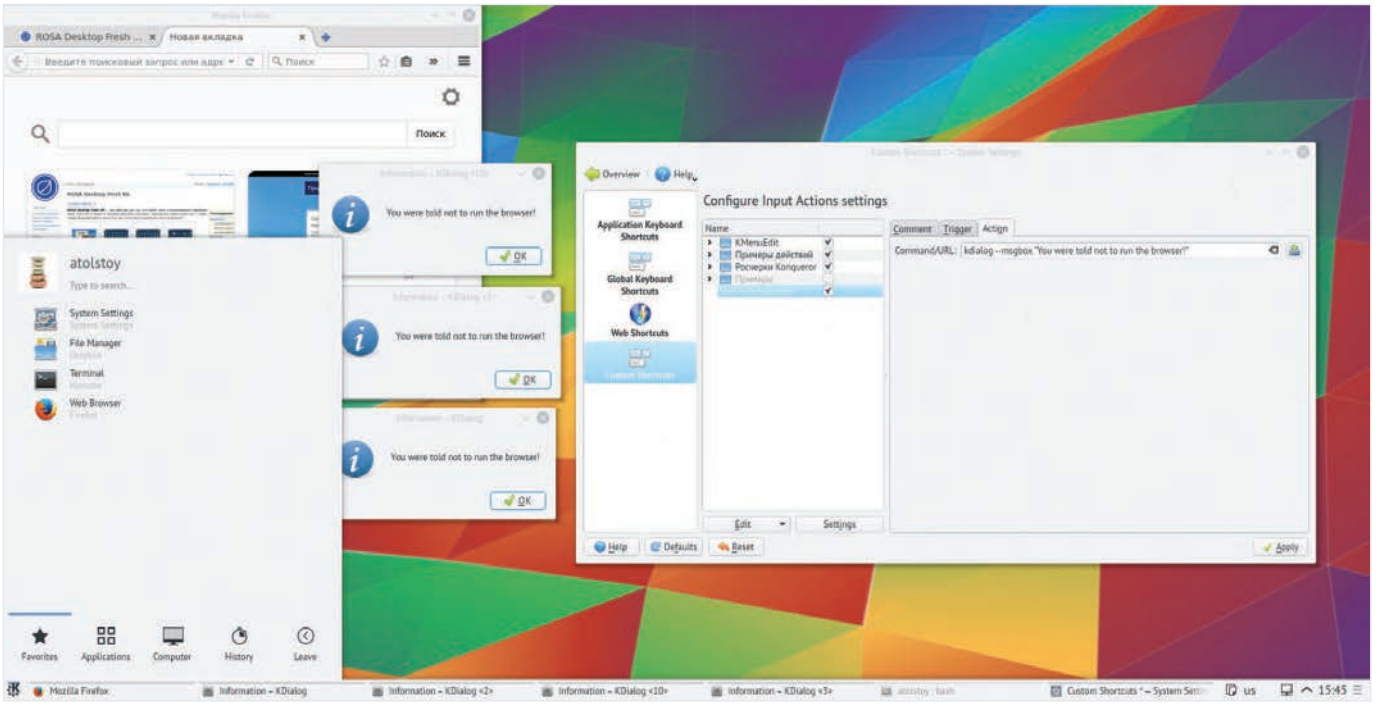
... and then:

```
$ plasmashell
```

... to start it again. All of your running apps will remain up. While the shell is down you can still access KRunner and even switch between applications by making use of the familiar Alt+Tab sequence.



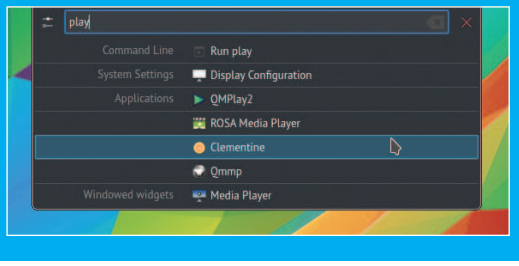
Right KDE Plasma 5 is perhaps the most customisable and theme-able desktop on Earth



The power of KRunner

KRunner is a handy tool for launching commands. Besides running Linux shell commands, KRunner instantly searches for GUI applications and offers all matches as you type. Then, it looks up system settings entries, Plasma widgets and folder names as well. For instance, if you type 'music' or 'play', KRunner will show all media players capable of music playback, your default music folder, the music plasmoid and also related sound settings.

KRunner can also be used as a search tool alongside Baloo5 (both share the same engine), as it displays the documents whose names match your entry. Of course, this is all expected, but let's add some magic. KRunner can do web browsing! Type your favourite web address starting with 'www.' and select the search result – the address will be opened in the default browser. Next, let's use KRunner as a calculator. When you type your calculation, KRunner detects it by the '=' sign at the beginning or the end of the sequence and shows the result instantly. The calculator module is very intelligent; it can handle equations with different units.



Use event-based actions

It's no secret that KDE's KWin manager is a powerful beast that can do almost anything you can think of, except for maybe making coffee or taking out the rubbish. You are probably already familiar with the very useful Shortcuts section in System Settings, where you can change certain keystrokes and even mouse gestures. But there's more to it, as KDE can trigger custom commands upon changing a state of a window. This can be useful when, for example, you want to implement custom parental controls or some kind of alarm in case of certain events that have occurred on a school computer. Event-based action means that you can trigger a command or a script when a window opens or closes, or gains or loses its focus. Let's see how it works in a simple example, where a message box will appear each time Firefox is launched.

In the Shortcuts section, right-click on the left pane and select New>Window Action>Command/URL and give a name to your action. On the Trigger tab, select the 'Window gets focus' option and then click Edit for the 'Window simple' entry. Here you can define how the target window will be identified. Commonly, there are many ways to detect a window, and in our example it is sufficient to tell it that the window title contains the word 'Firefox'. On the Action tab insert the following:

```
$ kdialog --msgbox "You were told not to run the browser!"
```

Once you hit Apply, your action will soon kick into action and then duly annoy anybody who uses Firefox on your PC. Feel free to play with this stunning feature and see what mischief you can get up to!

Above Use event-based actions to trigger commands

Compact UI elements

Plasma 5 has a distinctive visual peculiarity: application buttons are too large, with wide padding all around. If you don't like this, there is a simple cure: use a compact QtCurve theme, such as Midna (**github.com/KaOSx/midna**). Of course, you can tweak any theme manually, but switching to Midna is a nice, quick solution.

Use Plasma 5 in VirtualBox

Hardware-accelerated desktop is a killer feature of Plasma 5, yet the downside is its unstable performance as a guest in VirtualBox. The fix may sound brutal, but turning off 3D acceleration in the machine settings does the trick. You may also encounter the incorrect scaling of fonts and icons – this time you'll have to manually set the fonts' DPI to 96 in the KDE fonts settings.

Make use of the Windows key

Many new Linux users come from the world of Windows where there are a number of keyboard shortcuts and other features that people get familiar with. Those who use Gnome on a Linux PC or OS X on a Mac know that the desktop metaphors are completely different there and a user is supposed to change their habits in order to work efficiently on a new desktop. In KDE things are quite reversed: you can reproduce certain keybindings to work just the same as they do in Windows. In this example we will use the Windows button to launch the Start menu.

The default KDE Plasma desktop doesn't let you assign actions on a single key, but there's a workaround called KSuperkey. Once it is installed, just launch it with the **ksuperkey** command (and add it to KDE's autostart section) and then just hit the Windows key – in KDE it is detected as 'Meta'. KSuperkey works fine with the default Plasma 5 Application launcher.

Custom keybindings

As a continuation of the previous step, we'll try to reproduce the behaviour of the Win+D and Win+L combinations, which will minimise all windows and lock the desktop respectively.

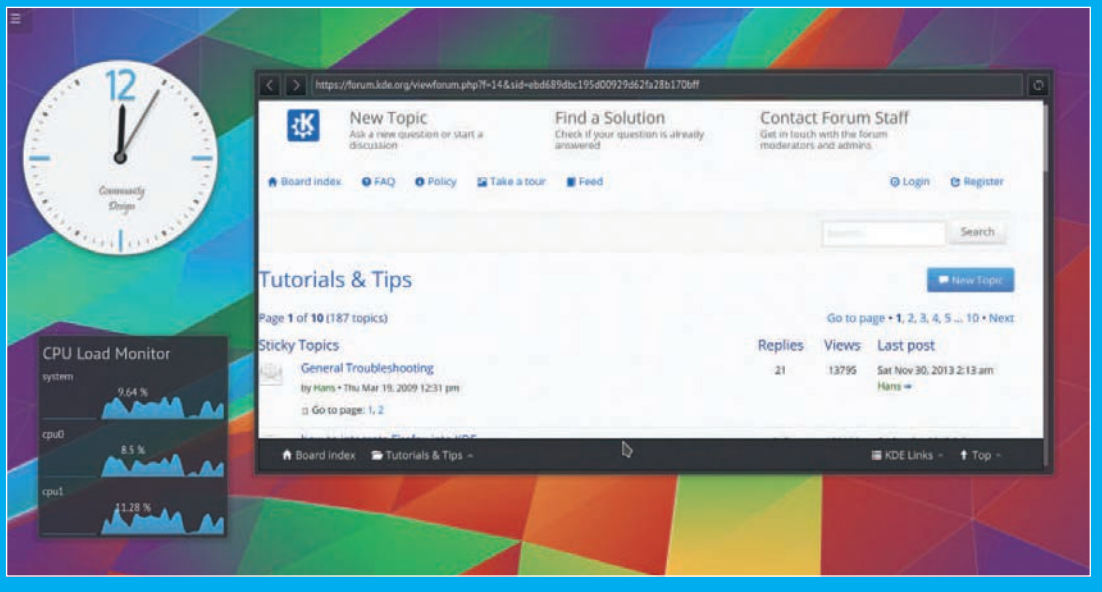
For the first trick we'll need the wmctrl package, which can be found in almost any Linux distro. Once you have it, create a script with the following content:

```
------
#!/bin/sh
target=on
if xprop -root  _NET_SHOWING_DESKTOP | fgrep '=
 1' ; then
    target=off
fi
wmctrl -k ${target}
------
```

Save it as a 'showdesktop.sh' file, then make it executable (**chmod +x showdesktop.sh**) and finally assign it to the Win+D binding in the Shortcuts section of System Settings.

Locking the screen is an easier process: there's no need to wrap anything into a script; instead, using a plain command will be enough:

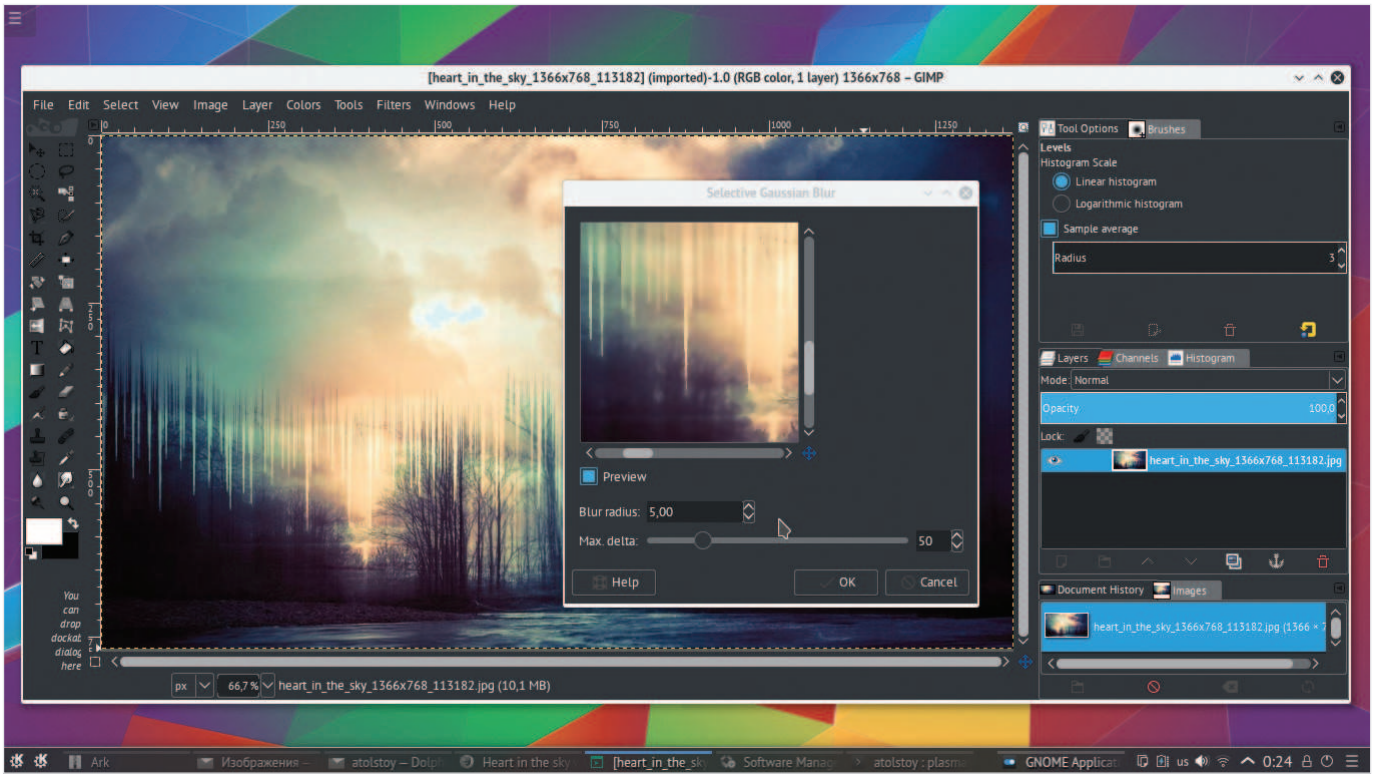
```
$ qdbus org.freedesktop.ScreenSaver /ScreenSaver org.
 freedesktop.ScreenSaver.SetActive True
```



Built-in web browser

There is an abundance of browsers for Linux, but how about a browser right on your desktop? We've seen something similar in Ubuntu's Unity, but in KDE a mini-browser is implemented as a plasmoid, meaning it doesn't get in your way together with applications' windows and it sits quietly on the desktop. To add it, just right-click on the desktop and select the 'Add Widgets…' item, then search for 'Web browser'. Double-click to add it to the desktop and see the default **www.kde.org** page opening in a tiny window. The plasmoid doesn't have many settings except for a shortcut binding, so to change the website, type its address in the plasmoid's text entry field with the mandatory 'www.' prefix. The plasmoid will load your page and auto-adjust its layout to the size of the plasmoid itself – you can resize to make the page inside the plasmoid readable.

The web browser plasmoid currently uses the QtWebkit engine, the same as in the Rekonq browser, but the KDE team plans to switch to the newer QtWebEngine, which is based on the Chromium code and renders pages much better. Still, the current plasmoid is very capable and has a decent score of 390 points at **html5test.com**.



Launch single settings modules

All KDE Plasma 5 settings are conveniently grouped into the System Settings application, which can also be launched by calling upon the **systemsettings5** command. There is no denying that this application is very nice, but finding the perfect setting can take some time and you might want to access everything in a more elegant way. This can be done using the **kcmshell5** command. First, let's see what modules we have:

```
$ /usr/bin/kcmshell5  --list
```

Use any module as an argument to **kcmshell5** and it will then launch the respective single module without accessing the control panel. For example:

```
$ /usr/bin/kcmshell5 kwincompositing
```

... will launch the desktop compositor's setup. This technique lets you launch any module from the list and then, maybe, create a shortcut with such a command, or bind it with a keyboard shortcut. It is also possible to launch several modules with a single command. For instance:

```
$ /usr/bin/kcmshell5 mouse nic xserver
```

... will display a nice window with all these modules shown as different sections inside a single window.

For even Plasma 5 tips and tricks, it's always worth paying a visit to the KDE wiki –community members are always uploading guides to **userbase.kde.org/Tutorials**.

> "Plasma 5 desktop is modular, so we can kill the shell without disturbing anything else, be it a background process or a graphical application"

Uniform look

As far as we can remember, the first attempt to clear the visual difference between Qt-based and GTK-based apps was taken in Red Hat Linux 8.0 with the introduction of the Bluecurve theme. Nowadays, the problem is still present: modern GTK2 and GTK3 apps don't quite fit well into the KDE desktop. There are various workarounds for this issue, including the QtCurve theme, which works well for GTK2 but has poor support for GTK3. Another option is to use the Orion GTK theme, which looks similar to Plasma 5 Breeze (the default theme) but still has huge visual inconsistences with it. Luckily, we have a working port of the Breeze theme to GTK at **github.com/dirruk1/gnome-breeze**, covering both light and dark variants. Simply copy the theme into either ~/.themes or /usr/share/themes and then select it in System Settings>Widget style>GNOME Application style (GTK). Now you have a consistent visual experience regardless of the graphical toolkit used.

Above This GTK-based version of the Breeze theme will keep your desktop looking consistent

Create user interfaces with MonoDevelop forms

Tam Hanna
has been in the IT business since the days of the Palm IIIc. Serving as a journalist, tutor, speaker and author of scientific books, he has seen every aspect of the mobile market more than once

Resources

MonoDevelop
monodevelop.com

Creating GTK-based UIs used to be an exercise in frustration, but MonoDevelop changes all that…

Working with GTK is a pain: its handle-function-driven coding paradigm might have looked cool in the time of Palm OS, but it looks hopelessly dated now.

Vala attempted to improve the situation by providing an object-oriented wrapper for most GUI classes. This led to more compact code, but it's miles away from true RAD (rapid application development); seeing your user interface as you work on it is a true productivity booster. Visual Basic was always a leader in this respect. The very first version of the product ran under DOS, but it did not skimp on providing a WYSIWYG-GUI editor.

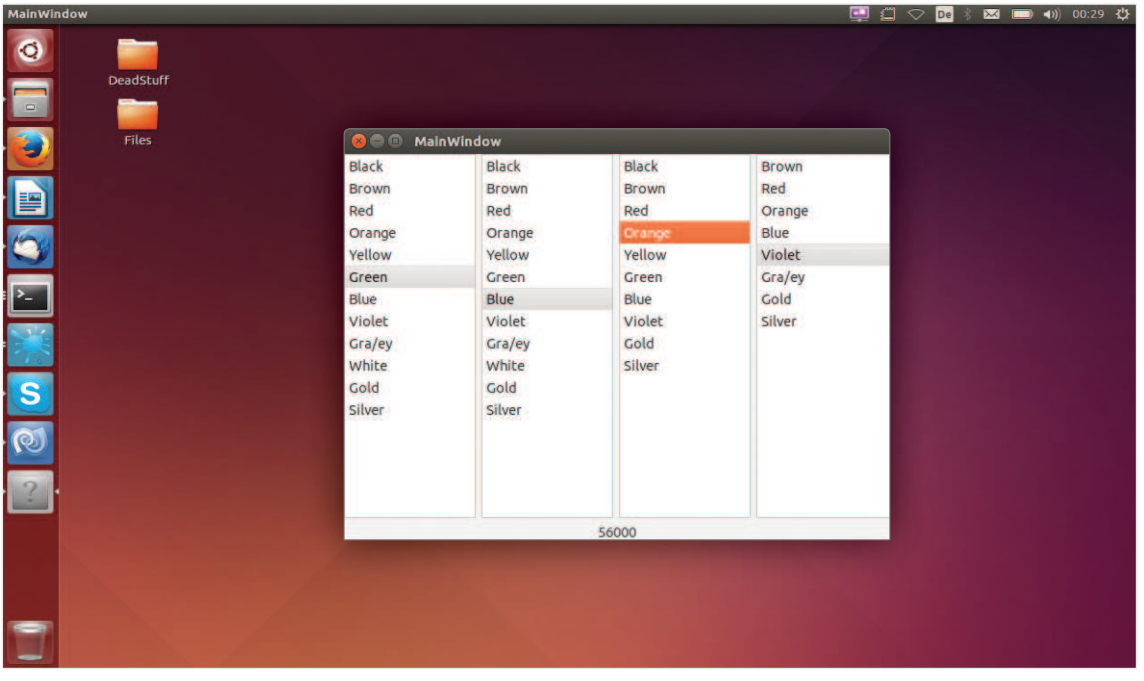
Even though Mono contains a runtime emulation for Microsoft's Windows.Forms namespace, MonoDevelop's WYSIWYG editor is based on GTK. Since that framework is just as cross-platform as Mono, using GTK should not be an issue for most desktop-based applications. Furthermore, many of the concepts discussed here can also be applied to a variety of other GUI tools.

Begin by firing up MonoDevelop. We covered the installation and deployment instructions in the first part of this C# and MonoDevelop series (issue #157). Then create a new project based on the template C#>Gtk 2.0 Project.

MonoDevelop will produce a default GTK 2.0 project skeleton for you. In addition to the known C# files, our project now also comes with a folder containing a form layout and a resource file with a few icons. Actual program execution is still handled via Program.cs, which by default contains the following code:

```
class MainClass
{
  public static void Main (string[] args)
  {
    Application.Init ();
    MainWindow win = new MainWindow ();
    win.Show ();
    Application.Run ();
  }
}
```

Application.Init() is a static method domiciled in the GTK framework's **main** class. After creating and displaying a new instance of the MainForm class, we invoke **Run()** in order to start the event loop.



Right We'll make a simple resistor colour code calculator this time

Our form is made up of a WYSIWYG-component arrangement and a code-behind class that looks like this:

```
public partial class MainWindow: Gtk.Window
{
  public MainWindow (): base (Gtk.WindowType.Toplevel)
  {
    Build ();
  }
```

MainWindow's constructor calls the **Build()** method. It connects to the MonoDevelop back-end and uses the logic found there in order to spawn a ready-to-run instance of our form. **OnDeleteEvent** shows two interesting paradigms:

```
protected void OnDeleteEvent  (object sender,
DeleteEventArgs a)
{
  Application.Quit ();
  a.RetVal = true;
}
```

First of all, **Application.Quit()** is invoked during the closing of the MainForm. This prevents the app from living on in the background in a 'zombie-esque' state.

Secondarily, a flag is set to mark the event as 'completely handled'. This is important as most GUI stacks create a similar event pipeline: events file past a group of event handlers, which either accept or reject each event. If the developer's code shows no interest in an event, it is usually 'buried' by the system default handler.

Encode resistors

Electricians *hate* dealing with resistors – their weird colour coding makes finding out component values difficult. Consequently, it is not surprising to see a large variety of apps dedicated to this rather simple task.

We will start out by double-clicking MainForm's form file. It can be displayed in the text or GUI modes by clicking the Source or Designer toggles at the lower-left, respectively.

Small tabs located on the right-hand side of your screen will permit you to open the toolbox or the property dialog. The first contains a list of various controls that can be placed in the form via drag and drop, while attributes and settings related to the currently selected element are shown in the Properties dialog.

Developers switching from Visual Studio need to accustom themselves to a small oddity. Selecting controls in the form preview will not cause their properties to show up; instead, you need to select them in the document's outline window instead.

We will limit our program to the decoding of four-band resistors to reduce complexity. Increase the form's size by selecting it in Designer mode and drawing its borders out.

Tree view

Users have ever-changing tastes with regards to screen resolutions: some swear by their 1280 x 800 12-inch HP EliteBook, while others work their magic on a 100-inch set of two 4K screens. Sadly, human vision does not scale so wide – a fact that is accommodated by frequent resizing of forms.

GTK+ forces you to respect this by requiring the adding of forms. Add a HBox and populate its top cell with a VBox in order to achieve the look shown in the Document Outline in the image below. The amount of grid cells and their spacing can be adjusted by right-clicking the element in hand. MonoDevelop reacts to this by displaying a context menu with further options. Next, add four tree views along with a label by dragging them into the grid created by our HBox and VBox layouts. When done, the forms should be in the same structure as the left-hand side of the image below.

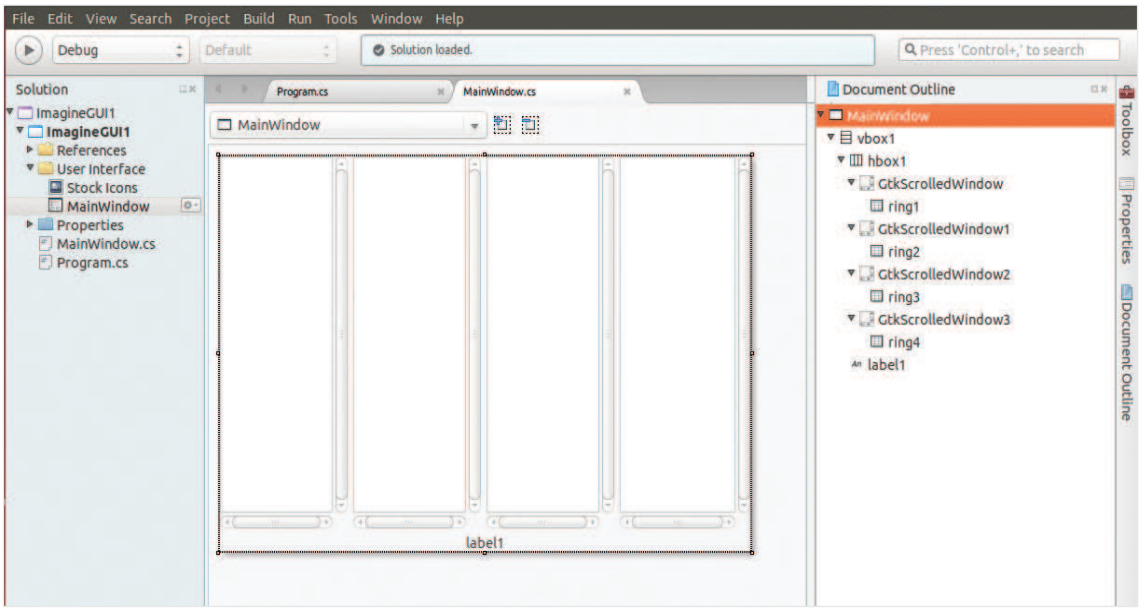
MonoDevelop's version of GTK+ does not provide classic list boxes. Instead, developers are to add a tree view to their form and restrict it to traditional list mode by manipulating it from the code-behind. The properties permit you to assign friendly

Add another form – part 1

Our calculator is a special example in that it can make do with only one form, as most applications need a group of them. Let us deploy an About button in order to add some realism.

Reopen the WYSIWYG part of MainForm and add a line to the layout. Then add a button and connect its Clicked method with the following code:

```
protected void
BtnClicked (object
sender, EventArgs e)
{
  AboutWindow myWin
= new AboutWindow ();
  myWin.Show ();
}
```



Left Our resistor calculator will be made up of four adjacent scrolling window forms

Add the form – part 2

With that completed, it's time to add the second form. Click File>New>File and select the template Gtk>Window. Set the name field to 'AboutWindow' and click New, in order to add the resource to the project. In the next step, remove the **namespace** declaration from the code-behind file. Run the program in its current state to feast your eye on the second empty dialog – it is just waiting for your creativity!

names to the widgets. The next steps assume you've got the same structure and tree view as shown in the previous image.

Model madness

Tree views differ from classic list views as they provide additional flexibility by implementing the model view controller concept. Data is displayed via a combination of headers, layout managers and a data container model.

```
public MainWindow (): base (Gtk.WindowType.Toplevel)
{
  Build ();
  var column = new Gtk.TreeViewColumn ();
  column.Title = "RING";
  ring1.AppendColumn (column);
  ring1.HeadersVisible = false;
```

Firstly, the basic 'model' of the data stored in the tree view has to be set up. We only have to add only one column of data; its title does not need to be displayed.

The next step concerns itself with the creation of the actual data. It finds sanctuary in a ListStore – and because it can also work as a multicolumn store, **AppendValues** is a variable argument function:

```
var rStore=new Gtk.ListStore(typeof(string));
rStore.AppendValues ("Black");
rStore.AppendValues ("Brown");
rStore.AppendValues ("Red");
rStore.AppendValues ("Orange");
rStore.AppendValues ("Yellow");
rStore.AppendValues ("Green");
rStore.AppendValues ("Blue");
rStore.AppendValues ("Violet");
rStore.AppendValues ("Gra/ey");
rStore.AppendValues ("White");
rStore.AppendValues ("Gold");
rStore.AppendValues ("Silver");
```

Finally, the actual display process needs to be started up. This is accomplished by assigning renderer classes to the rows of the model. They display each element assigned to them via their embedded drawing logic. As we are dealing with plain text, the humble **CellRendererText** is all we need:

```
var renderer4Cell = new Gtk.CellRendererText ();
column.PackStart (renderer4Cell, true);
column.AddAttribute (renderer4Cell, "text", 0);
ring1.Model = rStore;
```

The other three rings are to be populated by analogue: adjust the number of elements added to the list by adjusting the amount of invocations of AppendValues along with the colour names provided to them.

Be aware that each 'helper object' can be assigned to only one list box at a time. Attempting to recycle, such as the one shown in the listing, is a sure-fire way to harvest a crash:

```
var column = new Gtk.TreeViewColumn ();
column.Title = "RING";
ring1.AppendColumn (column);
ring2.AppendColumn (column);
```

Run the program to see the form finally resemble the images on the previous and opposite pages. If you populated the rows correctly, our resistance calculator is ready to rumble.

Handle some events

Displaying a list of resistor values is quite a feat, but sadly, our product currently does not do much else. Since the minimisation of clicks is always beneficial, we will update the value displayed whenever the user clicks an item in one of the lists. This can be done by adding event handlers. Select one of the tree views and switch to the Signals pane of the Properties window.

Next, click the white area next to Tree View Signals Cursor Changed. MonoDevelop will replace the white space with a text box where a method name can be entered. Finally, press Enter to generate the following skeleton; it will not be shown automatically, but it's waiting for you in the code-behind:

```
protected void CurChange (object sender, EventArgs e)
  {
    throw new NotImplementedException ();
  }
```

Since duplicating program logic is bad, we wire the activation events to a common method called **refreshContent**. Due to space constraints, we restrict the printed code to the new version of **CurChange**:

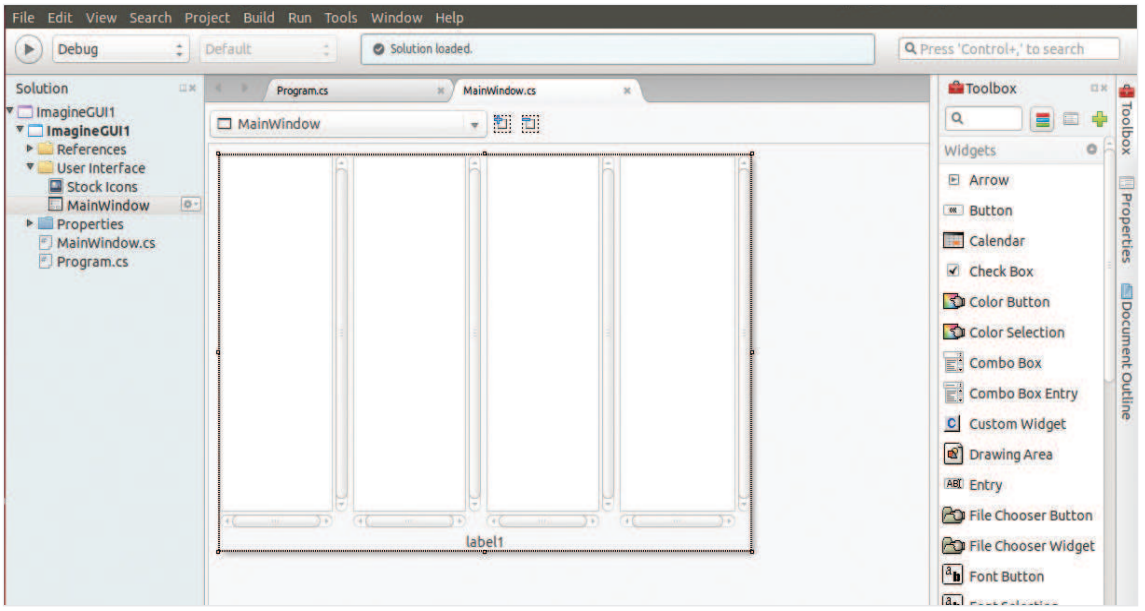
```
protected void CurChange (object sender, EventArgs e)
{
  recalculateWindow ();
}
```

If multiple widgets are to share one event handler, advanced programmers can avoid multiple functions via a small trick. Copy the text found in the first control's event-handler field and paste it into the other controls. MonoDevelop detects the similarity and refrains from spawning another method.

The **recalculateWindow** starts out by figuring out the currently selected row in each of the windows: if one of them is 'untouched', we print an error instead:

```
private void recalculateWindow()
{
  if (ring1.Selection.CountSelectedRows () == 0 ||
    ring2.Selection.CountSelectedRows () == 0 ||
    ring3.Selection.CountSelectedRows () == 0 ||
    ring4.Selection.CountSelectedRows () == 0) {
    label1.Text = "Missing Values";
    return;
  }
```

The next step involves the actual calculation of the resistance. Our code is intentionally contrived as it demonstrates two approaches to the problem of deciding which item is actually selected. In the case of a known relationship between list index and element, a formula can be devised to determine the content without a model lookup:

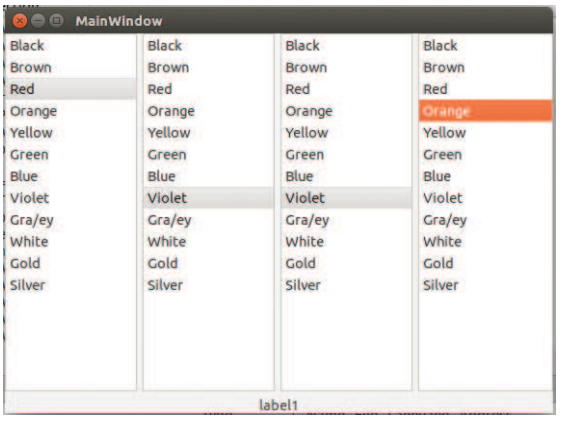


Left Explore the Toolbox to find plenty of elements that you can drop in

```
float val;
TreePath[] temp1=ring1.Selection.GetSelectedRows ();
TreePath aPath1 = temp1 [0];
TreePath[] temp2=ring2.Selection.GetSelectedRows ();
TreePath aPath2 = temp2 [0];
val = aPath1.Indices [0] * 10 + aPath2.Indices [0];
```

If you find that the index number of the element at hand is not enough, code can also extract the part of the model responsible for the row selected. This is accomplished as per the following:

```
TreeModel model;
TreeIter iter;
var s = ring3.Selection;
s.GetSelected (out model, out iter);
var str3 = model.GetValue (iter, 0).ToString ();

switch (str3)
{

case “Black”:
   val = val;
   break;
case “Brown”:
   val = val*10;
   break;
case “Red”:
   val = val*100;
   break;
case “Orange”:
   val = val*1000;
   break;
case “Yellow”:
   val = val*10000;
   break;
case “Green”:
```



Left At this stage, event handlers have not been added and the label is still blank

```
   val = val*100000;
   break;
case “Blue”:
   val = val*1000000;
   break;
case “Violet”:
   val = val*10000000;
   break;
case “Gold”:
   val = val*0.1f;
   break;
case “Silver”:
   val = val*0.01f;
   break;
}
```

Finally, the content is printed to the label by assigning it to its Text property. Our example omits the analysis of the tolerance ring for brevity reasons:

```
label1.Text = val.ToString ();
```

Mihalis Tsoukalos
is a Unix administrator, a programmer (for Unix and iOS), a DBA and also a mathematician. He has been using Linux since 1993

Resources
Text editor
GCC compiler

Computer science: Find strings with hash tables

Take the time to learn how to use hash tables and discover how they make your life easier

Hash tables are a key component of computing and programming languages; they are being supported in almost any programming language, including AWK. Strictly speaking, a hash table is a data structure that stores one or more key and value pairs. A hash table uses a hash function to compute an index into an array of buckets, or slots, from which the correct value can be discovered. Ideally, the hash function will assign each key to a unique bucket, provided that you have the required number of buckets. As you will see, this rarely happens. In practice, more than one key will hash at the same bucket.

The worst restriction of a hash table is that once you define the hash function and the number of buckets, neither of these two properties can be changed without rebuilding the entire hash table. Therefore the best thing you can do to make a hash table faster is to increase the number of buckets and make sure that the hash function uses all buckets!

The hypothetical problem that you are going to solve here is to find out whether a word can be found in a dictionary or not – this is an operation that has to occur fast.

All C programs will be compiled as follows:

```
$ gcc -Wall programName.c -o programName
```

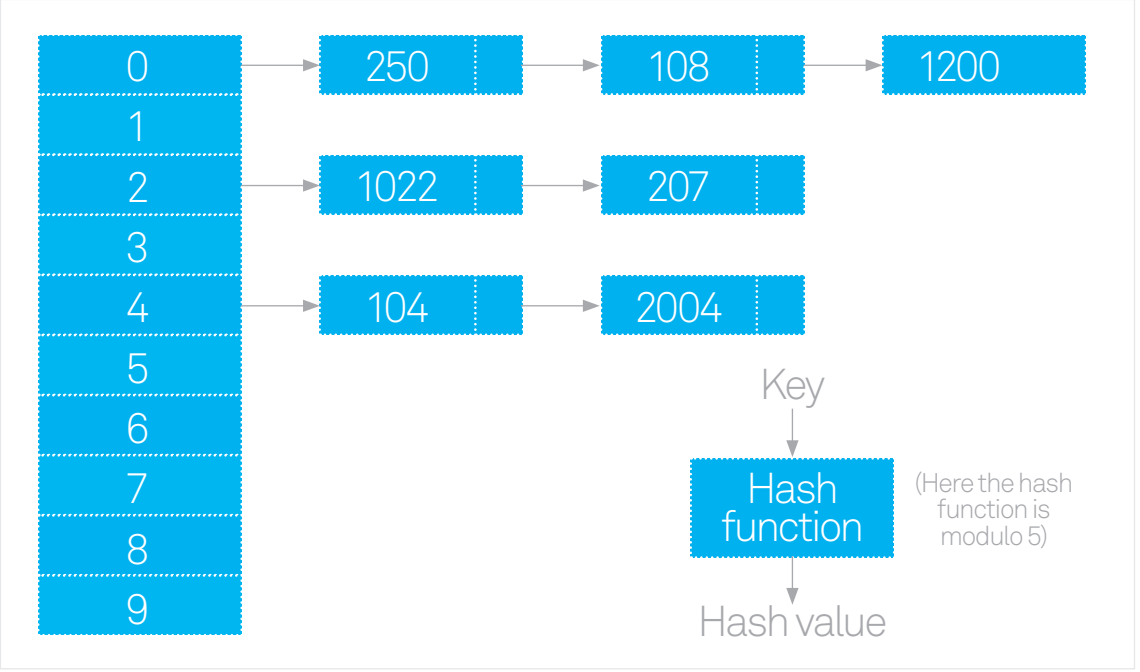
The theoretical part

The single most important characteristic of a hash table is the number of buckets used by the hashing function. The second most important characteristic is the hash function itself. The most crucial feature of the hash function is that it should produce a uniform distribution of the hash values, as it is inefficient to have unused buckets or big differences in the cardinalities of the buckets (see the page 52 image).

Having n keys and k buckets means that the search speed goes from O(n) for a linear search to O(n/k). Although the improvement might look small, you should realise that for a hash array with only 30 buckets, the search time would be 30 times smaller!

The diagram on the next page shows a simple yet fully characteristic hash table. The only problem with it is that it has a poorly chosen hash function, which does not return hash values that will fill all available buckets – as you can

Tutorial files available:
filesilo.co.uk



Left This diagram illustrates what a simple hash table looks like

see, half the buckets are not being used at all! If you know what the modulo operator does, you can easily fix this by using modulo 10 in the hash function instead of modulo 5.

It is important for the hash function to work consistently and output the same hash value for identical keys, because otherwise it would be impossible to find the information you want. A collision happens when two keys are hashing to the same index – this is not an unusual situation. There are many ways to deal with a collision.

A good solution is to use separate chaining: the hash table is an array of pointers, each one pointing to the next key with the same hash value. When a collision occurs, the key will be inserted in constant time to the head of a linked list. The problem now is that when you have to search the hash value of a given key, you will have to search a whole linked list for this key. In the worst-case scenario, you might need to traverse the entire linked list; this is the main reason why the linked list should be moderately small, raising the requirement for a large number of buckets. As you can understand, resolving collisions involves some kind of linear search, which can be slow by design. So it is important that you have a hash function that minimises collisions as much as possible. Other techniques for resolving collisions include open addressing, Robin Hood hashing and 2-choice hashing.

The hash function

As the hash function is going to be used all the time in every interaction with the hash table, it is extremely important for the hash function to operate fast. Please note that a poorly designed and implemented hash function may leave a number of buckets unused, which not only makes the finding process slower, but is also a terrible waste of resources. The best way to avoid this is by examining the cardinality of the buckets as often as possible in order to test the efficiency of the whole implementation.

In the worst-case scenario, where the hash function puts all elements into the same bucket, the hash table is in reality a linked list with an extra overhead – the execution of the hash function – and a plain linked list would therefore have been a faster solution.

Delete, insert and look up

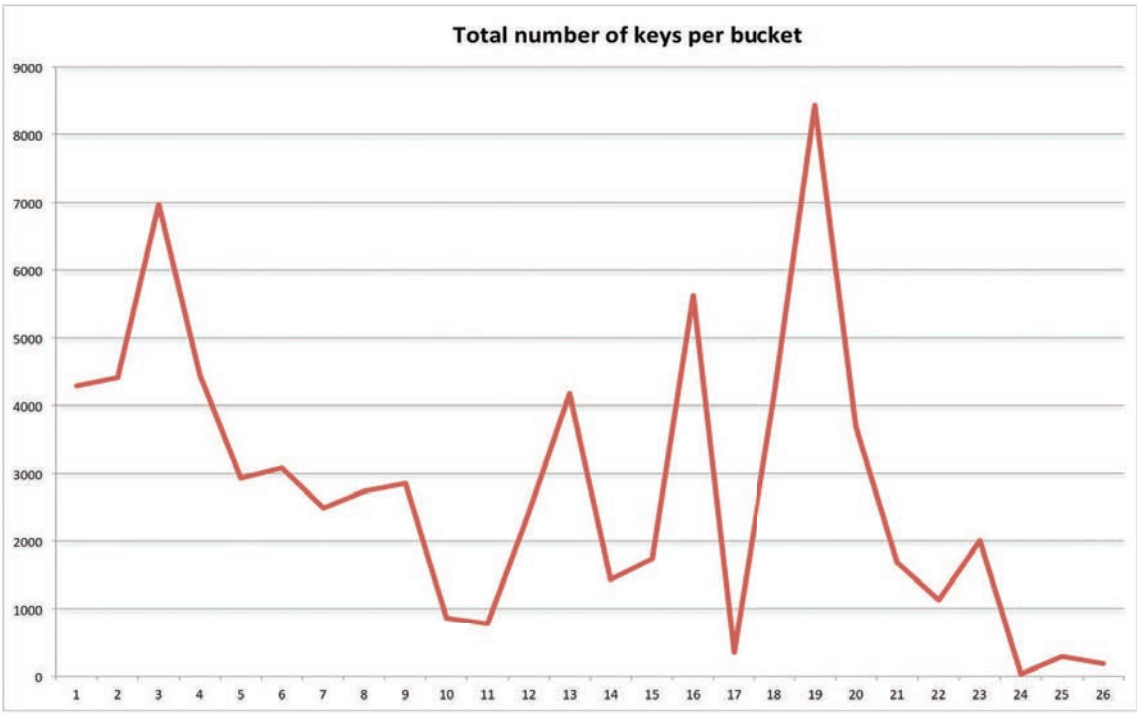
The main operations on a hash table are insertion, deletion and looking up. You use a hash value to determine where in the hash table a key will be stored. Later, you use the same hash function to determine where in the hash table to search for a given key.

Once the hash table is populated, searching is the same as doing an insertion. You hash the data you are searching for, go to that place in the array, look down the list that starts from that location and see if what you are looking for can be found in the linked list or not. The number of steps is **O(1)**. The worst-case search time for a hash table is **O(n)** – that can happen when all keys are stored in the same bucket! Nevertheless, the probability of that happening is so small that both the best and average cases are considered to be **O(1)**.

You can find many hash table implementations both on the Internet and by reading good books. The tricky part is using the right number of buckets and choosing an efficient hash function that will distribute values as uniformly as possible without bias. A distribution that is not uniform will definitely increase the number of collisions and subsequently slow down the entire hash table, as shown above.

Implement a hash table in C

The following listing shows the C code for hashTable.c, which uses a native hash function. The purpose of hashTable.c is to understand how a hash table works; this can be reached by examining its main points – the fully commented version can be found in this tutorial's **FileSilo.co.uk** resources.



Right Here we see unused buckets and large differences in bucket cardinality – this is inefficient

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#define TABLESIZE 1001

typedef struct node
{
  char *data;
  struct node *next;
} node;

unsigned int hash(const char *str, int tablesize)
{
    int value;
    value = toupper(str[0]) - 'A';
    return value % tablesize;
}
static int lookup(node *table[], const char *key)
{
    unsigned index = hash(key, TABLESIZE);
    const node *it = table[index];

    while (it != NULL && strcmp(it->data, key) != 0)
    {
        it = it->next;
    }
    return it != NULL;
}

int insert(node *table[], char *key)
{
    if (!lookup(table, key))
    {
        unsigned index = hash(key, TABLESIZE);
        node *new_node = malloc(sizeof *new_node);

        if (new_node == NULL)
            return 0;
        new_node->data = malloc(strlen(key)+1);

        if (new_node->data == NULL)
            return 0;

        strcpy(new_node->data, key);
        new_node->next = table[index];
        table[index] = new_node;
        return 1;
    }
    return 0;
}
int populate_hash(node *table[], FILE *file)
{
    char word[50];
    char c;

    do {
        c = fscanf(file, "%s", word);
        size_t ln = strlen(word) - 1;
        if (word[ln] == '\n')
            word[ln] = '\0';
        insert(table, word);
    } while (c != EOF);
    return 1;
}
```

How linked lists work

Hash tables use linked lists for storing their data. Is it mandatory to use linked lists? No, but linked lists have many advantages. Linked lists are really good at sequential searching, especially when used with C pointers; they are a simple and elegant data structure that can dynamically grow, do not need any extra space apart from a few pointers, and are good at insert operations. Finally, they do not need too much housekeeping, provided you don't delete elements from a hash table often.

However, linked lists have disadvantages. If a linked list has many elements, reaching the last element of the list can be really slow. Additionally, linked lists can be really slow when dealing with sets with a large amount of elements: one way or another, a linked list with 100,000 nodes will be slow!

A forthcoming tutorial will talk more about linked lists; for now, look at linkedList.c for more information. You will find the linkedList.c code analogous to the code in hashTable.c.

```
void printHashTable(node *table[], const unsigned
 int tablesize)
{
    node *e;
    int i;
    int length = tablesize;
    printf("Printing a hash table with %d
 buckets.\n", length);

    for(i = 0; i<length; i++)
    {
        e = table[i];
        int n = 0;
        if (e == NULL)
          printf("Null bucket %d\n", i);
        else
        {
            while( e != NULL )
            {
                n++;
                e = e->next;
            }
        }
        printf("Bucket %d has %d keys\n", i, n);
    }
}

int main(int argc, char **argv)
{
    char word[50];
    char c;
    int found = 0;
    int notFound = 0;

    node *table[TABLESIZE] = {0};
    FILE *INPUT;
    INPUT = fopen("INPUT", "r");

    populate_hash(table, INPUT);
    fclose(INPUT);
    printf("You can now use the Hash Table!\n");
    int line = 0;
    FILE *CHECK;
    CHECK = fopen("CHECK", "r");

    do {
        c = fscanf(CHECK, "%s", word);
        size_t ln = strlen(word) - 1;
        if (word[ln] == '\n')
            word[ln] = '\0';
        line++;
        if (lookup(table, word))
            found++;
        else
            notFound++;
    } while (c != EOF);

    printf("Found %d words in the hash table!\n",
 found);
    printf("Not found %d words in the hash table!\n",
 notFound);

    fclose(CHECK);
    return 0;
}
```

C code is not difficult, but you will have to make sure that you fully understand this before continuing. The node structure is defined at the beginning of the program and contains two fields: the data field – a single value – and a pointer to the next node of the linked list. You can have as many fields with data as you want; the only required field is the pointer that points to the next node of the linked list. Apart from the compulsory **main()** function, there are five more functions.

After populating the hash table using a text file named INPUT, **main()** checks the performance of the hash table by checking if the words in the CHECK text file are present in the hash table. The only operation not implemented is deleting a key, because deleting a key is a very rare operation on a dictionary. The **hash()** function does what its name implies and is the most important function of the entire program. The **lookup()** function checks whether a value is already present in the hash table. Both insertion and removal operations need the **lookup()** function; the former for avoiding duplicates and the latter for ensuring that you are not trying to remove a value that does not exist in the hash table. The **insert()** function finds the correct bucket, according to the return value of the hash function, and inserts a new value in front of the appropriate linked list, provided that the value is not already present. The **populate_hash()** function reads a text file that contains values and inserts them into the hash table.

The last function, **printHashTable()**, is a utility function that displays the cardinality of each bucket. The graph opposite shows the cardinality for each of the 26 buckets of the hash table with the help of the **printHashTable()** function for the given input and the given number of buckets. The input file that was used contains about 100,000 unique words.

Implement a hash table in Perl

Perl has built-in support for hash tables that are also known as Dictionaries. You cannot control the hash function that

Test the speed

Now let us compare the speed of the C and the Perl implementations. As you can see in the graph at the bottom of the page, a C implementation that uses a good hash function with a large number of buckets can beat the Perl implementation. Nevertheless, the Perl implementation is pretty fast. On the other hand, once again you can understand that a small number of buckets can destroy an otherwise good hash function.

is controlled by Perl itself. Moreover, each element (value) has a unique key, which is the element itself. The problem with the Perl implementation of hash tables is that you have absolutely no control over the process and the way Perl works with dictionaries. However, the Perl way has many advantages, including simplicity as well as easier lookup, insert and delete operations. The code for a Perl implementation of a hash table can be found inside your **FileSilo.co.uk** resources: hashTable.pl.

The main problem with the Perl implementation is that it needs more memory than the C version, which is mainly caused by the fact that each word has its unique place in the dictionary structure.

Perl is not the only programming language that natively supports hash tables, though. Ruby, Python, PHP, AWK and many others support hash tables, which are also known as associative arrays. The main disadvantage of such implementations is that it is almost impossible to keep your data sorted without using a supplementary structure; however, what is the point of using a supplementary structure to support your hash table and not use the supplementary structure on its own?

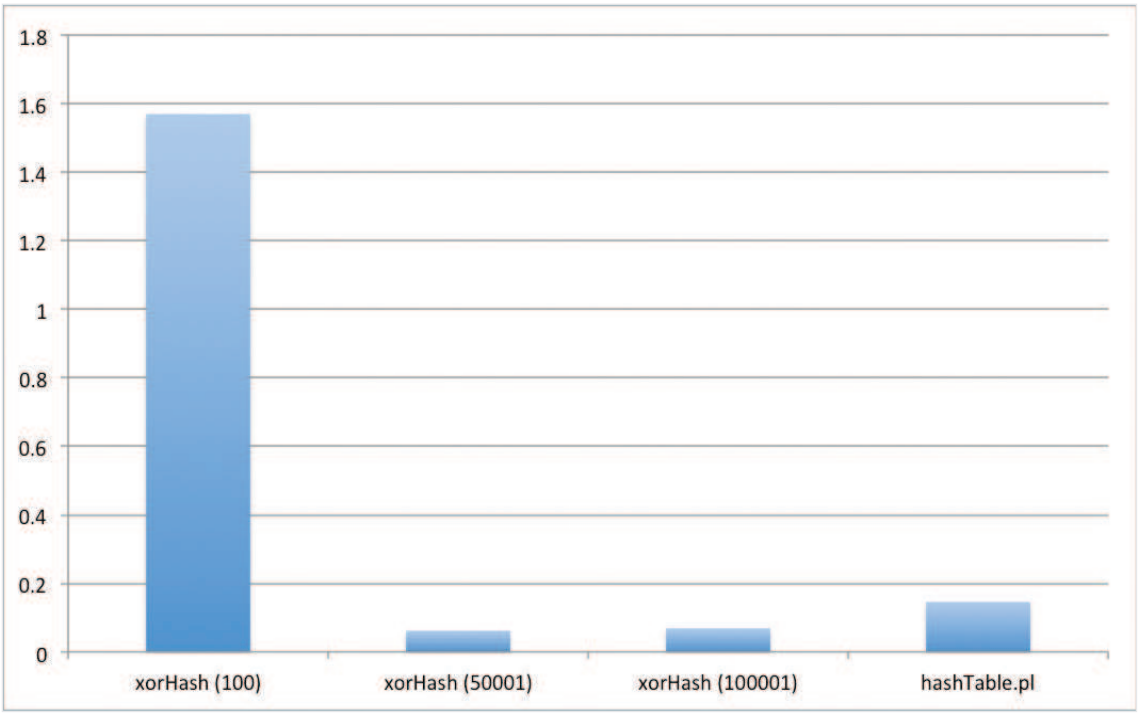
If keeping the data sorted is not something that you find you particularly need, then associative arrays might be the ideal solution for storing your data in a flat way – you will have as many buckets as the cardinality of your data without the need for multiple linked lists to store the data. As we mentioned earlier, you just have to trust that your programming language uses an efficient method for performing lookup and insertions, which is not always a bad thing – especially when you consider the fact that the internals of the programming languages are usually being improved when a newer version comes out.

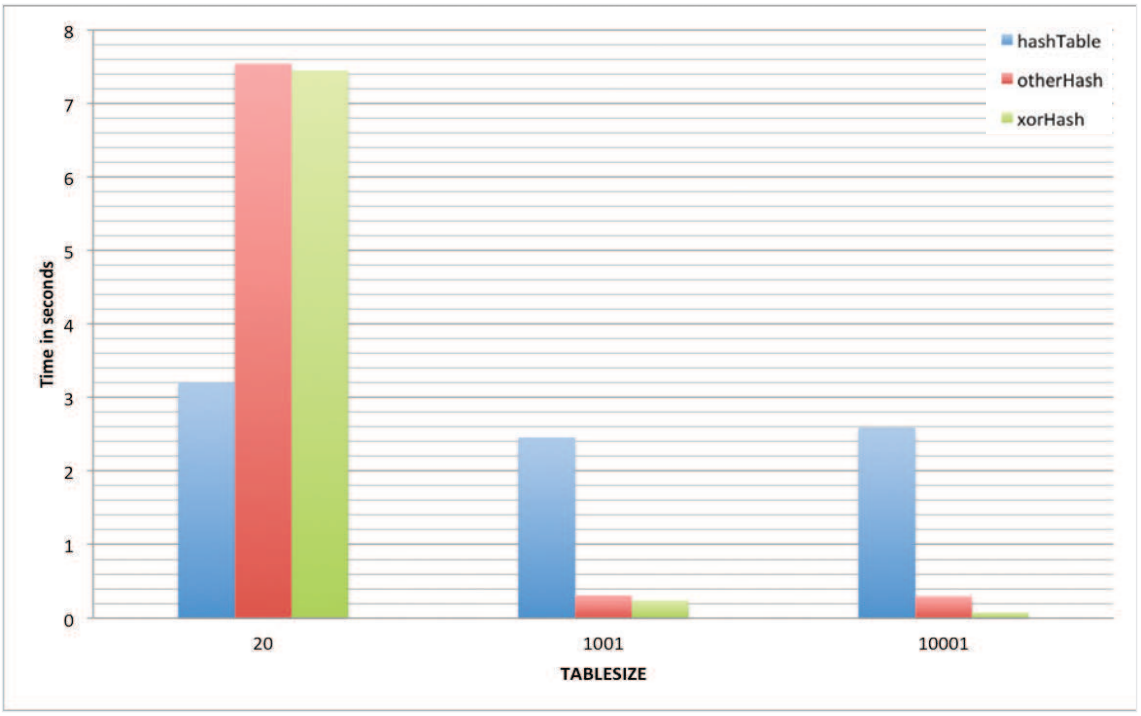
The benefit of hash tables

The main reason for using hash tables is their ability to keep the average cost for a lookup independent of the number of elements stored in the table. If insertions and deletions are not enabled, which is not a rare scenario, it is relatively easy to decrease the average lookup cost by carefully choosing the hash function, the bucket table size and the internal data structures used. Hash tables also perform well when the total number of entries is known in advance and the bucket array can be allocated once with the optimum size, without ever needing to be resized.

Hash tables have disadvantages as well. First of all, they are not so good at keeping data sorted. Therefore it is not efficient to use a hash table if you need your data to be sorted. Hash tables are not efficient when the number of entries is very small because despite the fact that operations on a hash table take constant time on average, the cost of a good hash function can be significantly higher than the inner loop of the lookup algorithm for a sequential list or search tree. For certain string-processing applications, hash tables may be less efficient than trees or finite automata; however, both trees and finite automata have a more complicated way of operating. Although the average cost per operation is constant and fairly small, the cost of a single operation may be fairly high. In particular, if the hash table uses dynamic resizing, inserting or deleting, a key may once in a while take time proportional to the number of entries – this may be a serious drawback in applications where you want to get results fast. Hash tables also become quite inefficient when there are many collisions; the difficulty is that you might not know in advance the number of collisions you are going to have, particularly when inserting dynamic unknown data.

Right The far-left column shows how slow a hash function without enough buckets can be





Left OtherHash.c and xorHash.c both get significant speed boosts from a larger **TABLESIZE** value

Do some benchmarking

As only two things can vary in a hash table implementation, the benchmarking process will only change these two factors. Three different hash functions will be used (hashTable.c, otherHash.c and xorHash.c) as well as three different TABLESIZE values (20, 1001 and 10001). So a total of nine tests will be performed and measured using the time command.

The graph above shows the performance of the three hash functions using the various table sizes. You should now understand the importance of a good hash function and a large array that holds the various linked lists. Please remember that when you change the TABLESIZE variable in the C code, you should recompile the C file for changes to take effect.

As you should realise by now, a good hash function has to evenly distribute elements and minimise collisions as much as possible. Additionally, it should take into account the kind of data it has to process as well as the amount, because without knowing its data, a hash function cannot perform well. The quest for the perfect hash function is endless; nevertheless, the best advice when trying to implement a hash function is to always consider the kind of data you are going to have.

Compare a hash table to a linked list

This part will compare the search (lookup) speed of a hash table and a linked list; both implementations will be in C. Please bear in mind that the concept of a linked list is simpler than the concept of a hash table. Generally speaking, the simpler the concept, the more general its usage can be. However, the biggest problem of linked lists compared to hash tables is that when you insert a new node, you will have to traverse the whole linked list in order to make sure that the new node does not already exist in the linked list.

Hash tables in Python

This part will talk a little bit about how Python implements hash tables, which are called Dictionaries. The following Python code illustrates the way Python deals with dictionaries:

```
Hash = {}
Hash['key'] = 1
Hash['key2'] = "12"
Hash['key3'] = 3
for key in Hash.keys():
    print Hash[key]
```

The first line defines a new dictionary variable named Hash. The next three lines put data into the dictionary. The last two lines print all values found in the hash dictionary. More or less, the Python approach is similar to both Perl and AWK approaches and you should have no problems understanding it.

In order for the comparison to be fair, a large number of elements should be used for creating the hash table and the linked list. The linked list is the slowest of all presented programs except when the hash table has only one bucket. However, if the linked list elements were sorted then its search performance would have been much better. The most important benefit of a sorted linked list is that checking whether a node already exists in the linked list is much easier and does not require traversing the whole linked list. When the list needs to be sorted, the insertion can be very slow. Inserting an element to an unsorted linked list is faster.

Deleting an element from a hash table and a linked list is basically the same as searching for it. The actual cost of deletion is the lookup operation.

EXCLUSIVE
BOOKS & SPECIAL EDITIONS

★ GO TO GREATDIGITALMAGS.COM NOW ★

ALL AVAILABLE TODAY ON
GREATDIGITALMAGS.COM

CWCS
MANAGED HOSTING

Linux Server Hosting
from UK Specialists

24/7 UK Support • ISO 27001 Certified • Free Migrations

Managed Hosting • Cloud Hosting • Dedicated Servers

Supreme Hosting.
Supreme Support.

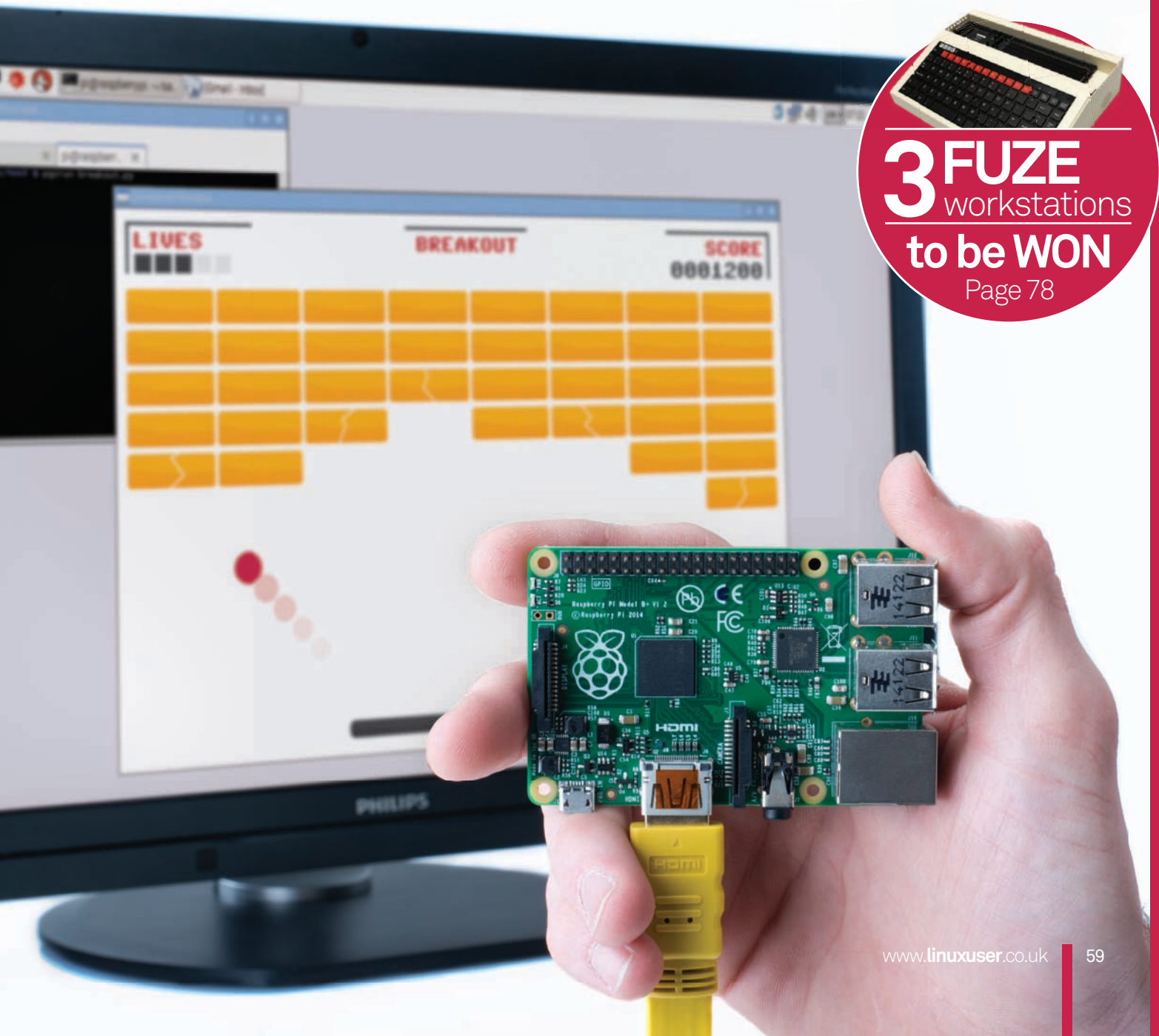
www.CWCS.co.uk

T: 0115 740 1234 • E: info@cwcs.co.uk • W: www.CWCS.co.uk

THE ESSENTIAL GUIDE FOR **CODERS & MAKERS**

PRACTICAL Raspberry Pi

Contents



3 FUZE workstations to be WON

Page 78

Wireless A NeoPixel Ring shines through this magic lamp when it is activated by the embedded Qi charger, which can also be used with mobiles

Computer The display uses the Raspberry Pi 2, with a keyboard-and-mouse combo hidden on the underside of the desk with a Velcro mount

Contactless Frederick made his own breakout board for the hover buttons, combining the Atmel AT42QT1070 touch sensor with a PCB

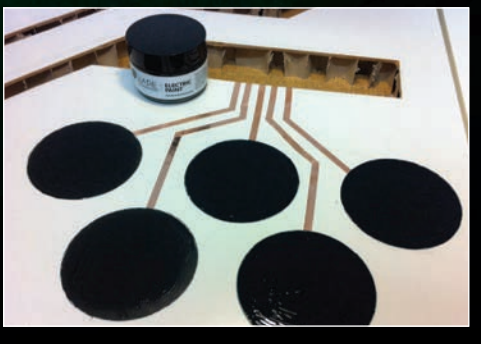
Lights The LED strip shines through the Plexi surface, and Frederick's script alters the LED's colour when the computer is popping up or down

Components list

- 12V power supply
- 12V-to-5V DC-DC converter
- 2 WiPi USB dongles
- **Desk** Raspberry Pi B+, USB sound card, 2.5W Class D amplifier, Gertbot, 2 stepper motors, 2m LED strip, capacitive touch sensor, 2 microswitches (on/off), mini speaker
- **Computer** Raspberry Pi 2, 2.8W Class D Amplifier, laptop LCD display, LCD display controller, 2 speakers
- **Magic lamp** Qi wireless charger, Qi wireless receiver, Adafruit NeoPixel Ring, Adafruit Trinket

Left The magic behind the lamp is a well-made base, picked to fit the NeoPixel Ring and the Qi receiver. When the Qi circuit connects, the LEDs shine through a sphere

Below For the capacitive touch buttons, copper tape was laid down and then Bare Conductive paint was applied over the top



Tron PiDesk

Frederick Vandenbosch transforms his desk with touch sensors, wireless charge pads and a retractable computer

Tell us about the competition that you entered

Sci Fi Your Pi was a challenge launched by element14 in collaboration with the Raspberry Pi Foundation. The goal was to create a sci-fi project centred around the Raspberry Pi, and it had to have a certain relation to a sci-fi movie. There were 25 finalists for the challenge and we had to build our projects in about 18 weeks.

Which movie did you choose?

I had the idea for the project first and then I had to link it to a movie, so I had to reverse the thinking in that sense. I linked it to *Tron* because of different aspects: you have the lights, you have the costumes with embedded LEDs or light strips, and this is why I have this pattern in the desk surface, to try to represent that. You also have the desk that is used to transfer between the virtual world and the real world, which also has an embedded computer.

What are the features on your desk?

The idea was to make it as futuristic as possible and also combine multiple features. The desk has to look like an ordinary desk, but once you start hovering your hand over the right side of the desk, you have embedded capacitive touch sensors which can trigger certain actions. For example, the main action is to make the computer pop up out of the desk, and this will then trigger some lights to let the user know that an action is in progress. When the computer is rising out of the desk you have the pattern light up in green. The reverse happens when you decide to shut down the computer – the screen will lower into the desk and you will have a moving pattern in red. I decided to use green and red because they're typical colours for starting and stopping. When you hover your hand over the capacitive touch sensors, it also triggers some sound effects based on MP3 files stored on the Raspberry Pi.

One of the other features is that I've embedded a wireless charger into the desk's surface – I use it to light up a lamp, based on a glass ball and NeoPixels put into a certain container, but you could also use it to charge your phone.

Do the capacitive touch controls only pop up and retract the display?

I have foreseen five buttons but I'm only using two. But there is a Python script on the Pi waiting for any input of those five buttons, and any action can be triggered. If I come up with an interesting idea then I will definitely expand the program.

How did you build the desk itself?

I knew I wanted to embed things inside the desk's surface, because I wanted it to look like a normal desk. I used an Ikea desk – the cheap ones are hollow – so I started drawing the patterns I wanted and some space to store the Raspberry Pi and the wireless charger, and then I started cutting into the top surface of the desk. I removed the cardboard structure, which then gave me space inside the height of the desk, and this was sufficient to hide components. For the sensors, because I wanted to keep them as flat as possible, I used conductive paint and copper tape to make the connections, so there's no level difference between one side of the desk and the other. Finally, I had to close everything up, and for that I used a big sheet of Plexi. I first covered the desk with two layers of paper from a paper roll, to hide the components and make it non-visible, and then by putting a top layer of Plexi you have a full surface for the desk. The lights are bright enough to shine through the paper, and the capacitive touch sensors are sensitive enough to be able to trigger by placing your hands over the Plexi.

Do you have speakers embedded inside the desk for the sound effects?

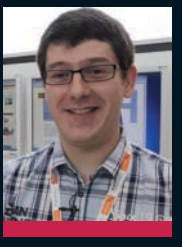
There are two parts for the audio: the Pi embedded in the desk, which is in charge of controlling the motors, lights and sound effects, so there is a small speaker using an amplifier breakout board inside the desk to trigger the sound effects of the controls. Then you have the desktop computer, which slides in and out of the desk – this is a separate Raspberry Pi with a separate amplifier, separate speakers.

How did you go about creating the pop-up display?

I had an old laptop from years ago and when it broke down, I decided to recoup the parts that I could – the display was one of them. I managed to find a control board on eBay, for exactly that type of LCD screen, and so I was able to re-use the screen using a 12V power supply. For the challenge, we received a kit containing a Raspberry Pi but also some add-on boards – one of those was a Gertbot, which is an add-on board specifically for the control of motors. I based the idea on the Z axis of 3D printers, where you have threaded rods which can then raise or lower a platform. I adapted the same principle by mounting two stepper motors on the bottom side of the desk with threaded rods, with the whole screen assembly resting on that platform. When the stepper motors are triggered, the platform is raised, causing the whole frame to come out of the desk. To have some guidance for this frame I used some drawer sliders, also from Ikea. The stepper motors have good torque, so they can handle the weight of the entire frame, but the speed is a bit of a problem. But in the end, it turned out that the boot time of the desktop computer is exactly the same as the rising time of the entire frame.

What was the most challenging part of the project?

I bumped into an issue where the control of the LEDs was impacting the audio, and vice versa. I ended up using an external USB dongle for the audio so that it wouldn't impact the GPIO control for the LEDs. Time constraints make you figure out things in any way possible. So the most challenging part is really the integration, once you start putting everything together and making things interact with each other. ■



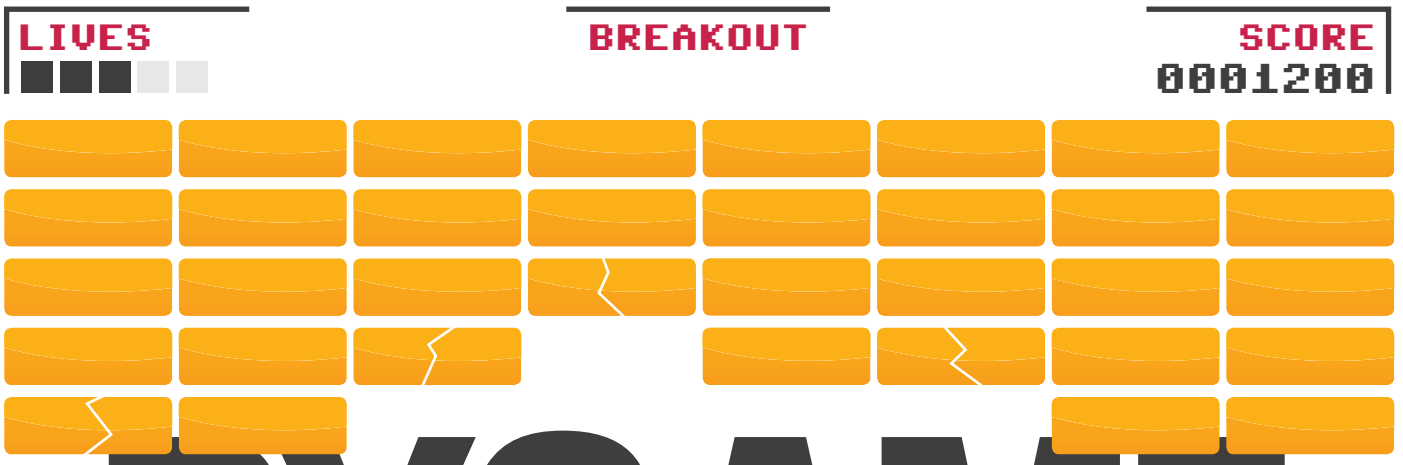
Frederick Vandenbosch is an integration engineer by day and builder of electronic things by night. He likes to build things as a way of learning about a subject and have fun while doing so. Not all builds are successful, but that's part of the learning process!

Like it?

If you'd like to see more of Frederick's work, check out his website: **bit.ly/1XC8uu9**. The rest of the Sci Fi Your Pi entries, including an actual Tricorder, can be found here: **bit.ly/1kekbsc**.

Further reading

Thinking of putting your own *Tron* desk together? There's a great woodworking video here from Bob Clagett at Make that shows you how to craft a coffee table with embedded LED strips, which is a great starting point: **bit.ly/1WlhVel**.



PYGAME ZERO

Pygame Zero cuts out the boilerplate to turn your ideas into games instantly, and we'll show you how

Games are a great way of understanding a language: you have a goal to work towards, and each feature you add brings more fun. However, games need libraries and modules for graphics and other essential games features. While the Pygame library made it relatively easy to make games in Python, it still brings in boilerplate code that you need before you get started – barriers to you or your kids getting started in coding.

Pygame Zero deals with all of this boilerplate code for you, aiming to get you coding games instantly. Pg0 (as we'll abbreviate it) makes sensible assumptions about what you'll need for a game – from the size of the window to importing the game library – so that you can get straight down to coding your ideas.

Pg0's creator, Daniel Pope, told us that the library "grew out of talking to teachers at Pycon UK's education track, and trying to understand that they need to get immediate results and break lessons into bite-size fragments, in order to keep a whole class up to speed."

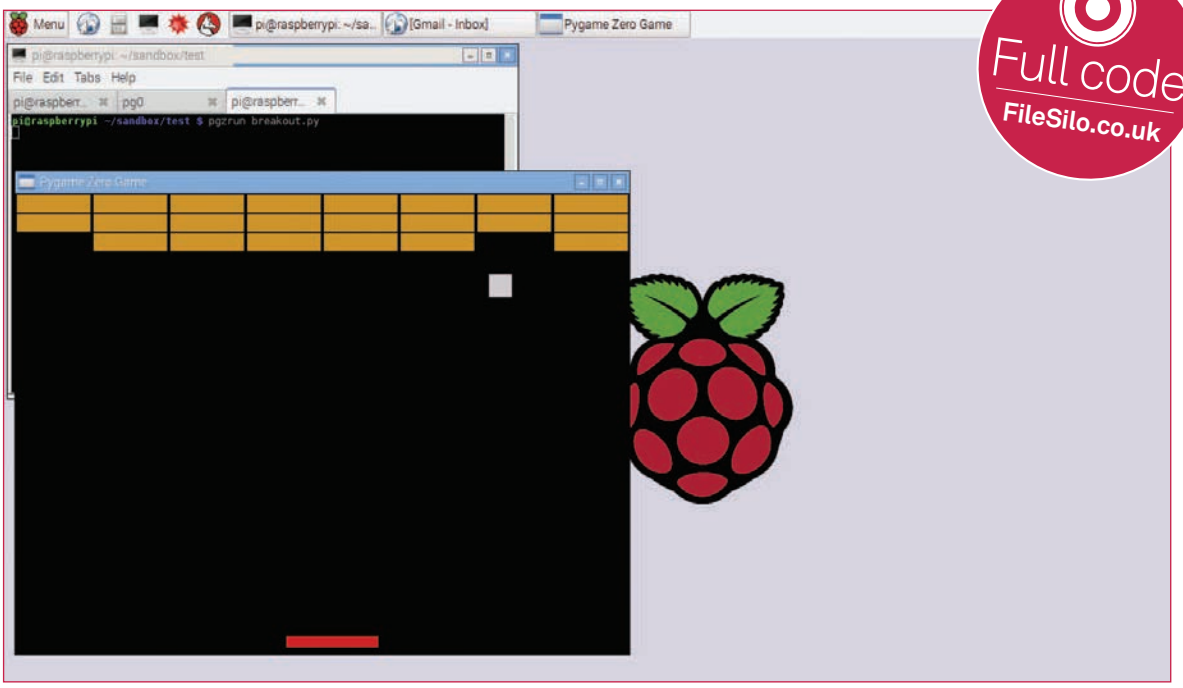
To give you an idea of what's involved, we'll build up a simple game from a *Pong*-type bat and ball through to smashing blocks *Breakout*-style. The project will illustrate what can be done with very little effort. Pg0 is in early development but still offers a great start – and is now included on the Pi in the Raspbian Jessie image. We'll look at installation on other platforms, but first let's see what magic it can perform.

What you'll need

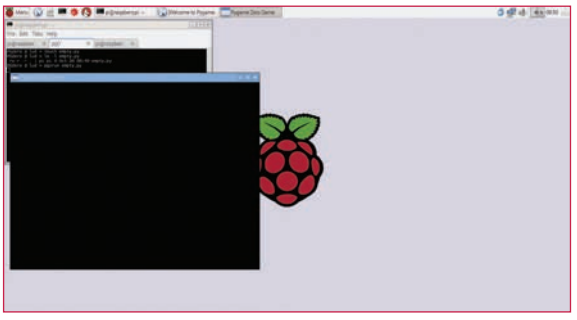
- Pygame Zero
  pygame-zero.readthedocs.org
- Pygame
  pygame.org
- Pip
  pip-installer.org
- Python 3.2 or later
  python.org

Richard Smedley is a sysadmin and writer, as well as a perpetual newbie in several programming languages. While his kids learn Python at Code Club, Richard uses Pg0 to sneak ahead of them

Full code FileSilo.co.uk



Right *Breakout* is a classic arcade game that can be reimagined in Pygame Zero



Young and old

In situations where Pygame is used boilerplate and all with young people, great results can also be achieved (see Bryson Payne's book), but Pygame and Pg0, despite their use as powerful educational tools, are also good for creating games for coders no matter what stage of learning they are at.

Great games are all about the gameplay, driven by powerful imaginations generating images, animations, sounds and journeys through game worlds. Good frameworks open up this creative activity to people who are not traditional learners of programming, which is an area where Python has long excelled.

01 Zero effort

Although game writing is not easy, getting started certainly is. If you've got Raspbian Jessie installed on your Pi, you're ready to go. Open a terminal and type:

```
touch example.py
pgzrun example.py
```

And you'll see an empty game window open (Ctrl+Q will close the window). Yes, it's that easy to get started!

02 Python 3

If you haven't got Raspbian Jessie, chances are you'll have neither Pg0 nor Pygame installed. The Python's pip package installer will take care of grabbing Pg0 for you, but the preceding steps vary by distro. One thing you will need is Python 3.2 (or newer). If you've been sticking with Python 2.x in your coding (perhaps because it's used in a tutorial you're following), make Pg0 your chance for a gentle upgrade to Python 3.

03 Older Raspbian

If you're still running Raspbian Wheezy, you'll need to run the following steps to install Pygame Zero:

```
sudo apt-get update
sudo apt-get install python3-setuptools python3-pip
sudo pip-3.2 install pgzero
```

> Pygame Zero deals with all of this boilerplate code for you, aiming to get you coding games instantly

04 No Pi?

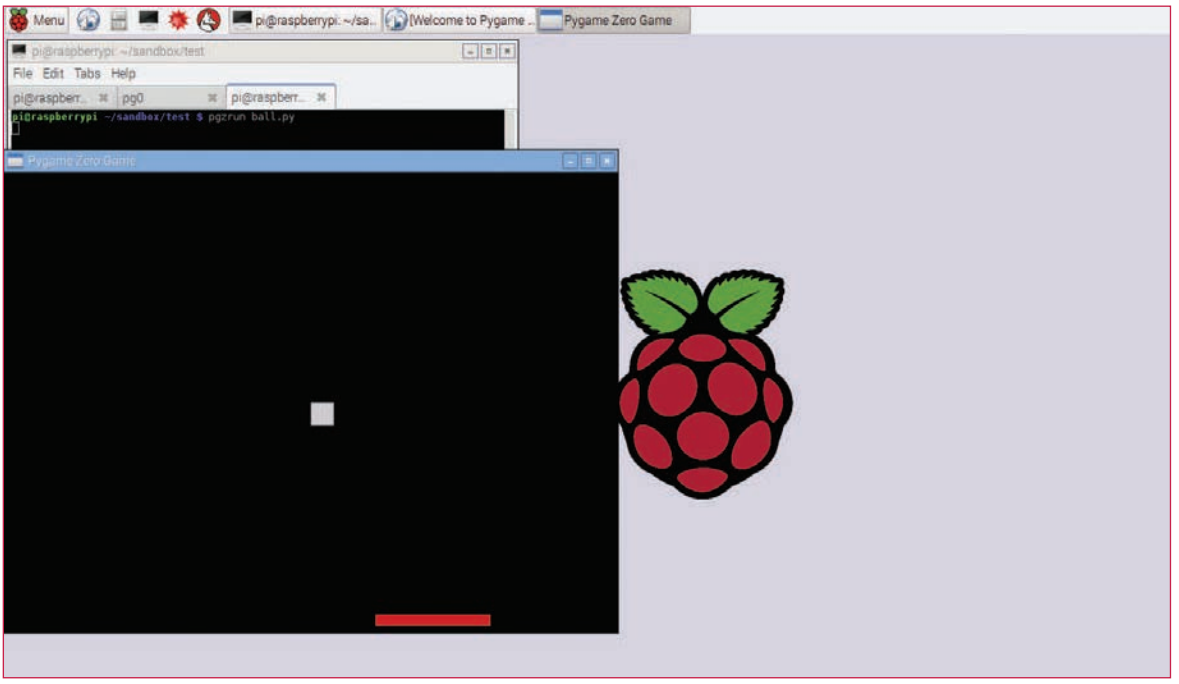
You don't even need a Raspberry Pi to install Pygame Zero – just install the Pygame library, then use pip to install Pygame Zero. Instructions vary by distro, but a good place to start is the documentation: **bit.ly/1GYznUB**.

05 Intro.py

That default black square of 800 by 600 pixels we saw in Step 1 can be overridden manually. For example, we can replace it with an oversized gold brick, in a nod to *Breakout*:

```
WIDTH = 1000
HEIGHT = 100
def draw():
    screen.fill((205, 130, 0))
```

That colour tuple takes RGB values, so you can quickly get colours off a cheatsheet; **screen** is built into Pg0 for the window display, with methods available for all sorts of different sprites...



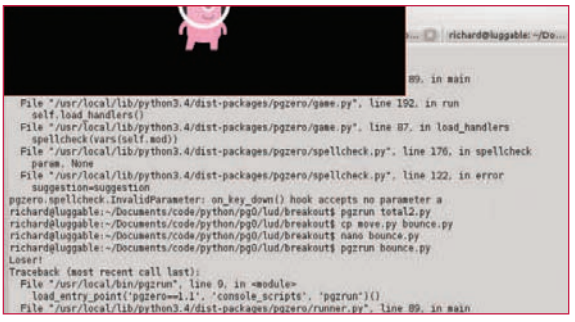
Right The bat and ball come first – they're the cornerstones of *Pong* and *Breakout*

Object orientation

David Ames, who uses Pg0 to teach younger children to code at events across the UK, told us: "One thing to avoid when it comes to teaching kids is Object Orientation." OOP (object-oriented programming) is partly abstracted away by Pg0, but it can't be ignored.

Perhaps the best approach is using Pg0 and some simple code to start, then dropping in a piece of OO when it's needed to solve a particular problem.

With the Code Club age group – about eight to eleven – feeding information to solve practical problems works well. It can work with adults, too – but there's always someone who's read ahead and has a few tricky questions.



06 Sprite

The intro example from the Pg0 docs expands on that with the **Actor** class, which will automatically load the named sprite (Pg0 will hunt around for a .jpg or .png in a subdirectory called images).

```
alien = Actor('alien')
alien.pos = 100, 56
WIDTH = 500
HEIGHT = alien.height + 20
def draw():
    screen.clear()
    alien.draw()
```

You can download the alien from the Pg0 documentation (**bit.ly/1Sm5lM7**) and try out the animation shown there, but we're taking a different approach in our game.

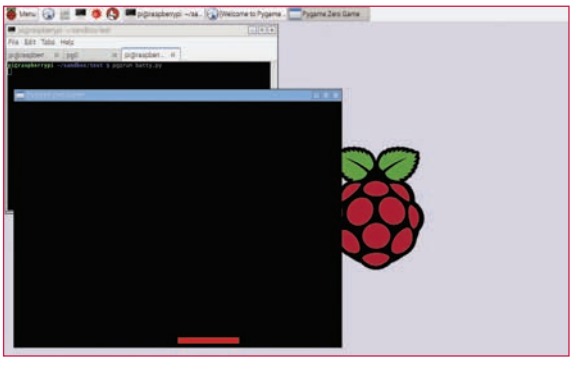
07 Breakout via Pong

While the Pi is something of a tribute to 1980s 8-bit computers, *Breakout* comes from the 1970s and is a direct descendant of the early arcade classic *Pong*. We'll follow the route from *Pong* to *Breakout* (which historically involved Apple founders Steve Wozniak and Steve Jobs) in the steps to creating our code, leaving you with the option of developing the *Pong* elements into a proper game, as well as refining the finished *Breakout* clone.

08 Batty

You can think of *Breakout* as essentially being a moving bat – that is, you're hitting a moving ball in order to knock out blocks. The bat is a rectangle, and Pygame's **Rect** objects store and manipulate rectangular areas – we use **Rect((left, top), (width, height))**, before which we define the bat colour and then call upon the **draw** function to put the bat on the screen, using the **screen** function.

```
W = 800
H = 600
RED = 200, 0, 0
bat = Rect((W/2, 0.96 * H), (150, 15))
def draw():
    screen.clear()
    screen.draw.filled_rect(bat, RED)
```



09 Mouse move

We want to move the bat, and the mouse is closer to an arcade paddle than the arrow keys. Add the following:

```
def on_mouse_move(pos):
    x, y = pos
    bat.center = (x, bat.center[1])
```

Use **pgzrun** to test that you have a screen, bat and movement.

10 Square ball

In properly retro graphics-style, we define a square ball too – another rectangle, essentially, with the (30, 30) size making it that subset of rectangles that we call a square.

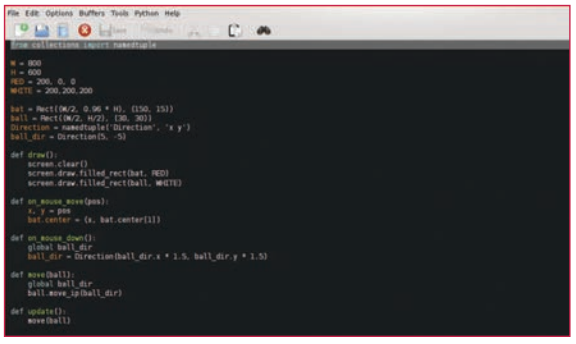
We're doing this because **Rect** is another built-in in Pg0. If we wanted a circular ball, we'd have to define a class and then use Pygame's **draw.filled_circle(pos, radius, (r, g, b))** - but **Rect** we can call directly. Simply add:

```
WHITE = 200,200,200
ball = Rect((W/2, H/2), (30, 30))
```

... to the initial variable assignments, and:

```
    screen.draw.filled_rect(ball, WHITE)
```

... to the **def draw()** block.



11 Action!

Now let's make the ball move. Download the tutorial resources in **FileSilo.co.uk** and then add the code inside the 'move.py' file to assign movement and velocity. Change the **5** in **ball_dir = Direction(5, -5)** if you want the ball slower or faster, as your processor (and dexterity) demands – but it's hard to tell now because the ball goes straight off the screen! Pg0 will call the **update()** function you define once per frame, giving the illusion of smooth(ish) scrolling if you're not running much else.

12 def move(ball)

To get the ball to move within the screen we need to define **move(ball)** for each case where the ball meets a wall. For this we use **if** statements to reverse the ball's direction at each of the boundaries. Refer to the full code listing on page 67.

Note the hardcoded value of 781 for the width of screen, minus the width of ball – it's okay to hardcode values in early versions of code, but it's the kind of thing that will need changing if your project expands. For example, a resizable screen would need a value of **W - 30**.

13 Absolute values

You might expect multiplying y by minus one to work for reversing the direction of the ball when it hits the bat:

```
ball_dir = Direction(ball_dir.x, -1 * ball_dir.y)
```

... but you actually need to use **abs**, which removes any minus signs, then minus:

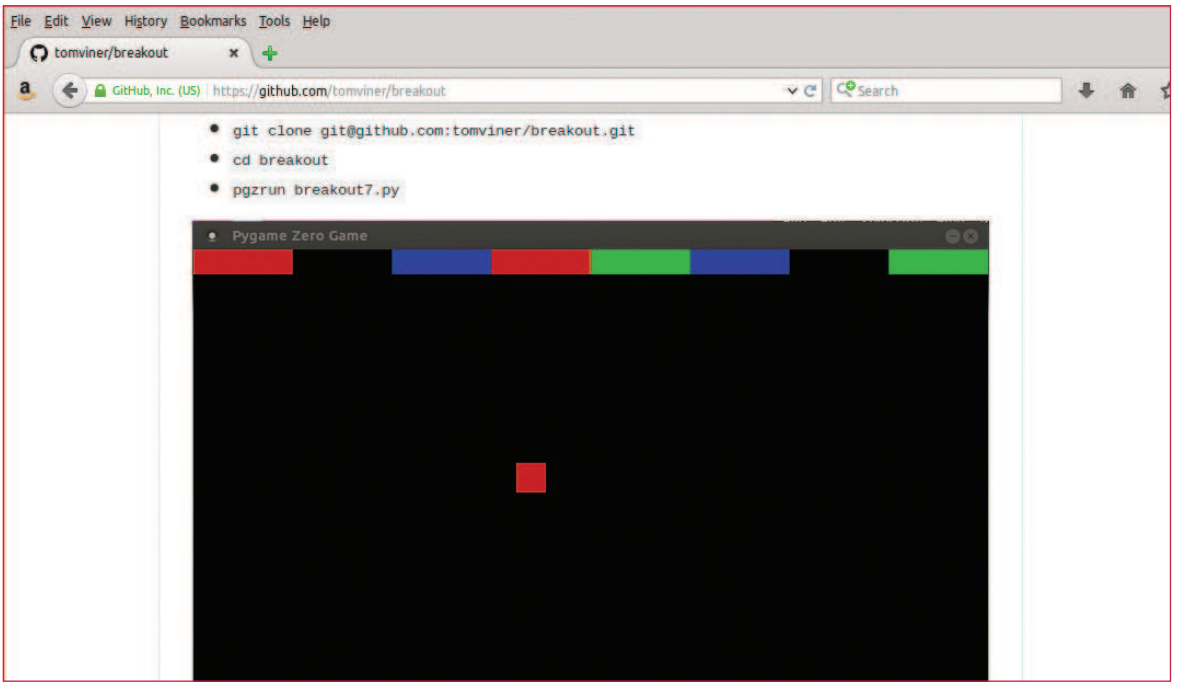
```
ball_dir = Direction(ball_dir.x, - abs(ball_dir.y))
```

Try it without in the finished code and see if you get some strange behaviour. Your homework is to work out why.

To get the ball to move we need to define **move(ball)** for each case where the ball meets a wall

Full code listing

```
## Breakout type game to demonstrate Pygame Zero library
## Based originally upon Tim Viner's London Python Dojo
## demonstration
## Licensed under MIT License - see file COPYING

from collections import namedtuple
import pygame
import sys
import time

W = 804
H = 600
RED = 200, 0, 0
WHITE = 200,200,200
GOLD = 205,145,0

ball = Rect((W/2, H/2), (30, 30))
Direction = namedtuple('Direction', 'x y')
ball_dir = Direction(5, -5)

bat = Rect((W/2, 0.96 * H), (120, 15))

class Block(Rect):

    def __init__(self, colour, rect):
        Rect.__init__(self, rect)
        self.colour = colour

blocks = []
for n_block in range(24):
    block = Block(GOLD, ((((n_block % 8)* 100) + 2, ((n_block //
                    8) * 25) + 2), (96, 23)))
    blocks.append(block)

def draw_blocks():
    for block in blocks:
        screen.draw.filled_rect(block, block.colour)

def draw():
    screen.clear()
    screen.draw.filled_rect(ball, WHITE)
    screen.draw.filled_rect(bat, RED)
    draw_blocks()

def on_mouse_move(pos):
    x, y = pos
    bat.center = (x, bat.center[1])

def on_mouse_down():
    global ball_dir
    ball_dir = Direction(ball_dir.x * 1.5, ball_dir.y * 1.5)
```



Right Tom Viner's array of blocks negates the need for bordered rectangles

14 Sounds

Also upon bat collision, **sounds.blip.play()** looks in the sounds subdirectory for a sound file called blip. You can download the sounds (and finished code) from **FileSilo.co.uk**.

Actually, now we think about it, ignore the previous comment about homework – your real homework is to turn what we've written so far into a proper game of *Pong*! But first let's finish turning it into *Breakout*!

15 Blockhead!

If you're not very familiar with the ancient computer game *Breakout*, then:

```
apt-get install lbreakout2
```

... and have a play. Now, we haven't set our sights on building something quite so ambitious in just these six pages, but we do need blocks.

16 Building blocks

There are many ways of defining blocks and distributing them onto the screen. In Tom Viner's team's version, from the London Python Dojo – which was the code that originally inspired this author to give this a go – the blocks are sized in relation to number across the screen, thus:

```
N_BLOCKS = 8
BLOCK_W = W / N_BLOCKS
BLOCK_H = BLOCK_W / 4
BLOCK_COLOURS = RED, GREEN, BLUE
```

Using multicoloured blocks which are then built into an array means that blocks can join without needing a border. With its defining variables in terms of screen width, it's good sustainable code, which will be easy to amend for different screen sizes – see **github.com/tomviner/breakout**.

However, the array of colour bricks in a single row is not enough for a full game screen, so we're going to build our array from hard-coded values...

17 Going for gold

Create a Block class:

```
class Block(Rect):

    def __init__(self, colour, rect):
        Rect.__init__(self, rect)
        self.colour = colour
```

... and pick a nice colour for your blocks:

```
GOLD = 205,145,0
```

18 Line up the blocks

This builds an array of 24 blocks, three rows of eight:

```
blocks = []
for n_block in range(24):
    block = Block(GOLD, ((((n_block % 8)* 100) + 2,
                ((n_block // 8) * 25) + 2), (96, 23)))
    blocks.append(block)
```

19 Drawing blocks

Draw_blocks() is added to **def draw()** after defining:

```
def draw_blocks():
    for block in blocks:
        screen.draw.filled_rect(block, block.colour)
```

20 Block bashing

All that remains with the blocks is to expand **def move(ball)** – to destroy a block when the ball hits it.

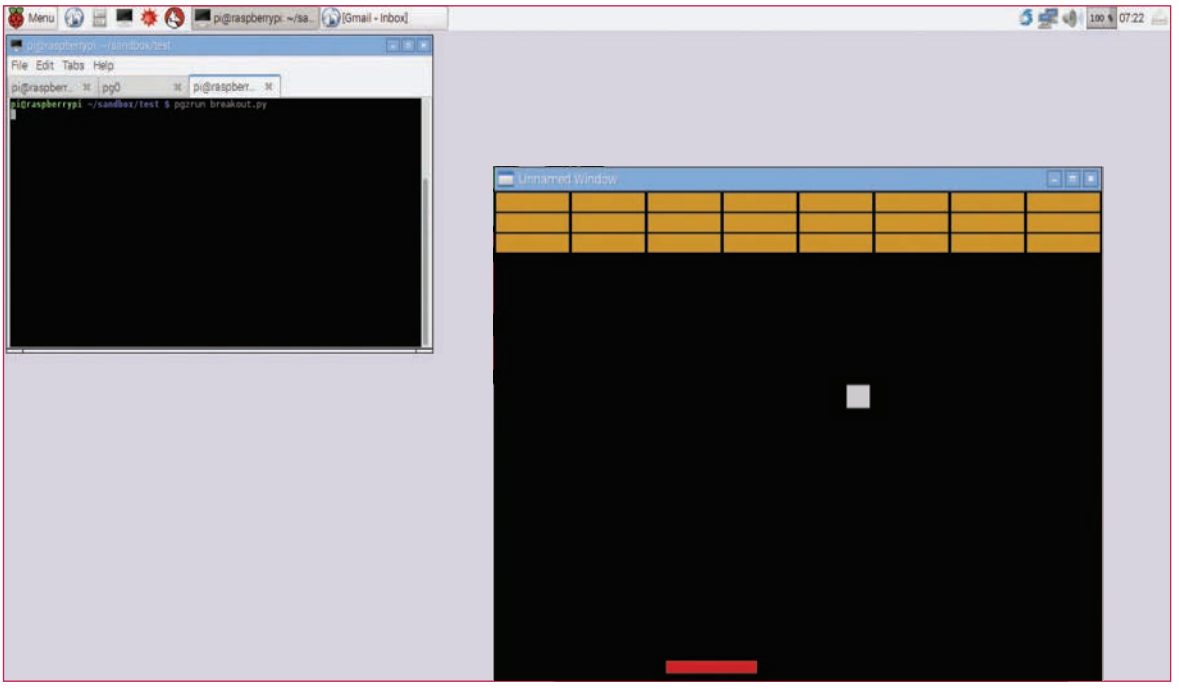
```
to_kill = ball.collidelist(blocks)

if to_kill >= 0:
    sounds.block.play()
    ball_dir = Direction(ball_dir.x, abs(ball_dir.y))
    blocks.pop(to_kill)
```

Pg0 +1

There's a new version of Pg0 in development – it may even be out as you read this. Pg0 creator Daniel Pope tells us "a tone generation API is in the works," and that at the Pg0 PyConUK sprint, "we finished Actor rotation."

Contributions are welcome – not only to the Pg0 code, but more examples are needed not just to show what can be done, but to give teachers tools to enthuse children about the creative act of programming.

Pg0 has also inspired GPIO Zero, to make GPIO programming easier on the Raspberry Pi, with rapid development occurring on this new library as we go to press.



Left Test your game once it's finished – then test other people's *Breakout* games to see how the code differs

21 Game over

Lastly, we need to allow for the possibility of successfully destroying all blocks.

```
if not blocks:
    sounds.win.play()
    sounds.win.play()
    print("Winner!")
    time.sleep(1)
    sys.exit()
```

22 Score draw

Taking advantage of some of Pygame Zero's quickstart features, we've a working game in around 60 lines of code. From here, there's more Pg0 to explore, but a look into Pygame unmediated by the Pg0 wrapper is your next step but one.

First refactor the code; there's plenty of room for improvement – see the example 'breakout-refactored.py' in your tutorial resources. Try adding scoring, the most significant absence in the game. You could try using a global variable and writing the score to the terminal with **print()**, or instead use **screen.blit** to put it on the game screen. Future versions of Pg0 might do more for easy score keeping.

23 Class of nine lives

For adding lives, more layers, and an easier life-keeping score, you may be better defining the class **GameClass** and enclosing much of the changes you wish to persist within it, such as **self.score** and **self.level**. You'll find a lot of Pygame code online doing this, but you can also find Pg0 examples, such as the excellent pi_lander example by Tim Martin: **github.com/timboe/pi_lander**.

24 Don't stop here

This piece is aimed at beginners, so don't expect to understand everything! Change the code and see what works, borrow code from elsewhere to add in, and read even more code. Keep doing that, then try a project of your own – and let us know how you get on. ■

Full code listing (cont.)

```
def move(ball):
    global ball_dir
    ball.move_ip(ball_dir)

    if ball.x > 781 or ball.x <= 0:
        ball_dir = Direction(-1 * ball_dir.x, ball_dir.y)
    if ball.y <= 0:
        ball_dir = Direction(ball_dir.x, abs(ball_dir.y))

    if ball.colliderect(bat):
        sounds.blip.play()
        ball_dir = Direction(ball_dir.x, - abs(ball_dir.y))

    to_kill = ball.collidelist(blocks)

    if to_kill >= 0:
        sounds.block.play()
        ball_dir = Direction(ball_dir.x, abs(ball_dir.y))
        blocks.pop(to_kill)

    if not blocks:
        sounds.win.play()
        sounds.win.play()
        print("Winner!")
        time.sleep(1)
        sys.exit()

    if ball.y > H:
        sounds.die.play()
        print("Loser!")
        time.sleep(1)
        sys.exit()

def update():
    move(ball)
```

Add multimedia to your Pi

You can use separate specialised modules to handle multimedia, but using the pygame module is a one-stop fix

Joey Bernard is a true renaissance man, splitting his time between building furniture, helping researchers with scientific computing problems and writing Android apps

Why Python?

It's the official language of the Raspberry Pi. Read the docs at **python.org/doc**

There may be many times when you want to use multimedia within your interactions with an end user. You may want to display a really interesting bit of video or have your program chat audibly with the user. In both of these cases, there are lots of modules available to enable you to work all of these other forms of data. However, a truism about programming is to 'find the easiest way to do something'. Here, this means if you can find a single framework that provides everything you need, you should probably use it. This is the role that the module pygame fills. This isn't to say that you will be able to do everything you may want to do using pygame, but you should never undersell the worth of your time in learning and using multiple modules and the way multiple development groups think in their coding styles. This article will cover at least some of the various methods and techniques you can use to interact with your users in some form other than text.

> In the most sensitive code sections, optimised assembly language is used

Pygame is written to be as portable as possible. In order to do this, it supports many different graphic backends, from OpenGL to even using ASCII text to display graphics. It is designed to use all of the computing resources, ie CPUs and GPUs, available to do the type of calculations required when playing with these types of files. In order to get the best performance out of your hardware, several of the heaviest functions are written in optimised C. This gets you a 10- to 20-fold speed increase. In the most sensitive code sections, optimised assembly language is used to get even more performance. This means that even your Raspberry Pi can get pretty impressive output with its limited resources.

Installing pygame can be pretty messy. Depending on which video and audio back-ends are to be supported, you need to have all of the optimised C and assembler code we mentioned compiled. For your Raspberry Pi, the simplest solution is to use **sudo apt-get install python-pygame** to deal with all of the potential headaches. The documentation available at the main home page (**pygame.org**) is very extensive. There are several tutorials available and there is even a recipe section where you can find recipes outlining techniques for specific effects and interactions.

The first step is to import the pygame module. Pygame provides a complete multimedia subsystem, so you need to call **pygame.init()** before being able to use any of the functionality it provides. This initialises everything and loads all of the module contents for you. We will start by looking at displaying video to your users. The first thing you need is somewhere to do the actual displaying of your video. When you called **pygame.init()**, the pygame subsystem should have created a display object with the best display mode available on your current device. You may want something else in order to make your program behave the way you want it to. You can create a new graphic display with the function **pygame.display.set_mode()**, where you can give the function a sequence with a width and a height value. The actual object that you draw images on is called a surface in pygame. The display that you just created has a surface object associated with it. Anything that you draw on this surface is displayed immediately on the screen. However, this is usually not what you do. The usual technique is to create an off-screen drawing area and do your drawing there. Then you can dump the entire contents of the image onto the actual display in one movement. This is much more efficient than trying to draw on the screen fast enough to look good to the user. This dumping of image data to the screen is handled with the **blit()** function of the surface object. This takes data from one surface and moves it to a location on the second surface. In order to get this surface displayed on the physical monitor, you need to call the **flip()** function. This function takes the entire surface and dumps it into the physical hardware as the last step for display.

The first way to draw we will look at is to load an entire image file into Python. The function **pygame.image.load()** will read in an image file and load it into a new surface object. By default, pygame can only load uncompressed BMP files. The load function includes a translation layer that can handle converting and importing several other formats, such as JPEG, GIF, PNG and others. For some of these other formats, you may wish to call the image's **convert()** method to convert it to the natural format. It will then be faster to use. When you wish to save images, your options are more limited as you can only save to the formats BMP, TGA, PNG and JPEG. If you need a different format, you can always convert it after it has been saved by pygame.

Whether you are starting with a loaded image or with a blank surface, you will probably want to be able to draw on it. There is a whole suite of drawing functions under the **pygame.image** namespace that gives you that functionality. The first parameter in all of these functions is a surface on which to draw. This surface could be a previously loaded image or a blank canvas. For example, you could draw a circle with the function **pygame.image.circle(Surface, color, pos, radius, width=0)**. This function draws a circle with the given radius at the position given on the surface object. The width parameter tells pygame how wide to make the circumference of your circle. If the width is set to zero, then that means you want a solid, filled-in circle drawn.

This function will also return a rectangle object that represents the bounding rectangle of the pixels that changed. This is important if you are doing animation and need to track sprites as they move around. There are also functions to draw polygons, lines and rectangles, among other shapes.

Once you have finished all of the graphics you want to do, you will need to call **pygame.quit()**. This function cleans up all of the shared memory and graphics devices that had been created. You only need to worry about this if your program is going to continue doing work after you are done with the graphical display items. Otherwise, everything should be cleaned up properly when the Python interpreter exits at the end of your program run.

The other way of interacting with the users of your program is through audio. The required functionality is provided through **pygame.mixer**. Just as with the image system, the audio system must be initialised before you can use it. This is handled by the function **pygame.mixer.init(frequency=22050, size=-16, channels=2, buffer=4096)**, where the values shown here are the defaults. The first parameter gives the sampling frequency for the audio; the second gives the number of bits used to hold each sample; the third sets whether the audio is mono (**channels=1**) or stereo (**channels=2**); the last parameter sets the size of the internal buffer used by the mixer for audio processing. You can create a new sound object from an audio file with the function **pygame.mixer.Sound(filename)**. This function loads the entire audio file into a buffer. This may be an issue if you are dealing with larger files, such as music files. In these cases, you can use a streaming function like **pygame.mixer.music.load(filename)**. Both of these groups – **pygame.mixer** and **pygame.mixer.music** – contain several functions that help you manipulate audio data. The first thing to do is to start playing your audio file. The returned sound object from **pygame.mixer.Sound()** has a play method that starts the audio playing on the next available channel. The mixer portion of pygame can handle eight audio channels, so you can have multiple audio files playing at the same time. If you want to play audio on a specific channel, you can create a new channel object tied to a specific channel ID and then play specific sound objects on them. There are helper functions that can get and set the current volume for a playing sound object. You can also pause and play them. When you want things to go quiet, you can either call **stop()** to end it quickly or **fadeout(time)** where the sound will fade out over 'time' milliseconds before stopping. Just as with the image system, you need to call **pygame.mixer.quit()** to cleanly shut down the audio system and release all of the resources that were used for your sound playback.

The two previous sections describe how you can produce output for your end users, but how can you get input from them? Pygame can help you out here as well. You can get image information using the **pygame.camera.Camera** object. The physical camera you want to use needs to be recognised by the operating system before you can use it. Once it is, you can get a list of all available cameras with the function **pygame.camera.list_cameras()**. Once you have selected a camera to use, you can create a new camera object that is tied to it with the constructor **pygame.camera.Camera()**. The camera object's method **start()** connects to the camera, initialises it and then starts to record images from it. These images are placed into a buffer for further use. When you are ready to work with these images, you can start to pull them from this buffer with the function **get_image()**. This function returns the retrieved image into a new surface object. If you want to reuse an existing surface, you hand it in as a parameter to the function call. Unfortunately, pygame isn't able to record audio directly. If you want to do this, you will need to use a different module in your program, such as pymedia.

Now that we've covered the basics, you should be able to use pygame to add multimedia functionality to your next Python project on your Raspberry Pi.

What about movies?

While playing audio and presenting images is all well and good, sometimes a video is the clearest way to display output. Luckily, pygame can help you with this, too. There is a movie object available that can handle the playback of MPEG-1 video files. The constructor takes a filename for the movie file: **pygame.movie.Movie(filename)**. You can check the data and see whether there is video data or audio data with the helper functions **has_audio()** and **has_video()**.

Movie objects do need to be displayed on some kind of surface. The default surface that the movie object will use is the display surface. If this isn't to your liking, you can change this to a different surface with the method **set_display()**. This draws over everything on the surface, so you may want to actually draw on an off-screen surface and **blit()** it to the display surface once each frame.

We have to warn you that when it comes to audio, you do need to be a bit careful. The movie object expects to have total control over the audio on your Raspberry Pi, so you need to be sure that you don't have the mixer running. If there is a chance that it might be running then you can always guarantee that you don't have any conflicts by calling **pygame.mixer.quit()** before you start playing your movie. You can change the playback volume with the method **set_volume()**, giving a value between 0.0 and 1.0.

Once the file is loaded, you can start playback with the method **play()** and stop it again with the method **stop()**. If you want to just temporarily stop the playback, you can use the **pause()** method and then restart it by calling upon the pause method again. If you want to start playback at some fixed point, you can use the **skip()** method to jump ahead a certain number of seconds. However, please be aware that you can only jump forward. If you need to move backwards, you can instead use the **rewind()** method to move back to the beginning and then call **skip()** from there to jump forward to the desired point. You can also move around by selecting a specific frame to move to, but this might be a tad difficult to calculate. You can use the method **get_frame()** to get the current frame number, then use that as a starting point, adjusting the value and using **render_frame(id)** to move the video to the frame you want to display.

Hack a robot with Pi-Mote

Dan Aldred
is a Raspberry Pi Certified Educator and a Lead School teacher for CAS. He recently led a winning team of the Astro Pi secondary school contest

Code a program to control a toy robot via your Raspberry Pi and the Energenie IR board

In the previous issue (158) we looked at how to create a hack to control your TV using the Energenie Pi-Mote IR board and a Raspberry Pi. We return to the world of the IR board, only this time we use it to record signals from a remote control robotic toy and combine these with pygame commands to enable you to control your robot from your Raspberry Pi. Since the code is Python-based, you can then develop it further, use social media to control the robot with tweets, or install Flask to create a web interface accessible from your mobile phone.

01 Update the boot file

Before booting up your Raspberry Pi, attach the Energenie IR board to GPIO pins. The board slots onto the top of the pins, the default GPIO pins used when no pins are explicitly set are: input pin for received infrared signal = PIN12/GPIO18; output pin for transmitted infrared signal = PIN11/GPIO17. You are advised to use a fresh SD card image with no other software installed to reduce any software conflicts.

Boot up your Raspberry Pi and, in the LX Terminal, type:

```
sudo apt-get update
sudo apt-get upgrade
```

Then add a line of code to the /boot/config.txt file to enable the LIRC IR software and IR module to interact. In the LX Terminal, type **sudo nano /boot/config.txt**. This loads the config .txt file. Scroll to the bottom of the text and add the following line:

```
dtoverlay=lirc-rpi-overlay
```

Press Control and X to save the file, then reboot your Pi by typing **sudo reboot**.

What you'll need

- Pi-Mote IR control board
  bit.ly/1MdpFOU
- Remote controlled robot

Left The Pi-Mote IR control board is hugely versatilse and only costs £9.99

02 Install the software

Next, install the LIRC software; this stands for Linux Infrared Remote Control and is the program that enables you to interact with your robot and transmit commands. Open up the the LX Terminal again and type:

```
sudo apt-get install lirc
sudo apt-get install lirc-x
```

Once the installation has finished, you need to restart your Pi using **sudo reboot**.

03 Edit the hardware file

Next, we need to edit the hardware.conf file located in the /etc/lirc/ folder and make the changes shown below. In the terminal window, type:

```
sudo nano /etc/lirc/hardware.conf
```

Find the DRIVER, DEVICE and MODULES lines in the file, then make the following changes:

```
DRIVER = "default"
DEVICE = "/dev/lirc0"
MODULES = "lirc_rpi"
```

Press Ctrl+X to save the file, don't rename it, press Y and then return. Restart the LIRC daemon with the command **sudo /etc/init.d/lirc restart**.

04 Test the IR receiver is working

To test that the IR receiver is installed and working correctly, you need to head to the LX Terminal and then stop the LIRC daemon (line 1), enable the test mode (line 2) and then start the mode 2 testing (line 3):

```
sudo /etc/init.d/lirc stop
sudo modprobe lirc_rpi
sudo mode2 -d /dev/lirc0
```

This runs a program to output the mark-space of the IR signal. It measures the pulse and space length of infrared signals, returning the values to the terminal. Grab the robot's remote control, point it at the IR receiver and then press some buttons. You should see something like this:

```
space 16300
pulse 95
space 28794
pulse 80
space 19395
space 28794
pulse 80
```

05 Make the lircd.conf file: part 1

The lircd.conf file contains the code or instructions to control your device. Although you can find these online, you'll probably have to create one from scratch. This involves running the **irrecord** program, pointing your remote at the IR board and then pressing buttons! This records the signals from your remote and then you assign KEYS to each signal to transmit the signal to the robot. Stop the LIRC software by typing this into the terminal:

```
sudo /etc/init.d/lirc stop
```

06 Make the lirc.conf file: part 2

Next create your new lircd.conf configuration file and save the output. In the LX Terminal, type:

```
irrecord -d /dev/lirc0 ~/lircd.conf
```

This will open the program which will prompt you with instructions on how to record the signals from your remote. The first part involves you repeatedly pressing the buttons on the remote until there are two lines of dots on the screen. This measures and records the signals being sent from the remote. Do this in a logical order starting at the top of the remote and working downwards. Once the two lines of dots have been completed, your remote has been recognised.

Pygame buttons

Pygame enables you to create buttons and assign actions to them. This means you can create and assign buttons to the pygame window. You could control the robot with buttons instead of keys. There is a simple tutorial here: **pygame.org/project-Button+drawer-2541-.html**, which covers how to create and assign some interaction to the various buttons.

Above The infrared receiver on your robot will look something like this

Robot variation

You can use a bunch of different robots with this tutorial, such as the Makeblock that's pictured or the WowWee Roboquad that our author used. Pay attention to the event key setup in your code – different robots will have different native functions. The Roboquad, for example, has a 'Dance' command that you can see about halfway down the code listing on the opposite page.

07 Make the lirc.conf file: part 3

The second part of the program asks you to enter the names of the keys for each of the signals it has recorded. Follow each of the on-screen prompts, typing a suitable name for each of the remote buttons/ keys. For example, type 'KEY_UP' and then press the corresponding up key on the remote. You will then be prompted to type in the name of the next key – for example, 'KEY_BACK', in which case you would press the back key on the remote, and so on. Keep repeating this process until you have entered names for each of the recorded keys.

08 Rename the remote

Once saved, locate the new lircd.conf file in the /home/pi folder. By default, the name of the remote on line 14 will probably be named as /home/pi/lircd.conf. Change this: under the heading 'begin remote', find the 'name' label and rename /home/pi/lircd.conf to something more appropriate, such as 'Robot'. Doing this helps ensure that it is an easier process to refer to it in the program whenever you call a movement instruction or key name.

09 Transfer the lircd.conf file

Now your lircd.conf file is ready to transfer to the /etc/lirc folder; this is the folder that holds the hardware and lirc files. The simplest method is to open your new lircd.conf, which is saved into the /pi/home folder, and then copy and paste over the code that you have just created. However, this will overwrite any old configuration file setup that you have. To save a previous file, see the next step. In the LX Terminal, type:

```
sudo nano /etc/lirc/lircd.conf
```

10 Transfer file without overwriting

If you have set up a lircd.conf file, or want to use a new one but keep the old one, create a new configuration file. This is saved in the /home/pi folder and can be copied over to /etc/lirc. Make a backup of the original lircd.conf file by creating a copy of it, then save it as lircd_orginal.conf. In the LX Terminal, type:

```
sudo /etc/init.d/lirc start
sudo mv /etc/lirc/lircd.conf /etc/lirc/lircd_
original.conf
```

Then copy over your new configuration file:

```
sudo cp ~/lircd.conf /etc/lirc/lircd.conf
```

Your original configuration file will now be saved as lircd_original.conf.

11 Start to take control

Now your robot has a configuration file, you can use your remote control. Restart the LIRC with **sudo /etc/init.d/lirc restart**. Now test that the lircd file is working by listing all the registered KEYS stored in the file: **irsend LIST Robot " "**. This will list all the KEYS that are recorded in the lircd.conf file.

It's time to control your device. This uses the irsend application that comes with LIRC to send the commands. The commands are very simple: **irsend SEND_ONCE Remote_Name Remote_Button**. For example, to make the robot walk forward, just type this into the LX Terminal: **irsend SEND_ONCE Robot KEY_UP**. This will send the 'forward' infrared signal and your robot will then walk forward.

12 Python OS

The LIRC program enables you to control the Pi-Mote IR control board via the command line. However, this is impractical for coding purposes because it limits the interactions with other hardware and software. However, good-old Python has an OS module that enables you to control the command line and execute command line instructions from within a Python program. This enables you to create a program that can be controlled by pygame, which means you can move the robot with the arrow keys and keyboard.

13 Using Python OS

Open your Python editor and then import the OS module: **import os**. Now enter the command: **os.system("irsend SEND_ONCE Robot KEY_RIGHT")**. This will permit interaction between Python code and the Raspberry Pi's operating system. The code line will send the 'move right' command to the IR board and it will also transmit the assigned signal, instructing the robot to walk to the right. Test that all of the other movements work by changing the related 'KEY' name at the end of the line of code.

14 Get familiar with pygame

Pygame is a cross-platform library that was created to enable users to produce simple video games. It includes a number of graphics and sound libraries that have been specifically designed to be used with the Python programming language. Pygame also comes preinstalled on the Raspberry Pi. You can read more about the codes and creating games over in the official documentation: **pygame.org/docs**.

15 Not overwriting

Pygame runs in a window that has been preset by the user. Before adding the Python code to control the robot, you are going to have to set up the pygame structure. Although this program makes use of the controls on your keyboard, you still need to create the traditional pygame window. See the first part of the listing to the right, from the first line down to **runGame()**.

Set up the window dimensions first (lines 7 and 8). Next, initialise the pygame clock: **FPSCLOCK = pygame.time.Clock()**. The font size and typeface of the caption used in the window are next: **BASICFONT = pygame.font.Font('freesansbold.ttf', 18)**. Finally, set the caption for the window: **pygame.display.set_caption('ROBOT')**.

16 Restart the LIRC

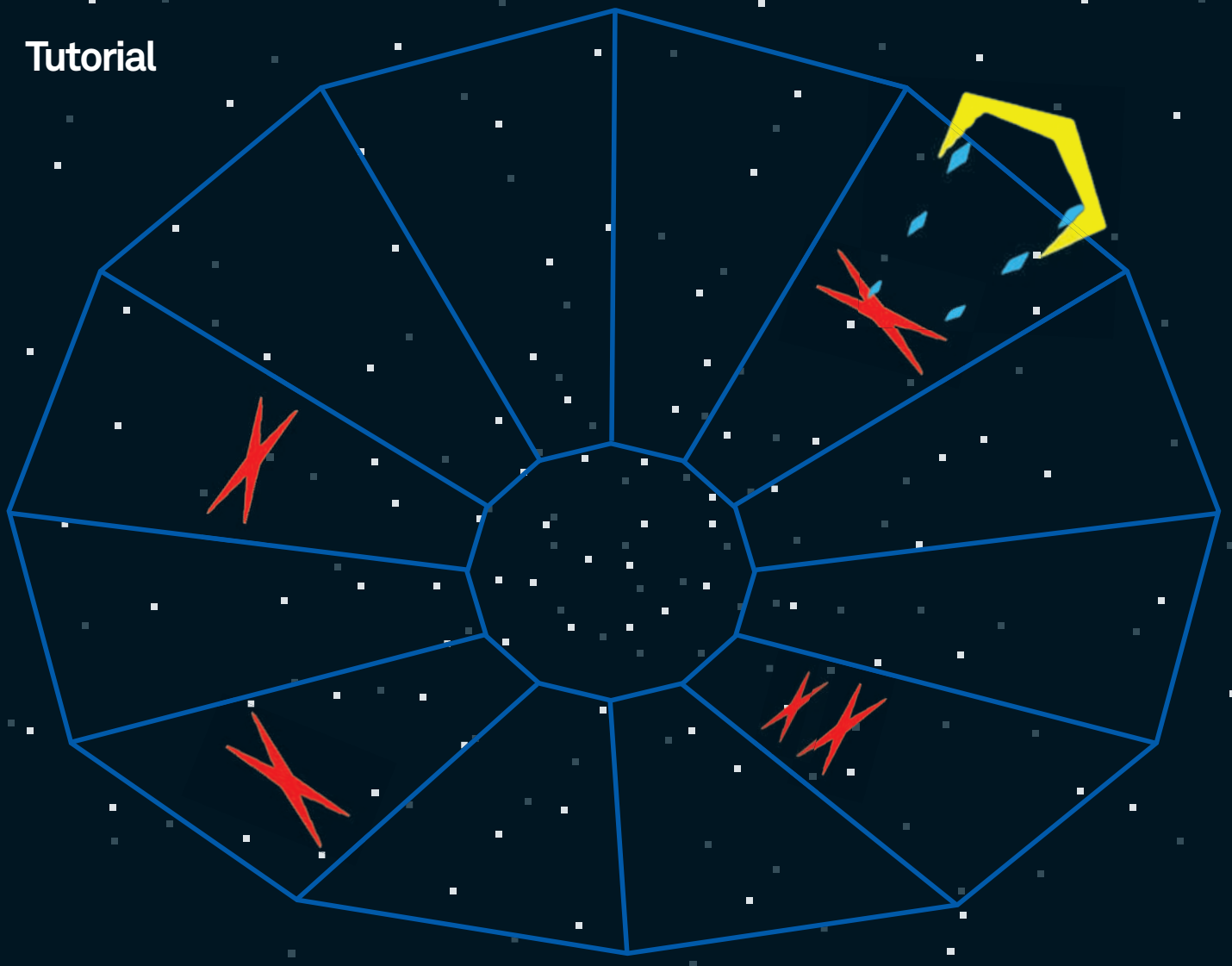
For this bit, refer to the next section of code: from **def runGame()** onwards. Once the game is initialised, you can set the code to control your robot. This makes use of the **get()** pygame event (see the **for** loop) to return the key that has been pressed. The first **elif** sets the event type to wait for 'keyboard' presses, such as the 'right' key shown in the next line. Since the OS has been imported, we use **os.system("irsend SEND_ONCE Robot KEY_RIGHT")** to transmit the 'move right' signal. When you are running the program, ensure that you have selected the pygame window – this is small, as we have coded it to 100 x 100 pixels. Once working, add the rest of your movements in the same way.

17 Common errors and code recap

Now that you can use Python code to control the robot, you can interface with a number of other modules. You could attach a Makey Makey (**makeymakey.com**) and use some spoons and forks to control your robot. You could even try installing and using the Tweepy module (a Twitter API) to make the robot respond to incoming tweets. ■

Full code listing

```
import random, pygame, sys
from pygame.locals import *
import os
import time

FPS = 15
WINDOWWIDTH = 100
WINDOWHEIGHT = 100

UP = 'up'
DOWN = 'down'
LEFT = 'left'
RIGHT = 'right'

def main():
    global FPSCLOCK, DISPLAYSURF, BASICFONT

    pygame.init()
    FPSCLOCK = pygame.time.Clock()
    DISPLAYSURF = pygame.display.set_mode((WINDOWWIDTH, WINDOWHEIGHT))
    BASICFONT = pygame.font.Font('freesansbold.ttf', 18)
    pygame.display.set_caption('ROBOT')

    while True:
        runGame()

def runGame():
    # Set a random start point.
    while True: # main game loop
        for event in pygame.event.get(): # event handling loop
            if event.type == QUIT:
                terminate()

            elif event.type == KEYDOWN:
                if event.key == K_RIGHT:
                    print "right"
                    os.system("irsend SEND_ONCE Robot KEY_RIGHT")

                elif event.key == K_LEFT:
                    print "left"
                    os.system("irsend SEND_ONCE Robot KEY_LEFT")

                elif event.key == K_UP:
                    print "Forward"
                    os.system("irsend SEND_ONCE Robot KEY_UP")

                elif event.key == K_DOWN:
                    print "Back"
                    os.system("irsend SEND_ONCE Robot KEY_DOWN")

                elif event.key == K_SPACE:
                    print "STOP"
                    os.system("irsend SEND_ONCE Robot KEY_STOP")

                elif event.key == K_d:
                    print "Dance Baby"
                    os.system("irsend SEND_ONCE Robot KEY_D")

                elif event.key == K_y:
                    print "Say Yes"
                    os.system("irsend SEND_ONCE Robot KEY_R")

                elif event.key == K_l:
                    print "Say Yes"
                    os.system("irsend SEND_ONCE Robot KEY_L")

                elif event.key == K_n:
                    print "Say No"
                    os.system("irsend SEND_ONCE Robot KEY_P")

                elif event.key == K_ESCAPE:
                    terminate()

if __name__ == '__main__':
    main()
```



Code a Tempest clone in FUZE BASIC Part 1

Luke Mulcahy is the FUZE team's in-house programmer. Just 15 years old, Luke is adept in many programming languages, though he does love to put FUZE BASIC through its paces

Remake a classic game in FUZE BASIC and delve into the world of programming

Welcome to our latest FUZE BASIC tutorial. If you haven't already gotten hold of this programming language, FUZE BASIC can be downloaded for free for both Linux and Raspberry Pi users from **fuze.co.uk/getfuzebasic**.

So why should you bother to learn BASIC? Well, BASIC (Beginner's All-purpose Symbolic Instruction Code) was instrumental in starting the computing revolution back in the Seventies and Eighties. Thirty years on and BASIC, or in this case FUZE BASIC, presents a modernised version of the original classic. Now with updated commands for advanced sprite and media handling, the removal of **goto** and **gosub** commands as well as line numbers, and the fact that on modern hardware it races along, it still remains one of the easiest introductions for users wanting to jump into the world of programming.

To prove just how straightforward it is to use, in this tutorial we're going to be looking at programming a classic arcade game. Then, over the next two issues we'll take you from this basic graphics engine to introduce enemies, firing and collisions, and then finally refinements such as an intro screen and hi-score tables, plus some tidy-ups and optimisations.

Our game, '73MP357', is a remake of a true classic – Dave Theurer's *Tempest* by Atari way back in 1981. We also couldn't resist paying tribute to the incredible remake by Jeff Minter for the Atari Jaguar.

What you'll need

- FUZE BASIC V3
  fuze.co.uk/getfuzebasic
- 73MP357PART1.fuze

01 Get the program listing

As this is a fairly large program, we won't be typing it line by line. Instead, you need to download the Part 1 code from **FileSilo.co.uk** or **fuze.co.uk/tutorials/73MP357PART1.fuze**. We will then look at each section and explain what is happening.

Open FUZE BASIC and either load (with F8) '73MP357' or copy and paste the code straight into the editor (F2 switches between the editor and immediate mode).

02 Try the code

Before we take a look at the code, let's see what it does. Run the program by pressing F3 or typing **run** in immediate mode. You'll be presented with three core events: the star field, the level and, of course, the player. You can move the player around the level using left and right cursors.

Notice the playing field view perspective changes as you rotate around. This particular effect is from *Tempest 2000* and not the arcade original.

Right, now to the code. Press Escape to stop the program and then F2 to bring up the editor.

03 System set up

You'll see that the code is well commented, with each section headlined with a single **#** and comment.

Our first section sets up our system and variables. We set the resolution and **updatemode** is set to zero so that nothing updates the screen other than an **update** command. In FUZE BASIC, if the mode is set to 1 or 2 then **print** statements will also update the screen, which slows everything down.

Three variable arrays are set up to store the star angle, speed and distance. These could be combined into one array but it is easier to read when separated.

The **while** loop fills the star field arrays with randomly positioned points with random angles and speeds.

04 See to the display

Next up are the variables required for the playing field level display. The **radius** and **radius2** variables define the outer and inner rings, and **vertices** sets up the number of sections our level is divided into. The minimum is four for a square, with a maximum of up to 13. The **gap** variable determines the step distance around the circumference so the player will always be positioned in the middle of each section.

Full code listing

Step 03

```
###### settings #######
xres = 840
yres = 640
updatemode = 0
setmode( xres, yres )

# variables for the star field effect
dim stars( 1000 )
dim starsSpeed( 1000 )
dim starsDist( 1000 )
starNum = 0

# setup star field effect
while starNum < 1000 cycle
  stars( starNum ) = rnd( 360 )
  starsSpeed( starNum ) = rnd( 5 ) + 1
  starsDist( starNum  ) = rnd( gheight )
  starNum = starNum + 1
repeat
```

Step 04

```
# level drawing variables
x = 0
y = 0
x2 = 0
y2 = 0
oldX = 0
oldY = 0
oldX2 = 0
oldY2 = 0
angle = 0
radius = gheight / 2 - 50 // outer radius
radius2 = gheight / 16 // inner radius
vertices = rnd( 9 ) + 4 // nothing less than a square
gap = 360 / vertices // angle between vertices
```

Step 05

```
# player variables
pAngle = gap / 2
pX = 0
pY = 0
pTurn = 0

# different centers for the parallax effect
centerX = gwidth / 2 // center of close objects
centerY = gheight / 2
centerX2 = centerX // center of far objects
centerY2 = centerY
starsCenterX = centerX // fixed center for the stars
starsCenterY = centerY

# movement variables
moveDelay = int( 80 / vertices ) // delay after moving
moveCount = 0
```

Step 06

```
loop
  cls2

  # movement input
  if scankeyboard( scanright ) then
    if moveCount = 0 then
      pAngle = pAngle + gap
      moveCount = moveDelay
      if pAngle > 360 then
        pAngle = gap / 2
      endif
    endif
```

Full code listing

Step 06

```
endif

if scankeyboard( scanleft ) then
  if moveCount = 0 then
    pAngle = pAngle - gap
    moveCount = moveDelay
    if pAngle < 0 then
      pAngle = 360 - ( gap / 2 )
    endif
  endif
endif
```

Step 07

```
# set max particle effects based on resolution
maxStars = gheight / 5

colour = white
while starNum < maxStars cycle

  # calculate star positions using expanding or
  # contracting circles
  starX = starsDist(starNum) * cos( stars(starNum) )
  starX = starsCenterX + starX
  starY = starsDist(starNum) * sin( stars(starNum) )
  starY = starsCenterY + starY
  starsDist(starNum) = starsDist(starNum) + starsSpeed(starNum)
```

Step 08

```
  # reset position if offscreen
  if starX < 0 then
    starsDist( starNum ) = rnd( 15 ) + 5
 endif
  if starX > gwidth then
    starsDist( starNum ) = rnd( 15 ) + 5
  endif
  if starY < 0 then
    starsDist( starNum ) = rnd( 15 ) + 5
  endif
  if starY > gheight then
    starsDist( starNum ) = rnd( 15 ) + 5
  endif

  plot( starX, starY )
  starNum = starNum + 1
repeat
starNum = 0
```

Step 09

```
# Draw the level by looping & drawing a segment at a time.
colour = blue
while angle < 360 cycle
  x = radius * cos( angle )
  x = centerX + x
  y = radius * sin( angle )
  y = centerY + y
  x2 = radius2 * cos( angle )
  x2 = centerX2 + x2
  y2 = radius2 * sin( angle )
  y2 = centerY2 + y2
  line( x, y, x2, y2 )
  line( x, y, oldX, oldY )
  line( x2, y2, oldX2, oldY2 )
  oldX = x
  oldY = y
  oldX2 = x2
  oldY2 = y2
  angle = angle + gap
repeat
```

05 Player variables

A few player variables are required to determine the location on-screen and its position in relation to the level.

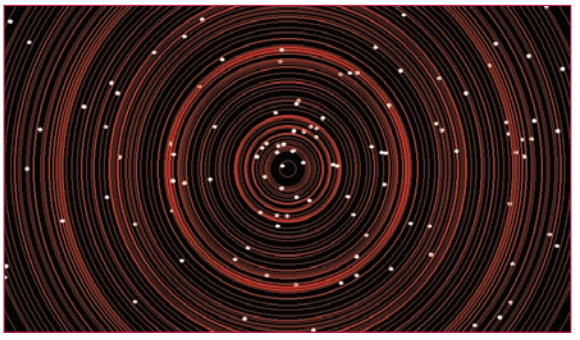
Three centre X and Y coordinates are used: two moving ones for the level and a fixed one for the star field. The moving centres will make sense a bit further on, but it is thanks to these that we have the slick perspective effects.

A **moveDelay** variable (80 divided by the number of sections, ie **vertices**) is used to keep player movement consistent regardless of how many sections there are in the level.

06 Remove flicker

Now we move onto the main **loop**. The **cls2** statement is critical and is very different to the more standard **cls** version: **cls2** wipes out the frame buffer memory; this is a separate screen memory where everything is drawn first. When you issue an **update** statement, the frame buffer is copied to the main screen memory. The **cls** version just clears everything, so it creates a lot of flicker. Both **cls2** and **update** ensure flicker-free updates, which is essential for games.

The player keyboard controls are checked next. If the right cursor is pressed then the player angle is increased by the **gap** amount (this is the distance to the centre of the next section). The left cursor, of course, does the opposite.



07 Position the stars

Then we calculate the star positions – that is each and every one of them! The number of stars is adjusted depending on the display resolution.

White is selected and a loop to count through the stars is initiated. Each star is a position on the circumference of a circle. The radius of the circle therefore determines the star's distance from the centre. We take the distance from the centre, **starsDist(starNum)**, and multiply it by the cosine of the angle, **stars(starNum)**. We then add this to the centre position, **starsCenterX**, to give the new **starX** position. The **starY** position is the same but uses the sine instead.

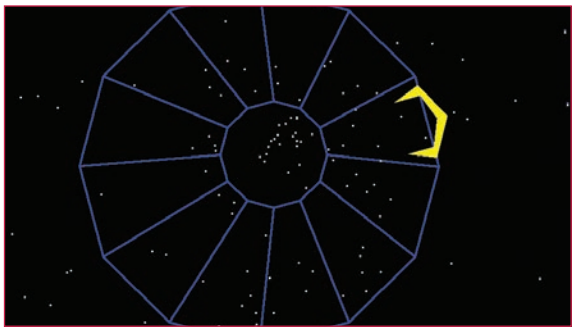
08 Check the star movement

We check to see if a star has left the display area and if so, reset it back to the centre plus a small offset, as we don't them to all appear from the dead centre.

Finally, the star is drawn using **plot(starX, starY)** and the **while** loop is repeated until all of the stars have been updated and plotted on the star field.

09 Make the level

On to the level itself. Each segment is drawn one at a time. The first line of the first segment has inner and outer X and Y coordinates. The next line along (again, from centre to outer rim) is calculated and the two connecting lines are drawn from the old positions to the new ones.



10 Place the player

Time to work out where we're drawing the player and adjust the viewpoint. The player position is calculated twice in different positions and averaged to match the polygon and parallax effects. The position of a point on a circle is calculated:

```
x = cos( angle ) * radius
y = sin( angle ) * radius
```

Sin and cos have a centre at zero so we need to add the screen centre point to the player's X and Y. The player centre is calculated from the radius multiplied by the cosine of the player angle plus the screen centre.

11 Fix a speed

The moveCount and moveDelay variables are used to fix the player's movement speed around the level regardless of how many sections there are.

The next section checks to see which quarter of the screen the player is in and moves the centre accordingly. The playing field is moved by a few pixels each frame until limited by:

```
if centerX < ( gwidth / 2 ) + ( gwidth / 20 )
```

Finally, the player's position is calculated and drawn using the polyplot command. The angle is fixed so the player faces the centre and the size is scaled against the display resolution.

12 Build a polygon

The polystart command allows a polygon to be built up from a series of lines and is automatically filled in.

Rather than work out the heavy sums within every plot in the polystart statement, we have calculated them beforehand with c = cos(pAngle + 90) and s = sin(pAngle + 90) and u is the scale of the ship against the display resolution.

13 Polygon equation

Each X point on the polygon is based on scale * line length) multiplied by cos(playerAngle + 90), minus (scale * line length) multiplied by sin(playerAngle + 90), plus playerX. For the Y coordinate, it's (scale * line length) multiplied by sin(playerAngle + 90) plus (scale * line length) multiplied by COS(playerAngle + 90) plus playerY.

14 Experiment with variables

The update statement at the very end copies the temporary frame buffer to the screen display... and voila!

There are plenty of variables to experiment with so feel free to change numbers and see what happens. Doing so will provide a great insight into what is going on.

Next month we will add enemies, firing and maybe a few sound effects, too. The Particle Laser is brilliant! ■

The number of stars is adjusted depending on the display resolution

Full code listing

Step 10

```
  # Calculate player position
  pX =  (( ( radius * cos( pAngle - ( gap / 2 ) )) + centerX ) +
( ( radius * cos( pAngle + ( gap / 2 ) ) ) + centerX ) ) / 2
  pY =  (( ( radius * sin( pAngle - ( gap / 2 ) )) + centerY ) +
( ( radius * sin( pAngle + ( gap / 2 ) ) ) + centerY ) ) / 2
  angle = 0

  if moveCount > 0 then
    moveCount = moveCount - 1
  endif
```

Step 11

```
  # move the center of closer objects more
  if pX + 2 < gwidth / 2 then // check the players position
    if centerX < ( gwidth / 2 ) + ( gwidth / 20 ) then
      centerX = centerX + 4
      centerX2 = centerX2 + 2
    endif
  endif

  if pX - 2 > gwidth / 2 then
    if centerX > ( gwidth / 2 ) - ( gwidth / 20 ) then
      centerX = centerX - 4
      centerX2 = centerX2 - 2
    endif
  endif

  if pY + 2 < gheight / 2 then
    if centerY < ( gheight / 2 ) + ( gheight / 20 ) then
      centerY = centerY + 4
      centerY2 = centerY2 + 2
    endif
  endif

  if pY - 2 > gheight / 2 then
    if centerY > ( gheight / 2 ) - ( gheight / 20 ) then
      centerY = centerY - 4
      centerY2 = centerY2 - 2
    endif
  endif
```

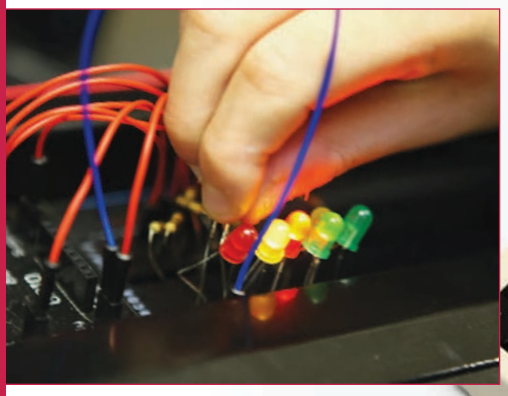
Step 12

```
  colour = yellow
  c = cos( pAngle + 90 ) // correct the view angle by 90 degrees
  s = sin( pAngle + 90 )
  u = gheight / 80 // scale the player ship to the resolution
```

Step 13

```
  # rotate & scale coordinates; draw player ship
  polystart
    polyplot((u*5 * c) - (u*5 * s) + pX, (u*5 * s) + (u*5 * c) + pY)
    polyplot((u*5 * c) + pX, (u*5 * s) + pY)
    polyplot(0 - (u*-2 * s) + pX, (u*-2 * c) + pY)
    polyplot((u*-5 * c) + pX, (u*-5 * s) + pY)
    polyplot((u*-5 * c) - (u*5 * s) + pX, (u*-5 * s) + (u*5 * c) + pY)
    polyplot((u*-7 * c) + pX, (u*-7 * s) + pY)
    polyplot(0 - (u*-4 * s) + pX, (u*-4 * c) + pY)
    polyplot((u*7 * c) + pX, (u*7 * s) + pY)
  polyend
  update
repeat
end
```

WIN! FUZE Special Edition



Closing date for entries **10 February 2016**

With decidedly retro roots, the award-winning FUZE is a programmable computer and electronics workstation. Born out of a passion for programming and love of electronics, the solid aluminium case features an integrated keyboard, a specially designed GPIO header and a breadboard for simple electronics. Housing the Raspberry Pi safely below the keyboard, the FUZE comes complete with FUZE BASIC already installed.

FUZE BASIC is an advanced, modernised version of the BASIC programming language, widely accepted as the easiest beginner language to teach and learn. Featuring a redesigned interface and advanced graphics support, including sprite and image scaling, angle and alpha controls and rotation, FUZE BASIC is fully configured to run with all models of the Raspberry Pi. The whole FUZE system is slick and intuitive, and more than capable of programming web and mobile games.

The FUZE Special Edition pays tribute to home computers of the 1980s like the BBC Micro, and comes complete with a robotic arm kit to integrate with your projects. The FUZE Special Edition also provides you with a projects pack and directs you to plenty of online resources at the FUZE Lair, giving you everything you need to get started. For more information, head over to **www.fuze.co.uk**.

Enter the competition

For a chance to win a FUZE T2-SE-R – a Special Edition with the robotic arm kit – just go to the link below. Answer the simple question provided on that webpage and then add your contact details to enter the competition. The three winners will then be announced in the New Year – good luck!

FUZE
Teaching kids to code

Answer the question on this webpage for a chance to win!

www.linuxuser.co.uk/news/win-a-fuze-se

Internet of Things?

Wireless devices from £9.99 inc VAT

wirelessthings.net/lxu-offer

@wirelessthings

PCI Express LTD

UK Distributor

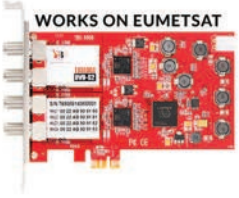
TBS is a professional manufacturer of digital TV tuners

Their products include:
DVB-S/S2 (SD Satellite & HD Satellite)
DVB-T/T2 (SD Terrestrial & HD Terrestrial)
DVB-C (SD Cable)
IPTV Streamer
USB TV Tuners

With a wide variety of choices:
Single, Dual, Quad Tuner
Common Interface (CI)
And now the all new MINI tuner (7220 below)

All TBS tuners are compatible with Windows 7 Media Center and support Linux with the latest kernel.

Get the best digital TV tuners for your PC!



TBS 6908 Professional HD DVB-S2 Quad Tuner PCIe Card

TBS6908 is a professional level digital satellite TV Tuner card with PCI Express interface. It supports not only normal DVB-S2/DVB-S QPSK, 8PSK which is supported by normal satellite receivers, but also CCM, ACM, VCM, Multi Input Stream, 16APSK, 32APSK, Generic Stream Mode , which most satellite receiving devices can't support.
With use of dedicated TBS tools, those special streams can be captured. Both Windows BDA driver and Linux driver up to the latest kernel 3.X are ready.

S.R.P £329.95

TBS 6290 DVB-T2/T/C Dual Tuner Dual CI PCIe Card

TBS6290 is the latest DVB-T2/T PCI Express tuner, it's equipped with dual tuner and dual CI slots, which allow you to enjoy two live terrestrial TV channels from two different frequencies at the same time, and it supports both reception of FTA (Free to Air) TV channels and encrypted TV channels.
TBS6290 can be used as a digital video recorder for recording digital terrestrial TV programs with full HDTV support. With the time-shifting function.

S.R.P £179.95

TBS 6905 HD DVB-S2 Quad Tuner PCIe Card

TBS6905 is a PCI Express interface digital satellite TV Tuner card with four tuners for watching and recording FTA (Free to Air) satellite TV on PC. The four tuners allow you to watch TV channels from one transponder or satellite while recording three other channels from different transponders/satellites at the same time.
It's compatible with Windows 7 Media Center and popular software like MediaPortal, DVBlink, DVBDream, DVBViewer, ProgDVB, Skynet, TSreader, XBMC and MythTV.

S.R.P £179.95

OUR SMALLEST CARD EVER !

TBS 7220 DVB-T2/T/C mini PCIe Card

TBS 7220 DVB-T2/T/C TV Tuner mini PCIe Card, is a digital terrestrial tuner card with DVB-C function added, which allows you to enjoy Free-to-Air (FTA) digital terrestrial/ cable TV and digital stereo radio on PC.
TBS 7220 can be used as a digital video recorder for recording digital terrestrial/cable TV programs with full HDTV support.
It also enables you to pause a live broadcast & continue from where you left with the time-shifting function.
It's ideal for watching UK Freeview SD and HD.

S.R.P £69.95

CHECK OUT OUR WEBSITE FOR MORE GREAT TBS PRODUCTS

WWW.TBSCARDS.CO.UK

EXCLUSIVE OFFER

Buy online at www.tbscards.co.uk and receive a 10% discount on any TBS product using code: LINUXUD

Trade Enquiries Welcome
E: mike@pciex.co.uk

Buy Online:
www.radtv.co.uk
www.valueav.co.uk
www.amazon.co.uk

Reviews

GROUP TEST

Backup solutions

Backup tools are not a luxury – everyone needs to have one in order to recover precious data when the main storage fails

luckyBackup

LuckyBackup is a popular archiving tool with a Qt4 interface. The application was given high scores on **kde-apps.org** in 2008–2010 and it is still in the top chart. Currently, the luckyBackup development is nearly stalled; however, this can hardly be considered a pitfall because back-up tools will remain useful for years.

Download: bit.ly/1PtjlhV

Back In Time

The Back In Time site offers many outdated packages, but there are also fresh commits. The tool is actively developed and receives frequent updates. Back In Time is also a Qt-based application, but has separate integration packages for KDE and GNOME, so it looks like a pro in any environment.

Download: bit.ly/1BcpJ8V

Déjà Dup

Déjà Dup is the only contender in this test that is directly based on sync. It is tightly integrated in GNOME, Unity and Cinnamon, and resides in the control centre there. The design idea behind Déjà Dup is to hide away all hassles and complexities and also be friendly with users who aren't the most tech-savvy.

Download: bit.ly/1k4WLpp

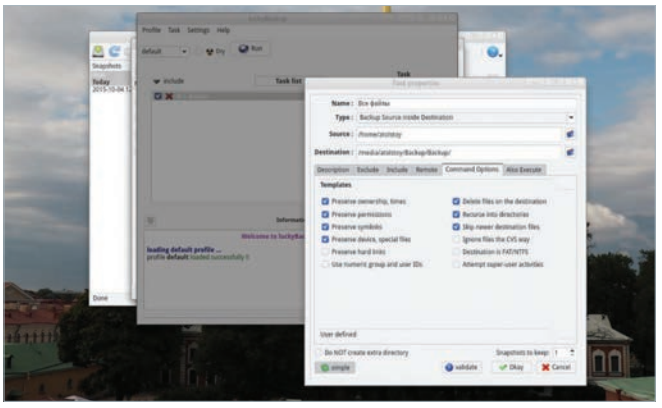
KBackup

There's no need to explain the letter 'K' here, but KBackup is a fully desktop-agnostic application. It shares most features with Back In Time and luckyBackup, but offers an even cleaner and easy-to-understand GUI. Despite being so simple-looking, KBackup is capable of producing good backup copies.

Download: bit.ly/1JAr7cf

luckyBackup

A sophisticated and complete wrapper around multiple sync options



The Advanced button expands the rich set of available options and tune-ables

Ease of use

The first time luckyBackup is started, it throws you off a bit with its variety of menus, toolbar buttons and window areas. To do the job you must create a new Task, fill its name, source and destination directories, and optionally change the default behaviour in advanced settings. Not too complicated, but for newcomers it may be worth googling the right step sequence.

Feature set

LuckyBackup has profiles, scheduling and support for various types of behaviour, like keeping snapshots, updating incremental backups, syncing or replicating. It can also store backup copies on remote locations, perform dry runs, log everything and even notify you by email.

Access backed-up data

Though luckyBackup can zip the backup copy into an archive, the default settings suggest the source files and directories are transferred onto the destination directory intact. It means your data is at your fingertips, ready for read and write operations. LuckyBackup doesn't prevent Linux users from manipulating backed-up files.

Availability and integration

LuckyBackup is a third-party open source app, but it is included in most Linux distributions. The application behaves and looks equally good under any desktop environment and supports native notifications of most desktops. However, there is no integration with Control Centre or System settings – luckyBackup is a completely standalone app.

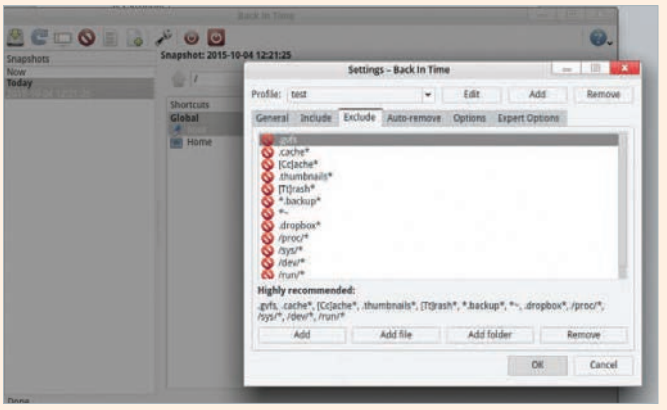
Overall

LuckyBackup is definitely a smart app and once you spend some time discovering its options, you'll be ready to manage any sort of backup. We liked the 'dry run' option, which can save you from doing harm to your data.

8

Back In Time

A powerful tool loaded with goodies and focused on incremental backups



Back In Time already knows what you could have skipped from the backup copy. Of course, you can customise the list

Ease of use

Back In Time claims to be simple yet it takes time and skill to get around. The first time you run the app, it opens the Settings window and asks you to fill in required fields, like the destination directory, directories included for backing up and so on. After that, you'll be dealing with the main Back In Time window, which looks like a file browser with backup snapshots as entry points.

Feature set

Back In Time takes care of many auxiliary things often ignored in other 'simple' tools, such as using the **nice** and **ionice** tools to reduce system load for scheduled backups. However, it doesn't support file compression and requires filesystems with hard-link support for backup destinations.

Access backed-up data

Once the first snapshot is created, you navigate to the destination directory and find the snapshot metadata files together with the 'backup' subdirectory, which contains the copy of your files. Under the massive subdirectories tree, your files will appear as they do in the source. Back In Time cannot encrypt or archive data; it just replicates it.

Availability and integration

The application has a command line interface that you can discover by issuing the **backintime -help** command. The graphical application exists in two flavours: **backintime-kde4** and **backintime-gnome**. Back In Time doesn't come as default in major Linux distributions, but it's widely available for many, including openSUSE OBS and Arch AUR.

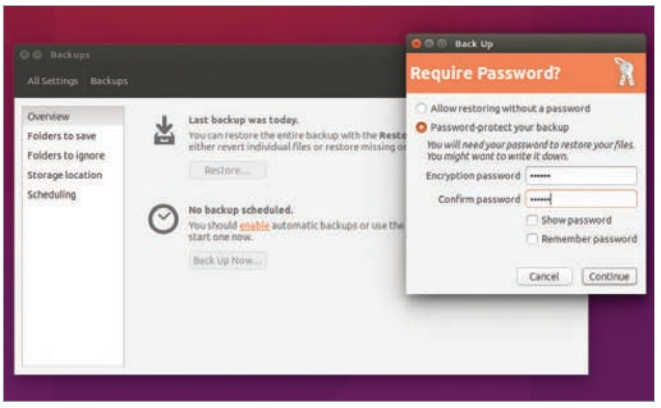
Overall

If you're ready to play with tons of rsync-related options and create a perfect backup copy then Back In Time is the right choice. The application is smart and efficient, and it also fits well into any desktop of your liking.

9

Déjà Dup

This GTK application uses Duplicity to produce encrypted archives



Déjà Dup is easy to get going with, which is an obvious advantage, but it does take control away from you

Ease of use

Déjà Dup shares the approach of the GNOME UI design: an application has to be simple and easy to get started. Déjà Dup has a clean and welcoming interface centred around 'what to back up' and 'where to back up' options. Beside that, it can ignore user-defined folders and perform backups according to a schedule.

Feature set

A result of such a clean and uncluttered interface is that you get only essential features. It's arguable whether an average user needs more or not, because GNOME and Unity users tend to think in the same manner that OS X users do, where simplicity is a sacred cow.

Access backed-up data

Again, novice users may not take to it, but the rest need to be confident that their data is accessible. The problem is that Déjà Dup doesn't want you to manipulate backed-up files outside the application. You should back up and restore your files only with Déjà Dup in case you somehow decompose the encrypted archive.

Availability and integration

This is where Déjà Dup shines: it is instantly available in any Linux distribution, which includes GNOME components and its default applications. In Unity it is also integrated with a side panel, where the Déjà Dup icon displays backup or restore progress. If you use KDE or a custom minimalistic desktop, Déjà Dup looks bad but it's not an issue.

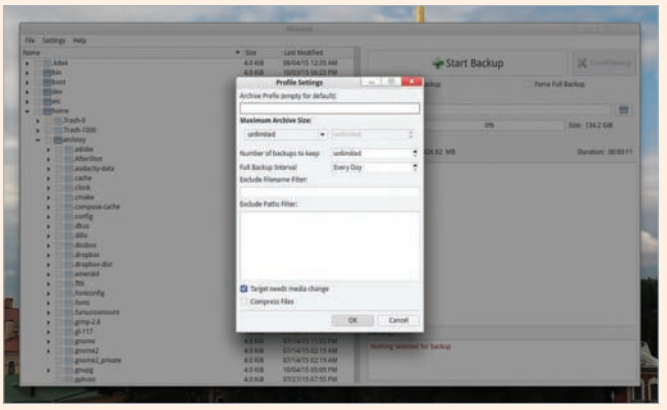
Overall

Déjà Dup is simple and clean, but it doesn't look as nice on other desktops as Qt-based apps look on a GTK3-based desktop. However, the biggest concern is that Déjà Dup takes you away from controlling your data.

7

KBackup

The backup application with a legacy floppy diskette on its logo



Tick the boxes for what you want to back up, choose a target directory and then press that big button!

Ease of use

KBackup clearly wins the test for user-friendliness – we even managed to fit the comprehensive user manual in the caption below the image! In fact, it's impossible to get lost with KBackup, with its straight logic and clean GUI layout. Selecting source files and directories then targeting it to the destination directory takes seconds.

Feature set

As always, there is a trade between usability and choice of features. KBackup looks poor this time, with the only extra thing being Profiles where you can limit the size of your target backup pool and set scheduling. KBackup cannot even restore files from the backup copy.

Access backed-up data

KBackup's approach to storing backup copies is a happy median between plain directory structure and encrypted archives. In the target directory you'll find a TAR file named after the date and time the backup was made. So KBackup packs everything into a single uncompressed file. Restore it with a command like **tar -xvf yourfile.tar**.

Availability and integration

The latest KBackup release hasn't been updated since 2012, but it's been packaged for major Linux distributions and is available either from standard repositories or from the KBackup page at **kde-apps.org**. The application supports system tray integration (not only KDE) and looks nice within any desktop environment.

Overall

Very nice, but perhaps too simple. KBackup doesn't even care about the way you'd want to restore your backed-up data. At least the essential replication feature works as expected and you won't get lost with the simple interface.

7

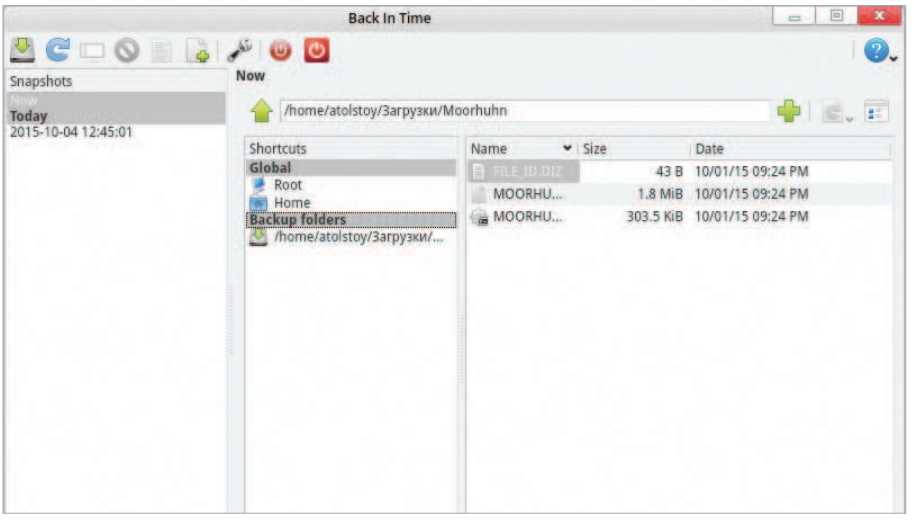
In brief: compare and contrast our verdicts

	luckyBackup		Back In Time		Déjà Dup		KBackup	
Ease of use	Takes a little time to figure out how to use it once you've loaded it for the first time	5	A bit tricky for first-time use, but it's okay later on once you're up to speed with it	5	Perhaps the most friendly entry-level backup tool in this group test	8	Even simpler than Déjà Dup and the interface looks very clean in addition to this	9
Feature set	A very capable feature set with cool commands like the 'dry run'	8	The most advanced feature set of all those tested, with only a small carve-out	9	Déjà Dup hides all complexities, which is at the cost of the modest feature set	6	KBackup is not equipped with a restoration tool. Very few features available	3
Access backed-up data	Your files are replicated without any changes being made – at least by default	9	Under the depth of subdirectories lie your files, ready for use if you can find them	8	You'll hardly be able to recover your files and data unless you have Déjà Dup to hand	3	All files are stuffed into an uncompressed TAR file, which takes extra action to 'untar' it	7
Availability and integration	A classic third-party standalone app without special integration to speak of	6	Good integration with KDE and GTK-based desktop environments, so good portability	8	Very smooth in GNOME, Unity and Cinnamon. It even comes as supplied as a default accessory	8	Integrates at your system tray and that's it: feels a little cheap and nasty	6
Overall	**This is a very balanced backup solution, but it does lack some polish and finish**	**8**	**The most balanced tool in this set with advanced features and a clean UI**	**9**	**A 'vendor lock-in' with an open source app? No thanks, we'll keep looking around**	**7**	**Redeeming with its simplicity, but not really good enough to beat the rest**	**7**

AND THE WINNER IS...

Back In Time

Back In Time clearly scored more points thanks to its impressive set of features that reveal the inner potential of sync and Linux shell utilities in general. It also boasts very polished integration with most desktop environments, clean and professional-looking design layout and very sane default presets – like the predefined list of commonly excluded folders. But there are a few more questions left to discuss that can affect your choice of a backup tool. The first one is simplicity. Back In Time is not the easiest tool to use, but this is true only when you launch it for the first time. Once you perform your first test backup, you'll come to love this powerful tool. In return, you get the precise control of what, where and how you back up – something that simpler tools like Déjà Dup take away from you.



The Back In Time window is an easy tool to manage backups and browse the filesystem

Another concern is the lack of an archiving option in Back In Time (and luckyBackup as well). We don't think it is a huge concern, though, because it brings little use. The most voluminous files you'd probably want to back up are videos, databases and other binary stuff that is already compressed or encoded. Encapsulating it into an extra compressed archive will require a lot of time and CPU horsepower, with little or no benefit in the end. After all, restoring files from an uncompressed state is a lot simpler as well.

Of course, there are even more backup software titles for Linux, such as Bacula for enterprise-level backups or dozens of cloud-based services (Dropbox/MEGA/SpiderOak) that are not open source. However, we wanted to pick up the most user-friendly and easy-to-set-up tools, which you can install and use on an average Linux machine. Of the many popular choices, Back In Time seems to be the most balanced and sane to use on a regular basis.

Alexander Tolstoy

MINI PC

Compulab Fitlet iA10

CPU
Quad-core 1.2GHz (2.2GHz Turbo) AMD A10-Micro6700T APU

GPU
Integrated 500MHz Radeon R6 with 128 shader cores

RAM / Storage
4GB DDR3 / 64GB mSATA SSD

Network
2x gigabit Ethernet, Intel AC-7260 Dual-Band Wi-Fi, Bluetooth 4.0

Connectivity
2x HDMI, 2x USB 3.0, 3x USB 2.0, 1x micro-SD, GPIO header, RS232 header, powered eSATA 6.0Gb/s

Dimensions
108mm x 83mm x 24mm

Price
£300

tinygreenpc.com (UK)
fit-pc.com (International)

With features rivalling desktops dozens of times larger, is Compulab's Fitlet product the ultimate example of a miniature Linux box?

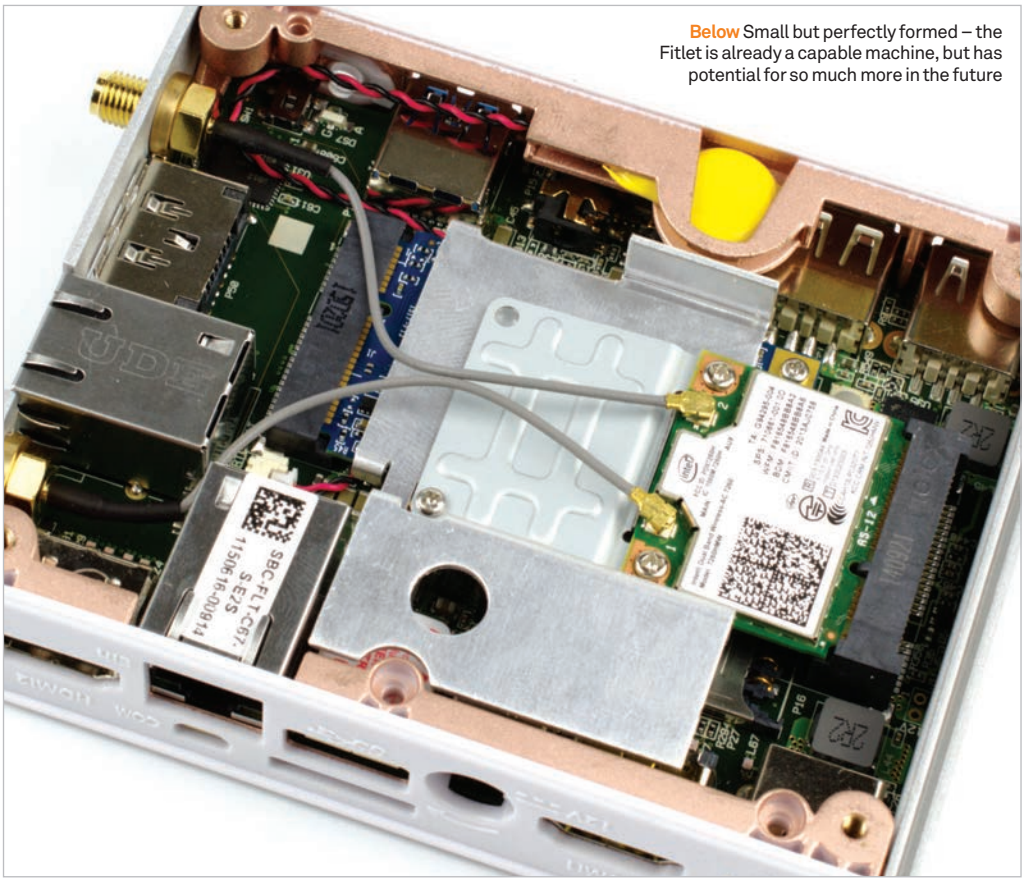
The Fitlet is a tiny PC, fully compatible with Linux, and has more features than you can shake a stick at. And while it isn't Compulab's first fanless mini PC, it may well be its best.

Unboxing the Fitlet is a great experience: the box seems too small to hold a computer of any power, but the machine itself – housed in a metal chassis, the top of which doubles as a heatsink – takes up a tiny fraction of this, with the remaining space given over to a wall-wart power supply, HDMI-to-DVI adapter cable, plus an antennae for the built-in Wi-Fi available on the top-end model, the Fitlet iA10.

The existence of a top-end model implies there's a bottom-end, of course. The Fitlet family, in fact, is made up of a range of externally identical devices: the Fitlet-b is the entry-level model, with the Fitlet-i offering more functionality and an improved processor, while the Fitlet iA10 improves the processor still more. There's even a Fitlet-X range, which replaces the dual gigabit Ethernet and eSATA ports of the Fitlet-i with three gigabit Ethernet ports – all of which run at full speed on the PCI Express bus.

This role flexibility is made possible by a system Compulab has dubbed Facet. Using Facet modules, which make up the side panel of the chassis, it's possible to alter the functionality of a Fitlet – a feature the company has promised to exploit more fully with additional Facet modules in future.

The model reviewed is the Fitlet iA10, supplied with a 64GB mSATA SSD and 4GB DDR3 RAM. This is unusual:



Below Small but perfectly formed – the Fitlet is already a capable machine, but has potential for so much more in the future

> "Using Facet modules, which make up the side panel of the chassis, it's possible to alter the functionality of a Fitlet"

Fitlets are usually sold bare-bones, although UK supplier TinyGreenPC allows the buyer to select from a variety of storage and memory options at the point of purchase.

Useful that may be, but TinyGreenPC does charge a premium over buying from Compulab direct – and a whopping £48 including VAT to copy a Linux Mint 17.2 system image to the optional SSD, a fee we'd suggest to readers should be eschewed in favour of booting from a Live USB and installing it yourself for free.

The ability to do this reveals a major advantage the, admittedly expensive, Fitlet has over rival ultra-compact computers: it uses an industry-standard UEFI BIOS and 64-bit x86 processor, the AMD A10-Micro6700T, meaning it's compatible with almost any modern desktop or server operating system. It's even possible to overclock the 1.2GHz quad-core chip, by selecting a new target thermal profile in the BIOS, to boost performance – and Compulab sells a replacement chassis lid with larger heatsink fins for $15 if you want to squeeze the most performance possible from the device.

Even overclocked, though, the Fitlet can't hold a candle to the most budget desktop system, nor would you expect it to: the CPU has a 4.5W thermal design profile, and the entire system is extremely energy efficient. That doesn't mean Compulab has skimped on features, though: there are two USB 3.0 and three USB 2.0 ports, a powered eSATA 6Gb/s port, two full-speed gigabit Ethernet ports, a micro-SD slot, analogue and S/PDIF digital audio inputs and outputs, and even a micro-RS232 header. The on-board graphics also drives two HDMI 1.4a ports for dual-display functionality. Compulab also sells a range of accessories: VESA and DIN-rail mounts, the aforementioned heatsink, a 3G modem and even a miniature 12V UPS.

The only real disappointment comes in the form of a general-purpose input-output (GPIO) header, for which Compulab sells an optional breakout cable: at the time of writing, GPIO drivers were not available, although work was claimed to be near-complete on support for the three 1.5V and six 3.3V user-addressable pins.

Gareth Halfacree

Pros

Amazingly flexible and available with Linux already installed, the Fitlet is an excellent choice for the energy- or space-conscious user

Cons

Its performance, naturally, lags behind that of a full-size PC, and the Fitlet's GPIO capabilities could not be tested

Summary

The Fitlet is impressive, and the Facet system promises flexibility for future redesigns. While its CPU performance is closer to a tablet than a full-size PC, so is its power draw – and its size is little larger than some laptop power supplies. Couple excellent Linux support with the potential to boost performance by overclocking and you've got a tempting machine.

TEXT EDITOR

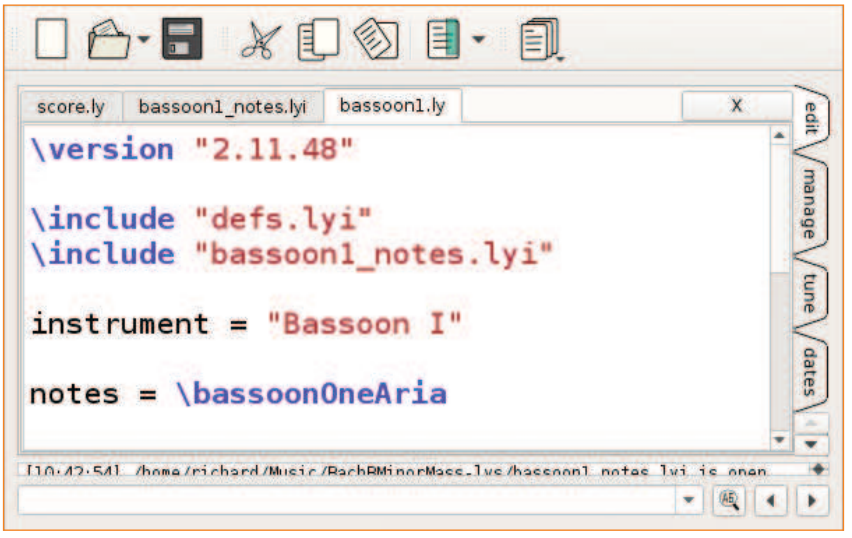
TEA 41.1.0

A text editor for when an office suite would be overkill, TEA is a refreshing alternative

Good text editors tend to be focussed on workflow for coding, or to be simplified letter and document writing tools. TEA manages to embrace both worlds, being a decent enough word processor – opening .odt, .rtf and .docx files, plus AbiWord, KWord and other formats – and a better alternative to default code editors like Gedit. Syntax highlighting includes everything from TeX and LilyPond (see screenshot), through Lua and XML, to NASM and Verilog.

Binaries are available for most platforms, as this is a long-established project, though surprisingly little known. TEA uses several Qt libraries, and can utilise aspell or hunspell for spellchecking duties – relatively minimal requirements for any editor or word processor. The first thing you'll notice is, in addition to conventional horizontal tabs for multiple files, vertical tabs give access to file management, configuration, calendar functions and the manual.

With its built-in, mc-like file manager; customisable hotkeys; image handling (within TEA, and easy drag and drop to other programs); JavaScript plug-in architecture; clipboard manager; WikiMedia and DocBook tools; string handling functions; bookmarks; built-in calendar; and support for templates and code snippets; this could be the most capable editor to run on minimal resource machines.



Above LilyPond, the file format for music engraving, benefits from syntax highlighting in TEA

Pros

The install size is small enough to fit on pretty much anything, which makes it very versatile, and it's fairly stable

Cons

There is no getting away from the idiosyncrasies, although they are minor, and there are missing features

Great for...

Low resource machines – like the Raspberry Pi

semiletov.org/tea

LOCK MANAGER

FLoM 1.0.0

FLoM can synchronise your shell commands to avoid conflicts

Sometimes you don't realise you need something until you discover it and start using it. FLoM, which has just made the small leap to version 1.0.0, is one such useful tool. It serialises your processes to enable scripts and shell commands to be synchronised – either locally, in a single user system, or over a network. If you've come to the limits of what can be done by command concatenation and creative use of **sleep** or **cron**, this could be what you need.

Installation is the usual **configure** and **make**, then **make install** with root permissions – there are no library dependencies to hunt down. Just as the **nice** program enables you to set priority, FLoM lets you run a command with a specific serialisation. Getting started is easy – just take a look at the wiki on FLoM's SourceForge website; several use cases are presented. A simple example is to enter **flom --sleep 10** at one terminal and **flom --ls** in a second, which will not execute the **ls** command until the **sleep 10** has finished running in the first terminal instance. This will create a lock on resources, in addition to it being released.

The ambitious roadmap covers plans for future features for use as an IoT gateway component, and in Docker, as well as a few interesting ideas which could potentially see FLoM widely adopted. Take a look for yourself and see if it turns out to be something that you could make use of.

Pros

This is an extremely easy API to get to grips with, and what makes it even better is that it works well

Cons

Maybe we are getting jaded, but it's yet another tool that has to be learned and incorporated

Great for...

Running scripts and tasks without resource conflict

sf.net/projects/flom

RPG CREATOR

RPGBoss 0.9.6

Open ended games are great to play, but RPGBoss lets you create one

Adventure games predate the home computer – both as command line software on the mainframe and paper-based role playing games (RPG) – but they really came into their own in the PC era, with some excellent Dungeons & Dragons-inspired titles. Online versions with thousands of players have stretched the genre, but a great way to go back to basics is to use RPGBoss to create your own. RPGBoss is a cross-platform RPG creator, written in Java and licensed under the AGPL3.

The real advantage with Java software is that you can just download the software and run it anywhere (it works with OpenJDK as well as proprietary Java). That said, we ran into the usual display weirdness we often encounter when a Java GUI tries to make sense of living on our favourite tiling window manager, Xmonad. Put it in a more conventional environment and you're ready to go: select a new project and start building a universe – well, a level at least – for your new role-playing game.

There are a few ready-built games on the RPGBoss wiki, to use as inspiration, or just to play if you haven't finished your own game. These include some horror and historical titles. Building your own is a simple case of following the wiki instructions down a well-worn path through events, encounters and maps; the available assets include music and battle scenes. It's not perfect, but it is good fun.

Pros

Actively developed, relatively easy to pick up, plus it lets you get nice and creative

Cons

Everything still feels a little clunky and it isn't yet fully documented, which isn't helpful

Great for...

An easy start to creating your own D&D adventure

rpgboss.com

MACHINE EMULATOR

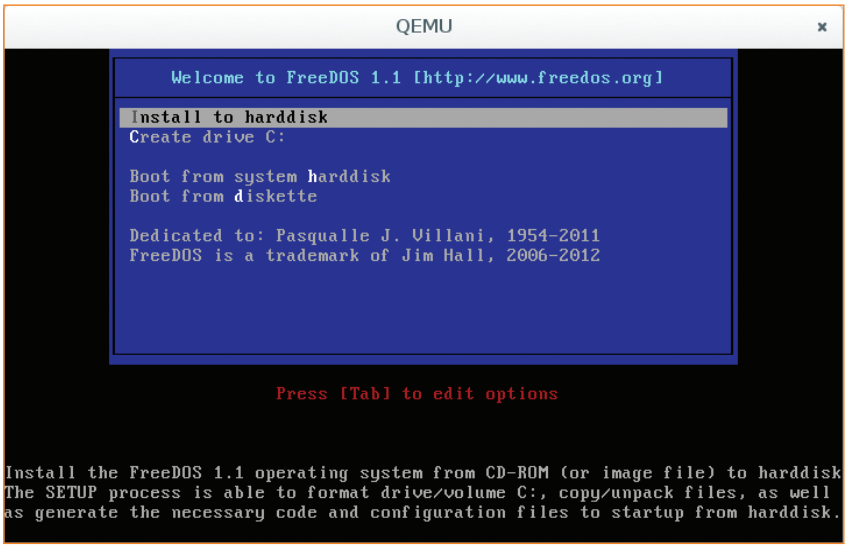
QEMU 2.4.0

QEMU gives the whole CPU, in many flavours

Amid all the glamour of recent developments in containers and hardware-assisted hypervisors, the humble QEMU – which can emulate many architectures entirely in software, through dynamic binary translation – is almost in danger of being overlooked. The recent release is a good excuse to remind ourselves what this wonderful piece of software will do.

Unlike container solutions, which enable you to share your (host) operating system's kernel with the guest OS, QEMU emulates an entire processor and system architecture. This is great for everything from developing for Raspbian to testing compiles across multiple architectures. It's also handy for playing old games on FreeDOS, or other old and neglected platforms.

As to speed issues, in most installations QEMU defaults to Linux's KVM, where available, and the benchmarks show it to be pretty speedy. KVM is not just available on x86 – you'll find it works on ARM Cortex A15 and PowerPC 440 and 970 chips, amongst others, so if you have a Samsung Chromebook Series 3 running Linux natively, or a PowerMac G5, you could (theoretically) get the same speed up as Intel users.



Above Emulate and virtualise thanks to QEMU

Pros

Offers users a really good level of versatility – and it's also the only choice for emulating certain architectures

Cons

The documentation leaves a lot to be desired – to get to grips with things you will need to read an online tutorial

Great for...

Old games, and new cross-platform software

qemu.org

ALL TITLES JUST £49.99

SPECIAL OVERSEAS OFFER

SAVE UP TO 17% ON A GIFT SUBSCRIPTION THIS CHRISTMAS

REAL CRIME
Uncover the most fascinating and notorious true crimes in history
13 issues, save 4%

HOW IT WORKS
The action-packed science and technology magazine
13 issues, save 10%

- **EXCLUSIVE CHRISTMAS OFFER**
- **SAVE UP TO 17%** ON THE SHOP PRICE
- NEVER MISS AN ISSUE OF **YOUR FAVOURITE MAGAZINE**
- **FREE E-CARD** TO SEND WHEN YOU BUY AS A GIFT
- **IDEAL CHRISTMAS GIFT** - THAT LASTS ALL YEAR

ALL ABOUT HISTORY
Bringing history to life for the whole family
13 issues, save 15%

GADGET
Packed with the latest, coolest and most exciting tech
12 issues, save 17%

WORLD OF ANIMALS
Everything you need to know about the world's most amazing wildlife
13 issues, save 4%

HISTORY OF WAR
The stories, strategies, heroes and machines of historic conflicts
12 issues, save 17%

DIGITAL PHOTOGRAPHER
Inspiration, tutorials and tools for enthusiasts and professionals
12 issues, save 17%

RETRO GAMER
The number one magazine for classic gaming
12 issues, save 17%

ALL ABOUT SPACE
Discover the wonders of the solar system and beyond
13 issues, save 15%

SCIFINOW
The number one magazine for sci-fi, fantasy and horror fans
12 issues, save 17%

FIND OUR FULL RANGE OF TITLES AT THIS **GREAT PRICE ONLINE!**

ONLINE AT

www.imaginesubs.co.uk/xmas151

ORDER HOTLINE

+44 (0)1795 592 869

BUY AS A GIFT OR TREAT YOURSELF!

Use code **XMAS151** for this extra-special price.

DISCOVER THE LATEST AND GREATEST GADGETS AND EXPLORE THE TECH INSIDE

TRY 3 ISSUES FOR JUST £1!

BRAND NEW FROM THE MAKERS OF HOW IT WORKS

Every issue packed with

The latest tech Everything from hoverboards, drones, wearables, sports tech, smart homes and more...

Buying advice New and cool kit and where to get it

Tech teardowns Under the skin of the latest gadgets

and more ...

CALL THE ORDER HOTLINE

0844 249 0270*

OR ORDER ONLINE

imaginesubs.co.uk/gadget

and enter code NEW15T

*Calls will cost 7p per minute plus your telephone company's access charge. Please use code NEW15T. This offer entitles new UK Direct Debit subscribers to receive their first 3 issues for £1. After these issues standard subscription charges will apply; currently £15 every 6 issues. New subscriptions will start with the next available issue. Details of the Direct Debit Guarantee available on request. This offer expires 31st December 2015.

Hosting listings

Got a deal you think we should list?

Whether you're a hosting firm or a happy customer who wants a favourite provider listed, drop us a line with the details!
linuxuser@imagine-publishing.co.uk

Dedicated server listings

NAME AND URL	PACKAGE	PHONE NUMBER	COST PER MONTH	MINIMUM CONTRACT TERM	CPU CORES / SPEED	DISK SPACE	1GBPS INTERNET CONNECTION	HARDWARE RAID	REMOTE POWER REBOOT	PERMANENT KVM	UPTIME GUARANTEE	NETWORK BACKUP STORAGE	PRIVATE SUBNET	24/7 PHONE SUPPORT
poundhost A NAMESCO COMPANY **PoundHost** www.poundhost.com	DS 2.2	0333 247 0222	£19.00	1 month	Intel Atom 2 cores / 1.8Ghz	1x 250GB	Optional	N/A	✓	1 hour	99.9%	10GB Free	TBA	✓
	BudgetBox 4	0333 247 0222	£29.00	1 month	Intel Xeon 2 cores / 2.3Ghz	From 2x 250GB	Optional	✗	✓	1 hour	99.9%	10GB Free	TBA	✓
	DS 3.1	0333 247 0222	£39.00	1 month	Intel Xeon 2 cores / 2.2Ghz	From 2x 500GB	Optional	✓	✓	1 hour	99.9%	10GB Free	TBA	✓
	DS 4.3	0333 247 0222	£59.00	1 month	Intel Xeon 4 cores / 3.2Ghz	From 2x 500GB	Optional	✗	✓	1 hour	99.9%	10GB Free	TBA	✓
	DS 4.5.1	0333 247 0222	£79.00	1 month	Intel Xeon 4 cores / 3.2Ghz	From 2x 500GB	Optional	✓	✓	1 hour	99.9%	10GB Free	TBA	✓
	DS 4.6.2	0333 247 0222	£115.00	1 month	Intel Xeon 4 cores / 3.4Ghz	From 2x 1TB	Optional	✓	✓	1 hour	99.9%	10GB Free	TBA	✓
	DS 6.5.1	0333 247 0222	£249.00	1 month	Dual Intel Xeon 12 cores / 2.2GHz	From 4x 1TB	Optional	✓	✓	1 hour	99.9%	10GB Free	TBA	✓
	DS 6.6	0333 247 0222	£349.00	1 month	Dual Intel Xeon 16 cores / 2.4GHz	From 8x 1TB	Optional	✓	✓	1 hour	99.9%	10GB Free	TBA	✓
Bravo14 (http://bravo14.co.uk)	Starter Linux	N/A	£20	N/A	N/A	2,000MB	N/A	✓	✓	✓	✓	✗	✓	✓
Bravo14 (http://bravo14.co.uk)	Starter Windows	N/A	£20	N/A	N/A	2,000MB	N/A	✓	✓	✓	✓	✗	✓	✓
Bravo14 (http://bravo14.co.uk)	Business Linux	N/A	£45	N/A	N/A	4,000MB	N/A	✓	✓	✓	✓	✗	✓	✓
Bravo14 (http://bravo14.co.uk)	Business Windows	N/A	£45	N/A	N/A	4,000MB	N/A	✓	✓	✓	✓	✗	✓	✓
Bravo14 (http://bravo14.co.uk)	Ultimate Linux	N/A	£60	N/A	N/A	Unlimited	N/A	✓	✓	✓	✓	✗	✓	✓
Bravo14 (http://bravo14.co.uk)	Ultimate Windows	N/A	£60	N/A	N/A	Unlimited	N/A	✓	✓	✓	✓	✗	✓	✓
catalyst2 (www.catalyst2.com)	Bronze Managed Dedicated Server	0800 107 79 79	£199	1 month	1x 2.4GHz vCPU	50GB	✓	✓	✓	✓	99.90%	✓	✓	✓
catalyst2 (www.catalyst2.com)	Silver Managed Dedicated Server	0800 107 79 79	£299	1 month	1x 2.4GHz vCPU	80GB	✓	✓	✓	✓	99.90%	✓	✓	✓
catalyst2 (www.catalyst2.com)	Gold Managed Dedicated Server	0800 107 79 79	£399	1 month	2x 2.4GHz vCPU	150GB	✓	✓	✓	✓	99.90%	✓	✓	✓
123-Reg (www.123-reg.co.uk)	Dell PowerEdge R200 (Ubuntu Linux)	0871 230 9525	£69.99	12 months	4x 2.13GHz	2x 160GB	10Mbit	✓	✓	✗	99.99%	0	✗	✓
123-Reg (www.123-reg.co.uk)	Dell PowerEdge R200 (Windows Web Edition)	0871 230 9525	£79.99	12 months	4x 2.13GHz	2x 160GB	10Mbit	✓	✓	✗	99.99%	0	✗	✓
Daily (www.daily.co.uk)	Linux VPS Pro	0845 466 2100	£29.99	1 month	2.27 Intel Quad Core	60GB	100Mbps	✓	✓	✗	✗*	✓ - full backup	✗	✗**
Daily (www.daily.co.uk)	Linux VPS Max	0845 466 2100	£59.99	1 month	2.27 Intel Quad Core	100GB	100Mbps	✓	✓	✗	✗*	✓ - full backup	✗	✗**
Daily (www.daily.co.uk)	Windows VPS Pro	0845 466 2100	£34.99	1 month	2.27 Intel Quad Core	60GB	100Mbps	✓	✓	✗	✗*	✓ - full backup	✗	✗**
Daily (www.daily.co.uk)	Windows VPS Max	0845 466 2100	£64.99	1 month	2.27 Intel Quad Core	100GB	100Mbps	✓	✓	✗	✗*	✓ - full backup	✗	✗**
Daily (www.daily.co.uk)	VPS Pro Hyper-V	0845 466 2100	£44.99	1 month	2.27 Intel Quad Core	60GB	100Mbps	✓	✓	✗	✗*	✓ - 1GB	✗	✗**
Daily (www.daily.co.uk)	VPS Max Hyper-V	0845 466 2100	£74.99	1 month	2.27 Intel Quad Core	100GB	100Mbps	✓	✓	✗	✗*	✓ - 1GB	✗	✗**
Daily (www.daily.co.uk)	VPS Ultra Hyper-V	0845 466 2100	£139.99	1 month	2.27 Intel Quad Core	200GB	100Mbps	✓	✓	✗	✗*	✓ - 1GB	✗	✗**
Heart Internet (www.heartinternet.co.uk/dedicated-servers)	Linux Dual Core	0845 644 7750	£79.99	12 months	Dual Core Xeon 2.33GHz	160GB	✓	✓	✓	✗	99.99%	✓	✗	24/7 Ticket support
Heart Internet (www.heartinternet.co.uk/dedicated-servers)	Windows Dual Core	0845 644 7750	£89.99	12 months	Dual Core Xeon 2.33GHz	160GB	✓	✓	✓	✗	99.99%	✓	✗	24/7 Ticket support
Heart Internet (www.heartinternet.co.uk/dedicated-servers)	Linux Quad Core	0845 644 7750	£129.99	12 months	Quad Core Xeon 2.5GHz	250GB	✓	✓	✓	✗	99.99%	✓	✗	24/7 Ticket support

O = Option

GET YOUR LISTING HIGHLIGHTED! CONTACT LUKE

luke.biddiscombe@imagine-publishing.co.uk
+44(0)1202 586431

Dedicated and Shared server listings

NAME AND URL	PACKAGE	PHONE NUMBER	COST	WEB SPACE	MONTHLY BANDWIDTH	POP3 ACCOUNTS	DATABASE SUPPORT	SHOPPING CART	VIRUS FILTER	FIREWALL	PHONE SUPPORT	EMAIL SUPPORT	WEB CONTROL PANEL	SERVICE LEVEL AGREEMENT
Netcetera www.netcetera.co.uk	Home	0800 808 5450	£34.88	1GB	Unlimited	500	✓	✓	✓	✓	✓	✓	✓	✓
	Designer	0800 808 5450	£71.88	5GB	Unlimited	1,000	✓	✓	✓	✓	✓	✓	✓	✓
	Business	0800 808 5450	£199.88	1000GB	Unlimited	1,000	✓	✓	✓	✓	✓	✓	✓	✓
	Reseller	0800 808 5450	£299.88	Unlimited	Unlimited	Unlimited	✓	✓	✓	✓	✓	✓	✓	✓
	Linux Cloud	0800 808 5450	£60	50GB	Unlimited	Unlimited	✓	✓	✓	✓	✓	✓	✓	✓
	Windows Cloud	0800 808 5450	£300	6GB	Unlimited	Unlimited	✓	✓	✓	✓	✓	✓	✓	✓
	3100QC Dedicated Server	0800 808 5450	£1,300	2TB	Unlimited	Unlimited	✓	✓	✓	✓	✓	✓	✓	✓
Blacknight (www.blacknight.com)	Minimus	+44 (0)845 5280242	€49.95	10GB	150GB	1,500	✓	✓	✓	✓	✓	✓	✓	✗
Blacknight (www.blacknight.com)	Medius	+44 (0)845 5280242	€89.95	20GB	300GB	5,000	✓	✓	✓	✓	✓	✓	✓	✗
Blacknight (www.blacknight.com)	Maximus	+44 (0)845 5280242	€149.95	30GB	600GB	Unlimited	✓	✓	✓	✓	✓	✓	✓	✗
Digital Gibbon Ltd (www.digitalgibbon.com)	Cheeky Chimp	N/A	Free	500MB	Unlimited	5	✓	✗	✓	✓	✗	✓	✓	✗
Digital Gibbon Ltd (www.digitalgibbon.com)	Digital Gibbon	N/A	£12	5GB	Unlimited	10	✓	✗	✓	✓	✗	✓	✓	✗
Digital Gibbon Ltd (www.digitalgibbon.com)	Silverback	N/A	£24	Unlimited	Unlimited	Unlimited	✓	✓	✓	✓	✗	✓	✓	✗
Digital Gibbon Ltd (www.digitalgibbon.com)	WordPress hosting	N/A	£12	5GB	Unlimited	10	✓	✗	✓	✓	✗	✓	✓	✗
Equiphase (www.equiphase.net)	Bronze	0121 314 4865	£30	200MB	2GB	10	✓	✓	✗	✓	✗	✓	✓	✓
Equiphase (www.equiphase.net)	Silver	0121 314 4865	£42	400MB	5GB	20	✓	✓	✗	✓	✗	✓	✓	✓
Equiphase (www.equiphase.net)	Gold	0121 314 4865	£72	800MB	10GB	100	✓	✓	✗	✓	✗	✓	✓	✓
Equiphase (www.equiphase.net)	Platinum	0121 314 4865	£114	1,200MB	40GB	200	✓	✓	✓	✓	✓	✓	✓	✓
Eurofasthost.com (www.eurofasthost.com)	Email only	02380 249 823	£40	1GB	2GB	10	✗	✗	✓	✓	✓	✓	✓	✓
Eurofasthost.com (www.eurofasthost.com)	Essential	02380 249 823	£75	2GB	5GB	10	✗	✗	✓	✓	✓	✓	✓	✓
Eurofasthost.com (www.eurofasthost.com)	Superior	02380 249 823	£140	5GB	10GB	25	✓	✓	✓	✓	✓	✓	✓	✓
Eurofasthost.com (www.eurofasthost.com)	Premium	02380 249 823	£250	10GB	25GB	100	✓	✓	✓	✓	✓	✓	✓	✓
Evohosting (www.evohosting.co.uk)	Starter	N/A	£29.99	500MB	1GB	3	✓	✓	✓	✓	✗	✓	✓	✓
Evohosting (www.evohosting.co.uk)	Home	N/A	£54.99	2.5GB	30GB	50	✓	✓	✓	✓	✗	✓	✓	✓
Evohosting (www.evohosting.co.uk)	Business	N/A	£79.99	6.5GB	Unlimited	Unlimited	✓	✓	✓	✓	✗	✓	✓	✓
Evohosting (www.evohosting.co.uk)	eCommerce	N/A	£159.99	30GB	Unlimited	Unlimited	✓	✓	✓	✓	✗	✓	✓	✓
Giacom (www.giacom.com)	Business Pro	0800 542 7500	£199	100MB	2GB	100	✓	✓	✓	✓	✓	✓	✓	✓
Hostway (www.hostway.co.uk)	Silver	0808 180 1880	£79.50	150MB	3GB	5	✗	O	✓	✓	✗	✓	✓	✗
NameHog (www.namehog.net)	Email Only	0845 612 0330	£11.75	25MB	1GB	5	✗	✗	✓	✓	✓	✓	✓	✓
NameHog (www.namehog.net)	Standard package	0845 612 0330	£35.25	100MB	4.5GB	10	✗	✗	✓	✓	✓	✓	✓	✓
NameHog (www.namehog.net)	Professional package	0845 612 0330	£58.75	250MB	8GB	25	✓	✓	✓	✓	✓	✓	✓	✓
NameHog (www.namehog.net)	Expert package	0845 612 0330	£105.75	500MB	15GB	75	✓	✓	✓	✓	✓	✓	✓	✓
Skymarket (www.skymarket.co.uk)	Standard 1	0800 321 7788	£49	10MB	2GB	1	✓	✗	✓	✓	✓	✓	✓	✓
Skymarket (www.skymarket.co.uk)	Standard 2	0800 321 7788	£69	20MB	2GB	1	✓	✗	✓	✓	✓	✓	✓	✓
Skymarket (www.skymarket.co.uk)	Premium 1	0800 321 7788	£99	25MB	2GB	1	✓	✗	✓	✓	✓	✓	✓	✓
ServWise (https://www.servwise.com)	Personal	0800 520 0716	£25.20	1GB	10GB	Unlimited	✓	✓	✓	✓	✓	✓	✓	✓
ServWise (https://www.servwise.com)	Business	0800 520 0716	£50.40	2GB	20GB	Unlimited	✓	✓	✓	✓	✓	✓	✓	✓
ServWise (https://www.servwise.com)	Reseller	0800 520 0716	£126	4GB	40GB	Unlimited	✓	✓	✓	✓	✓	✓	✓	✓

O = Option

Contact us...
Web: www.linuxuser.co.uk
Email: linuxuser@imagine-publishing.co.uk

Q & A

Your questions answered

Send us your questions and we'll do our best to answer them!

Access content

I have a couple of Linux PCs in my office: a full PC and a laptop. The laptop keyboard is a bit rubbish. However, there's work on there that can only be done on that laptop and cannot be moved to the main PC. I'd really prefer to use the human interface stuff from the main computer without having to unplug them every time I want to use the laptop. And before I start buying a new mouse and keyboard, I wondered if there was maybe a cunning software solution to my particular issue?

Mike

Indeed there is, so don't buy a new mouse and keyboard yet as there are two main solutions that instantly come to mind. They do depend on how close the PC and laptop are, and how you want to handle it.

The more common method would be to use a remote desktop app to dial into the laptop. You can set up a server on it using a number of services (such as Remmina) that you can then create a connection to on your main PC. Save the settings and you can quickly connect whenever you want. You'll have a view of the desktop on your main PC and be able to use your mouse and keyboard on it.

Otherwise, there's also Synergy. Synergy enables you to treat your PC as a server for the mouse and keyboard, which your laptop can then connect to. If your laptop is right next to your PC, you can basically start using it as a sort of second screen and access the files only on the laptop via the laptop screen. All you do is move the mouse to the other screen like in any dual-monitor setup and work from there. It also works cross-platform as well!

RAR files

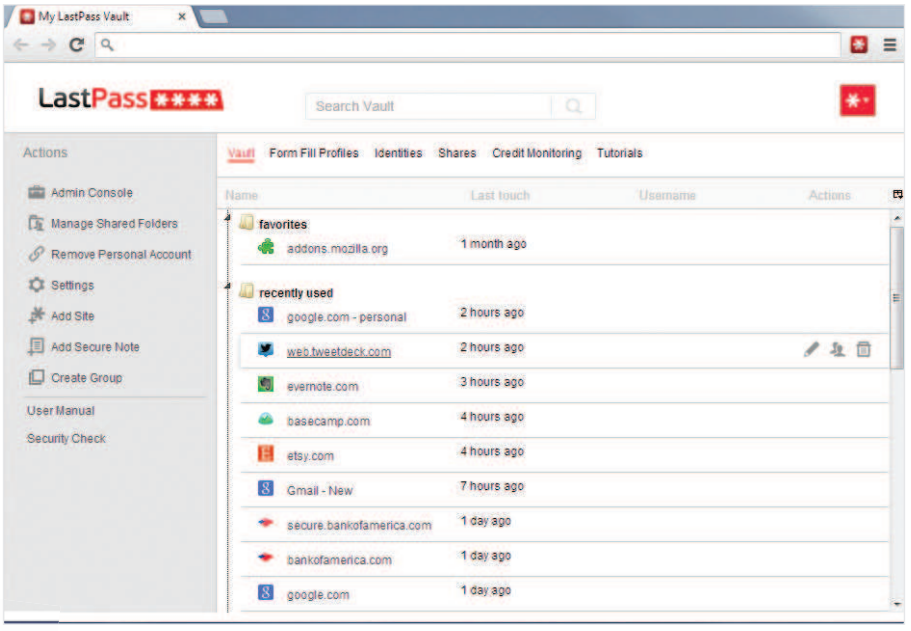
I have been sent some files by a friend (so I know they're legitimate!) and they decided to compress it into a RAR file. They're using Windows, so I guess in some circumstances that is the default compression option. Either way, I am having massive trouble trying to unpack this file in Ubuntu. It doesn't seem like the normal archive managers, or whatever, know how to open it, let alone get the files out!

Is there any way I can access the files on the RAR in Linux?

Nick

Left Remmina is useful for sorting out other people's PCs if they're having problems

LinuxUser & Developer
THE MAGAZINE FOR THE GNU GENERATION



Above LastPass saves passwords, bank details and other personal info very securely

Because RARs are quite proprietary, and not so often used in the Linux community, there aren't many apps that enable you to unpack RARs along with other types of compression. There is the package UnRAR, though, that lets you do it, so all is not completely lost!

*You'll need to install it on Ubuntu and most other distros (***sudo apt-get install unrar***). It's then used in the command line with something like the following:*

```
$ unrar /path/to/file.rar
```

It will unpack it in the directory you saved it to. Nice and quick!

Stay secure

I am always very conscious about my passwords and security on my system, and how I connect to the web. I have different passwords for all of them and like to think I use good password etiquette, but I'd like to really improve the way my details are kept private and secure. Short of using Tails, I'm not quite sure what methods I can use to go about this.

Laura

You're right, Tails is a bit of an extreme measure; although you'll definitely be very secure and private. However, it goes a step further than what you seem to want.

> "LastPass enables you to use one master password to access encrypted passwords within its database"

Our first step would be to get a good password manager, such as LastPass. Instead of having to remember several passwords, which can create problems and end up being a little less secure and very inconvenient, LastPass enables you to use one master password to access encrypted passwords within its database. As long as you have an excellent master password for LastPass, you can use complicated and highly secure passwords everywhere else.

We'd still suggest keeping email accounts off the LastPass list of protected online services though, as emails tie it all together.

Otherwise, you can try and use some hardware encryption tools so that it's difficult to snoop your data. Or try using an alias online.

Imagine Publishing Ltd
Richmond House, 33 Richmond Hill
Bournemouth, Dorset, BH2 6EZ
☎ +44 (0) 1202 586200
Web: www.imagine-publishing.co.uk
www.linuxuser.co.uk
www.greatdigitalmags.com

Magazine team

Editor Gavin Thomas
gavin.thomas@imagine-publishing.co.uk
☎ 01202 586257
Designer Sam Ribbits
Photographer James Sheppard
Senior Art Editor Will Shum
Editor in Chief Dan Hutchinson
Publishing Director Aaron Asadi
Head of Design Ross Andrews

Contributors

Dan Aldred, Joey Bernard, Gareth Halfacree, Tam Hanna, Richard Hillesley, Jessica Lamb, Jon Masters, Luke Mulcahy, Swayam Prakasha, Jon Silvera, Richard Smedley, Nitish Tiwari, Alexander Tolstoy, Mihalis Tsoukalos, Rob Zwetsloot.

Advertising

Digital or printed media packs are available on request.
Head of Sales Hang Deretz
☎ 01202 586442
hang.deretz@imagine-publishing.co.uk
Advertising Manager Alex Carnegie
☎ 01202 586430
alex.carnegie@imagine-publishing.co.uk
Sales Executive Luke Biddiscombe
☎ 01202 586431
luke.biddiscombe@imagine-publishing.co.uk

FileSilo.co.uk

Assets and resource files for this magazine can now be found on this website.
Support filesilohelp@imagine-publishing.co.uk

International

Linux User & Developer is available for licensing.
Head of International Licensing Cathy Blackman
☎ +44 (0) 1202 586401
licensing@imagine-publishing.co.uk

Subscriptions

For all subscriptions enquiries
☎ 0844 249 0282 (UK)
☎ +44 (0) 1795 418661 (Overseas)
Email: LUD@servicehelpline.co.uk
6 issue subscription (UK) – £25.15
13 issue subscription (Europe) – £70 (ROW) - £80

Circulation

Head of Circulation Darren Pearce
☎ 01202 586200

Production

Production Director Jane Hawkins
☎ 01202 586200

Finance

Finance Director Marco Peroni

Founder

Group Managing Director Damian Butt

Printing & Distribution

Printed by William Gibbons, 26 Planetary Road, Willenhall, West Midlands, WV13 3XT

Distributed in the UK, Eire & the Rest of the World by:
Marketforce, 5 Churchill Place, Canary Wharf
London, E14 5HU
☎ 0203 148 3300
www.marketforce.co.uk

Distributed in Australia by:
Network Services (a division of Bauer Media Group)
Level 21 Civic Tower, 66-68 Goulburn Street
Sydney, New South Wales 2000, Australia
☎ +61 2 8667 5288

ISSN 2041-3270

IMAGINE PUBLISHING

recycle
When you have finished with this magazine please recycle it.

FileSilo

YOUR FREE RESOURCES

LOG IN TO **WWW.FILESILO.CO.UK/LINUXUSER** AND DOWNLOAD THE LATEST DISTROS AND FREE SOFTWARE TODAY

LATEST DISTROS

YOUR BONUS RESOURCES

ON FILESILO THIS ISSUE, FREE AND EXCLUSIVE FOR **LINUX USER & DEVELOPER** READERS, YOU'LL FIND THESE GREAT RESOURCES...

- **Four latest distros**, including the newest Ubuntu and its MATE respin for the Raspberry Pi 2.
- **20 excellent FOSS packages**, including everything in our reviews section plus the tutorial tools.
- **Watch** The Linux Foundation, Red Hat and Raspberry Pi video guides, tutorials and webinars.
- **Code and assets** for this issue's tutorials, including everything you need for the Pygame Zero game.

FileSilo

LENGTH OF VIDEO TUTORIALS: **20** HOURS

TOP LINUX FOSS

TUTORIAL CODE & ASSETS

www.filesilo.co.uk/linuxuser

FILESILO – THE HOME OF PRO RESOURCES

DISCOVER YOUR FREE ONLINE ASSETS

- A rapidly growing library
- Updated continually with cool resources
- Lets you keep your downloads organised
- Browse and access your content from anywhere
- No more torn disc pages to ruin your magazines
- No more broken discs
- Print subscribers get all the content
- Digital magazine owners get all the content too!
- Each issue's content is free with your magazine
- Secure online access to your free resources

This is the new FileSilo site that replaces your disc. You'll find it by visiting the link on the following page.

The first time you use FileSilo you'll need to register. After that, you can use the email address and password you provided to log in.

The most popular downloads are shown in this carousel, so see what your fellow readers are enjoying!

If you're looking for a particular type of content like distros or Python files, use these filters to refine your search.

Green open padlocks show the issues you have accessed. Red closed padlocks show the ones you need to buy or unlock.

Top Downloads are listed here, so you can get an instant look at the most popular downloaded content.

Check out the Highest Rated list to see the resources that other readers have voted for as the best!

Find out more about our online stores, and useful FAQs like our cookie and privacy policies and contact details.

Discover our amazing sister magazines and the wealth of content and information that they provide.

HOW TO USE

EVERYTHING YOU NEED TO KNOW ABOUT ACCESSING YOUR NEW DIGITAL REPOSITORY

FileSilo

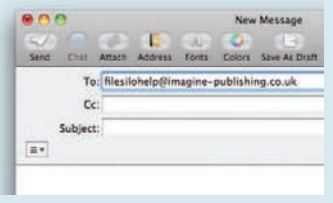
To access FileSilo, please visit www.filesilo.co.uk/linuxuser

01 Follow the instructions on-screen to create an account with our secure FileSilo system, then log in and unlock the issue by answering a simple question about the magazine. You can access the content for free with your issue.

02 If you're a print subscriber, you can easily unlock all the content by entering your unique Web ID. Your Web ID is the eight-digit alphanumeric code printed above your address details on the mailing label of your subscription copies. It can also be found on your renewal letters.

03 You can access FileSilo on any desktop, tablet or smartphone device using any popular browser (such as Firefox, Chrome or Safari). However, we recommend that you use a desktop to download content, as you may not be able to download files to your phone or tablet.

04 If you have any problems with accessing content on FileSilo, or with the registration process, take a look at the FAQs online or email **filesilohelp@imagine-publishing.co.uk**



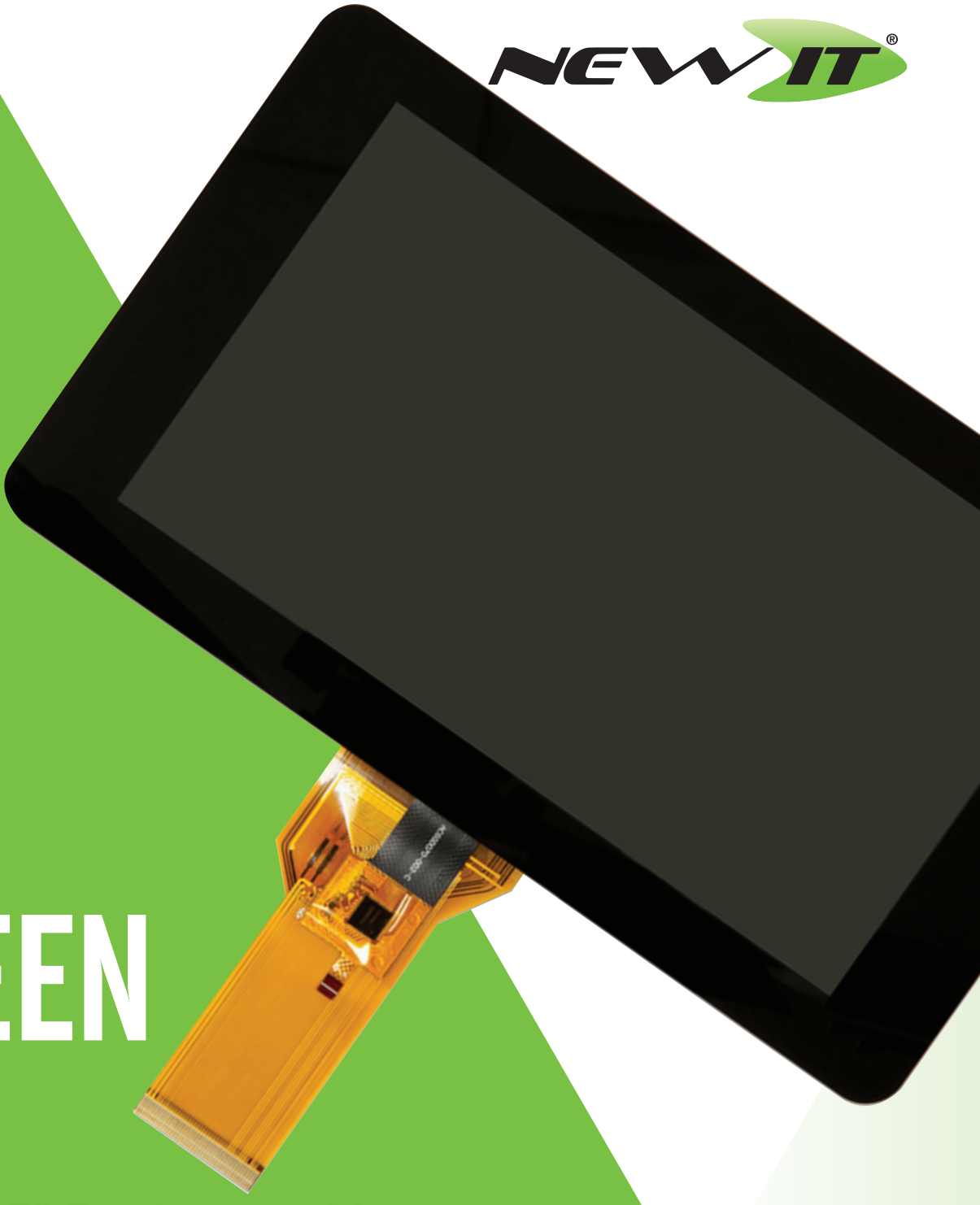
MORE TUTORIALS AND INSPIRATION

Finished reading this issue? There's plenty more free and open source goodness waiting for you on the **Linux User & Developer** website. Features, tutorials, reviews, opinion pieces and the best open source news are uploaded on a daily basis, covering Linux kernel development, the hottest new distros and FOSS, Raspberry Pi projects and interviews, programming guides and more. Join our burgeoning community of Linux users and developers and discover new Linux tools today.

www.linuxuser.co.uk

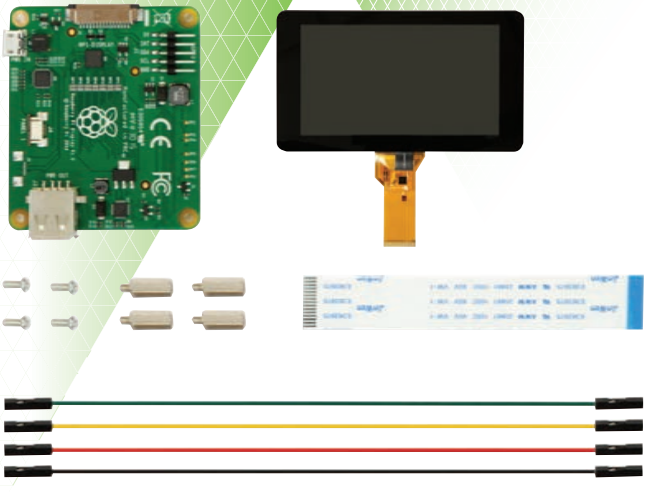
Issue 160 of Linux User & Developer is on sale 17 December 2015 from GreatDigitalMags.com

NEW IT®



SCREEN TIME

Raspberry Pi 7” Touchscreen Display



WWW.NEWIT.CO.UK