DESKTOP VIRTUALISATION

The best virtual tools reviewed and rated, including:

» Gnome Boxes 40 » Virtual Box 6.1 » VM Workstation 16.1

FREE DOWNLOAD BODHI 6.0

LINUX FORMAT

The #1 open source mag

PLUS! HOW TO...

- MODEL WITH BLENDER
- RECOVER DELETED FILES
- RUN CHROME OS ON A Pi

FASTER, BETTER SERVERS!

The race is on to replace CentOS with the next-gen of efficient, bulletproof server distros

67 pages of tutorials & features

Build a tiny custom Linux distro with Yocto

Use phones and tablets as handy second screens

Experience the Altair 8800, the first home computer

MAKER BASICS
Getting started with breadboards

MULTI-TASKING CODE
Asynchronous Python tools that wait for no server!

LXF August 2021

FUTURE

COVID

WHO IS MOST AT RISK? WHAT CAN PREVENT IT?

thomaswimberly.com @thomaswimberly @thomaswimberlystudios

PLEASE TAKE THE SURVEY

qmul.ac.uk/covidence

Queen Mary University of London

THE UNIVERSITY of EDINBURGH

Swansea University Prifysgol Abertawe

QUEEN'S UNIVERSITY BELFAST

SIGN UP

LINUX FORMAT

Corporate Patron Free Software Foundation

MEET THE TEAM

We're all about building a better server this issue, what tip would you give to our readers if they were setting up a home server?

Jonni Bidwell
Speaking from experience, it's a good idea to change your **bash** prompt or use **tmux** decorations – anything to embellish the fact that the machine you're working on is remote. It's no fun when you think you're **dd**-ing a USB stick, but it's actually your server's hard drive.

Les Pounder
A few years ago, the 4TB drive in my server died and it took the best part of a weekend to rescue all the data from a rapidly failing drive. My advice is to have backups of your important data – a second drive and a cloud service is the ideal solution.

Michael Reed
If you have the room, consider separating your home directory, your root (installation) directory and your data directory. You can do this by making use of manual partitioning when carrying out the distro installation. That way, you're left with a setup that can be upgraded at a later date.

Mayank Sharma
Make room for adding storage. The open source home server platforms are so dexterous that you'll soon be using them for more tasks than you initially planned. Powered USB drives will give you the best mileage, especially when using puny SBCs and plug computers.

Alexander Tolstoy
Putting together a home server is just the first part of the story. Another part is setting up proper reporting and monitoring tools to better manage the server in the long run. Personally, I'm addicted to those nice-looking *Zabbix* dashboard widgets.

You've been served

I suspect that many readers, whenever Linux is mentioned, imagine their personal computer sat in their homes. That's great: it shows how far the GNU/Linux world has come. It's clearly functional enough, easy enough and widespread enough to be used as a desktop system by your normal home user.

This does mean that much of our focus these days is on desktop systems – even if some of this is development or home-server applications. Perhaps we forget that most systems running Linux are servers, be that real or virtual.

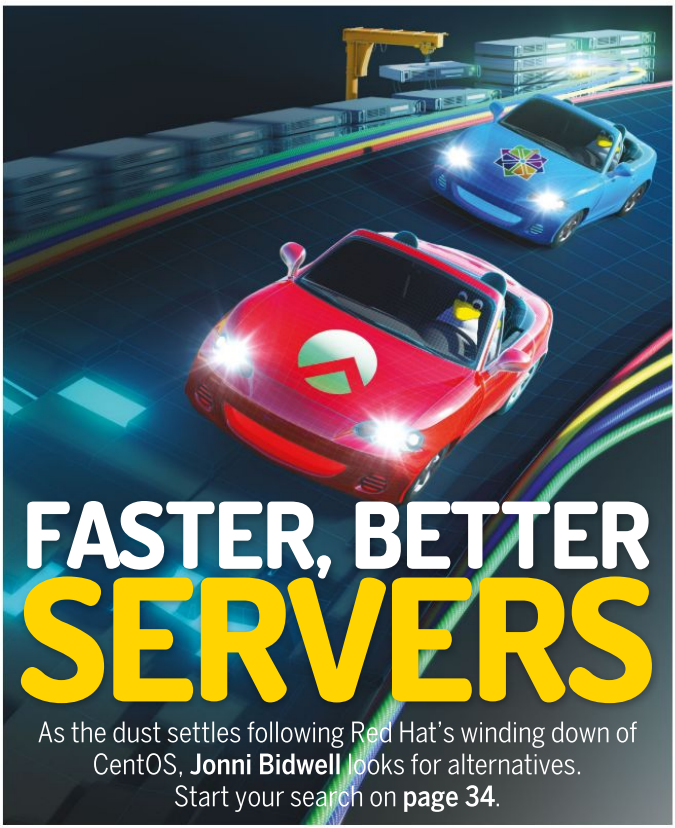
So with Red Hat pulling the server rug from under the CentOS community, we're taking the chance to look at how you can put together "better" servers using one of the most stable, reputable and well-regarded code-bases, whether that's with CentOS Stream, the new Rocky Linux or alternative AlmaLinux.

It's a protection that open source provides: even if a fundamental key player takes a project in a direction the majority doesn't agree with, there's every opportunity to create an alternative and head off in a direction you prefer. I've no doubt CentOS Stream will be successful, but likewise AlmaLinux and Rocky Linux too will find communities where they fit right in. So enjoy our feature!

Neil Mohr Editor
neil.mohr@futurenet.com

Contents

REVIEWS



FASTER, BETTER SERVERS

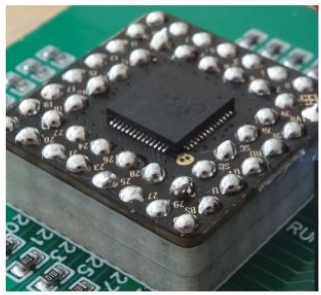
As the dust settles following Red Hat's winding down of CentOS, **Jonni Bidwell** looks for alternatives. Start your search on **page 34**.

ROUNDUP

IN-DEPTH

CONTENTS

Pi USER



CODING ACADEMY

REGULARS AT A GLANCE

ON YOUR FREE DVD
• Bodhi 6 • AntiX 19.4
• Lubuntu 21.04
See page 96

TUTORIALS

IN-DEPTH

Newsdesk

THIS ISSUE: Wi-Fi security in doubt » Users abandon Freenode » Audacity rethinks telemetry » AI gaming tech » LibreOffice success

SECURITY

Wi-Fi networks vulnerable to new FragAttacks

A new group of recently discovered vulnerabilities could affect billions of Wi-Fi networks and devices.

A series of vulnerabilities, dubbed FragAttacks, has been uncovered in the Wi-Fi specification that could potentially affect billions of devices around the world.

Wi-Fi networks transmit data using a process of frame fragmentation or frame aggregation. FragAttacks, short for fragmentation and aggregation attacks, abuse this process and enable malicious users to insert data into Wi-Fi traffic. It was security researcher Mathy Vanhoe who first brought these vulnerabilities to light.

Thankfully, the vulnerabilities, some of which appear to have been present since the creation of the Wi-Fi specification in 1997, can't be used to extract data, so there's no risk of passwords or other data being stolen. However, the idea that people could access your Wi-Fi network and the devices connected to it isn't a pleasant one. Furthermore, these vulnerabilities could be used with other hacks to cause damage.

Vanhoef, who has created a FragAttacks website detailing the vulnerabilities (you can read his findings at **www.fragattacks.com**), first realised the possibility of these vulnerabilities when he was working on the KRACK attacks in 2017. As he explains, "The discovered vulnerabilities affect all modern security protocols of Wi-Fi, including the latest WPA3 specification. Even the original security protocol of Wi-Fi, called WEP, is affected."

Although the vulnerabilities have only recently been publicly revealed, the past nine months have seen an industry-wide effort to analyse and identify them, and create patches. As Ars Technica reports (**http://bit.ly/LXF278Ars**), one of the most concerning vulnerabilities is CVE-2020-24588, which can be exploited to force devices to use a rogue DNS sever. This could mean that users of the compromised device are sent to dangerous websites rather than the one they intended to visit, putting their private information at risk.

While some of the vulnerabilities were found in the Wi-Fi specification itself, four were implementation flaws. This means they were found in bugs accidentally created by developers when creating code based on the Wi-Fi specification. These mainly affect firewalls and routers. The vulnerabilities, according to Vanhoef, enable attackers to "punch a hole through a router's firewall," and send malicious code to users. As we mentioned, Vanhoef has worked with companies to create patches for their hardware and software to protect against these vulnerabilities. Head to **http://bit.ly/LXF278FragAttacksPatches** for more information on the progress of these patches.

As with any good vulnerability, FragAttack has its own logo.

FRAGATTACKS IN A NUTSHELL

"The discovered vulnerabilities affect all modern security protocols of Wi-Fi, including the latest WPA3 specification."

SOFTWARE

Freenode's ownership drama sees users switch to Libera Chat

Numerous open source projects ditch the Freenode chat network after claims of a hostile takeover.

There's been some drama in the open source world recently following what some people claim is a hostile takeover of the Freenode IRC chat network (**https://freenode.net**), which was used by many communities and open source projects.

Freenode was sold to US developer and entrepreneur Andrew Lee in 2017, but according to a blog post by a former staff member (see **www.kline.sh**), Freenode staff had not been informed about the terms of the sale: "It turns out that this contract did indeed intend to sell the entire network and it's holdings." The post says that Lee had at the time promised that he would "never exercise any operational control over Freenode", but that this has now changed, with Lee asserting "total legal control over the network, including user data."

Things came to a head when certain Freenode channels (such as **##hntop**, see **http://bit.ly/LXF278FreenodeAbuse**) had been taken over by Andrew Lee, or other staff members, in a violation of Freenode policy. This led to many Freenode staff resigning in protest, and creating a new network: **libera.chat**.

As José Antonio Rey wrote in a blog announcing his resignation (**http://bit.ly/LXF278FreenodeResign**), "Mr Lee now has operational control over the Freenode IRC network. I cannot stand by such a (hostile?) corporate takeover of the Freenode network, and I am resigning as a staff volunteer along with most other Freenode staff."

Since then, many projects and distros such as Ubuntu, Gentoo and CentOS have switched from Freenode to Libera.Chat. Lee, however, claims that former staff are trying to "destroy [Freenode] from the inside." His plea to remaining Freenode users (**http://bit.ly/LXF278FreenodeBlog**) may not be enough, however, as a week after Libera Chat's launch, over 250 projects and 20,000 users have registered. This led to the Libera Chat team claiming it was the "fastest-growing IRC network ever."

Libera.Chat has seen an influx of users who've been disillusioned by Freenode.

SOFTWARE

Audacity shelves telemetry plans

Outcry from users prompts rethink from Muse Group.

The audio-editing program *Audacity* is one of the most popular open-source software in the world. However, recent developments have threatened to tarnish *Audacity's* reputation.

In April, *Audacity* was acquired by Muse Group, which is made up of a number of music brands, including Ultimate Guitar and MuseScore (read about the acquisition at **http://bit.ly/LXF278MuseGroup**). This worried long-term users, even though Muse Group promised that "the software will remain forever free and open source." While this may have allayed some people's fears, things took another concerning turn when a pull request appeared on GitHub (**http://bit.ly/LXF278AudacityGit**) for "Basic telemetry."

While the pull request was updated to ensure that "Telemetry is strictly optional and disabled by default. No data is shared unless you choose to opt-in and enable telemetry," the idea that collecting user data was even being considered left people believing their worst fears regarding Muse Group's acquisition were coming true.

The outrage was enough to cause the team to drop its plans, with Muse Group's head of creative software Martin Keary stating, "We assumed that making it opt-in would allay privacy concerns but since this isn't the case, we are dropping it." For error reporting, *Audacity* will not use Yandex and Google, because it "is at odds with the public perception of trustworthiness, so we will be self-hosting instead." You can read his full statement at **http://bit.ly/LXF278Keary**.

OPINION

UNDER A CLOUD

Matt Yonkovit
is Percona's head of open source strategy and a member of SHA (Silly Hats Anonymous).

"For us techies, the cloud has made a lot of things easier. Much of it is built on open source technology. Linux is the most common operating system for public cloud services with a 90 per cent share. Open source databases make it easier to host and manage data. But do public cloud providers support open source enough, and do they provide the right kind of support?

It's no surprise that everyone who responded to our latest survey thought public cloud providers could do more. However, 58 per cent of respondents said competition from public cloud companies who use open source projects but don't contribute back was one of the top challenges currently faced by open source companies.

So, how should public cloud providers contribute to open source? The top priority according to respondents was providing better security, followed by encouraging open source collaboration, and improving existing code quality.

It's important that cloud companies contribute in the right way to help projects be successful. This support can also help the companies behind those open source projects develop and grow, too."

OPINION

OUT OF TIME...

Keith Edmunds
is MD of Tiger Computing Ltd, which provides support for businesses using Linux.

"IT projects are notorious for running over time and over budget (see, for example, just about any government IT project). This also applies to many IT projects taking place at SMEs (small and medium-sized enterprises) too, but it doesn't have to be this way.

Here's a suggestion for your next IT project, even if its aims and aspirations are a little more modest than introducing identity cards for the UK's population. Before you start, write a checklist that you can refer back to at the end of the project to ensure that the project has met its goals. You may be surprised at how hard that is to do.

Next, you then need everyone involved with the project to agree that the list is a full and complete one (again, something that's easier said than done).

Ultimately, you need to agree on the project's scope at some point, and it's cheaper and quicker to do so before you start the project. The agreement should be that any changes required after the checklist has been agreed will require the same people to agree to those changes (which may impact cost and schedule). It's a simple but effective strategy. Give it a try with your next IT project."

GAMING

Nvidia brings AI tech DLSS to Linux

Could result in performance gains – as long as you have the hardware.

CREDIT: CD Projekt Red

Cyberpunk 2077 benefits hugely from DLSS, and could run even better on Linux.

One of the most exciting announcements to come out of Computex 2021 was the reveal that Nvidia is working with Valve to bring Deep Learning Super Sampling (DLSS) to Linux via Steam Proton.

DLSS uses AI to upscale games, enabling gaming PCs to run games at high resolutions without the performance hit you'd usually see when running the games at those native resolutions. The fact that this will run with Steam Proton is particularly encouraging. Steam Proton is a compatibility wrapper, based on *WINE*, which enables you to run Windows-only games in Linux. DLSS could potentially improve the performance of games run via Steam Proton.

The good news is that this tech should be arriving on Linux soon, with Nvidia claiming that "Support for Vulkan titles is coming" in June, with "DirectX support coming in the Fall." Slightly less good news is that you need a recent Nvidia RTX GPU to make use of the tech – and those are hard to find these days due to the current global chip shortage. Hopefully AMD's rival FidelityFX Super Resolution tech will come to Linux, because it supports both AMD and Nvidia GPUs, including much older cards.

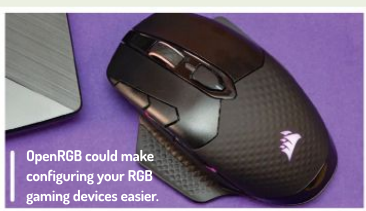
HARDWARE

Light 'em up!

OpenRGB is the one-stop solution for PC lighting.

Those into gaming will know that pretty much every gaming peripheral, from monitors to mice and keyboards, comes packed with RGB lighting. While having a gaming PC lit up like a hyperactive Christmas tree may not be to everyone's taste, there are a lot of gamers who approve of the lightshow.

However, most peripheral makers insist on using their own proprietary software to control RGB lighting of their products, and that means if you have devices from several manufacturers, you end up having to install lots of unwanted tools, and it's difficult to synchronise the lighting effects as well. *OpenRGB 0.6* (**http://bit.ly/LXF278OpenRGB**) looks set to fix that by enabling you to control all your RBG-enabled devices from a single program.



OpenRGB could make configuring your RGB gaming devices easier.

SOFTWARE

Year of the office!

Annual report reveals huge appreciation for LibreOffice.

The Document Foundation (TDF) has published its annual report for 2020 (which you can read at **http://bit.ly/LXF278TDF2020**), and as you'd hope, the entire report was made using its *LibreOffice* suite.

While 2020 was supposed to be a memorable year, with the 10th anniversary of *LibreOffice* being marked with events around the world, unfortunately, the year was instead memorable for something much different: the global Covid-19 pandemic. This meant those celebrations were cancelled, but the year was still a busy one. There were elections for the foundation's membership committee, and events to mark International Translation Day (*LibreOffice* is available in 119 languages thanks to the work of its community). The TDF also had a record year on the financial front. With many businesses and individuals having to work from home, downloads of *LibreOffice* skyrocketed, and that increase also meant an increase in donations. For the first time since *LibreOffice* started, donations exceeded €1,000,000.

So, while 2020 was a very difficult year for so many people, the fact that more people than ever have benefitted from *LibreOffice* is something to be celebrated.

Distro watch

What's down the side of the free software sofa?

RESCUEZILLA 2.2

The latest version of this Ubuntu-based distro, which is designed for backing up and rescuing PCs, has been released. This version brings one of the most requested features – a 'device-to-device' cloning option – as well as support for virtual machine images, making this already-useful tool even better. It can now restore from images made by all open-source imaging programs as well. There are also more ways to customise the compression format of your images as well. Find out more about this handy update at **http://bit.ly/LXF278Rescuezilla**.

This easy-to-use disk imaging distro could save yourself a lot of heartache if your system becomes faulty.

NIXOS 21.05

This independent distro is built entirely around the Nix package manager, including the kernel and configuration files, has a new version available to download. It brings updates to the kernel, along with the GNU Complier collection. The GNOME, Cinnamon and KDE Plasma desktops have all also been updated. Verskion 21.05 will be supported until the end of December 2021, which will then be replaced by 21.11. Head over to **http://bit.ly/LXF278NixOS** for details.

NixOS has had its kernel updated to 5.12, along with numerous other updates.

KALI LINUX 2021.2

Kali Linux is a popular Debian-based distro that focuses on security and forensic tools, and the latest update comes with two major new tools. *Kaboxer* is used for software developers to package their programs, which may have proved difficult to run in the past (for example, using legacy code such as Python 2) in a container. Meanwhile, *Kali-Tweaks* is a new tool that offers users greater customisation options for the distro. Find out more about these new tools, as well as the rest of the update, at **http://bit.ly/LXF278KaliLinux**.

Once known as BackTrack, Kali Linux is a great distro for security researchers.

JINGOS 0.9

JingOS 0.9 is now out. Based on Ubuntu for tablet computers, this new version brings some important changes to the user interface. One of the most useful new features is support for adaptive layouts. This means JingOS now runs 'perfectly' on devices with various screen resolutions. There's also improved virtual keyboard support, better wallpaper settings and support for complex passwords. To find out more, check out the release notes at **http://bit.ly/LXF278JingOS**.

JingOS is a great distro for running full Linux applications on tablet devices with touchscreens.

OPINION

BRIDGING THE DIVIDE

Rohan Garg
is a software engineer at Collabora

A new, low-overhead extension in Mesa means programs can mix OpenGL and Vulkan. This gives flexibility to developers while easing the transition path between the industry-standard Khronos APIs.

For the past several months we've been working with stakeholders within Mesa to bring the EXT_external_ objects functionality, which enables OpenGL and Vulkan to talk to each other on Chromium OS. The Iris driver, supporting modern Intel GPUs using Mesa's Gallium framework, was first to receive this support, and was followed by the i965 driver, supporting older Intel GPUs.

Implementing this feature enables developers to be able to use Vulkan for their rendering needs while using OpenGL for everything else, and while piglit tests pass, they are by no means a real world use-case. We're finding corner cases when running applications that make use of this new feature, which means we can expand our test coverage and make the code more robust. Application developers: do try out this new feature and report their findings to the Mesa community to help make the drivers more robust.

Bringing this feature to i965 and Iris enables Mesa to be competitive in the graphics world and merging this feature in allows for a very exciting future!

OPINION

STRAIGHT TALKING

Jon Masters has been involved with Linux for more than 22 years.

"With the summer months upon us, so too is the usual conference season, except these are still far from usual times. Several of the big annual Linux conferences have said they'll have to go virtual once again.

As Ted Ts'o put it, it just wouldn't be practical to hold the kind of discussion format in a socially distanced and safe manner. And while a growing number of countries are rolling out vaccination programmes, others are experiencing a far worse time, to say nothing of international travel constraints. Given the situation, it seems only reasonable to exercise caution, but the conversation on the Linux Kernel Mailing List (LKML) quickly became one of the more bizarre I've seen. An IT consultant from Germany replied with a rambling, incoherent rant, laced with conspiracy theories. These happen all too frequently on the LKML, but this time, Linus bit. His reply presents a concise summary of current mRNA vaccine technology, along with all of the reasons why people should go ahead and get vaccinated.

Given the times we live in, and all of the fake news that we have to contend with, it's refreshing to see people like Linus take a public stand. Who knows, maybe it will sway someone who had been hesitant?"

KERNEL WATCH

Jon Masters summarises the latest happenings in the Linux kernel, so that you don't have to.

Linus Torvalds closed the merge window for kernel 5.13, and released a series of Release Candidate kernels that have since settled down and stabilised ahead of the final release. Linus noted that there were "1,800+ developers and 14k+ non-merge [non-administrative] commits", which is on the larger side as Linux development cycles go. Still, it's been a fairly calm cycle so far without the usual kinds of scurrying that can happen by the time rc3 comes around.

As we reported last month, 5.13 gains initial support for Apple M1 bare metal hardware. In addition, it also includes a new option to randomise the location of the kernel stack on every system call, and the removal of the legacy **/dev/kmem** special file that traditionally allowed (privileged) applications to poke directly into kernel memory. While that might have helped old-school X Window servers and the like, it hasn't been needed by the vast majority of users for a long time, and is an obvious target for security compromises. Removing such old interfaces is usually for the best.

In ongoing development, an update was posted with plans for CXL (Compute Express Link, an enhanced bus that layers on top of PCIe and provides such features as an alternative to DDR slots for memory expansion in future server CPUs) enablement in the 5.14 cycle. There was also a fairly lengthy discussion about improving the robustness of virtio drivers in the kernel now that it has support for a number of (competing) Confidential Computing implementations that provide encrypted VMs intended to be safe from host hypervisor interference. In such a world, the guest can't trust the host "hardware" not to be malicious, requiring that drivers are carefully audited for unsafe/insecure assumptions.

The RISC-V KVM saga

A long running saga came to a very public boil recently with a discussion of the state of Linux support for the RISC-V hypervisor "H" extension and the reasons why it hasn't yet been merged into the kernel, in spite of having several years of development. As readers probably know, RISC-V is an open source ISA (Instruction Set Architecture). This means the organisation which holds copyright and ownership over the architecture (RISC-V International) is a non-profit entity that enables anyone to build RISC-V processors, royalty free.

A popular confusion is that this means all such processors are open source, which isn't the case. No requirement exists for that, only that they can be open source. The architecture is built from a base instruction set and a series of extensions that are "ratified" by certain members of RISC-V International. At least, that's how it's supposed to work. The problem is that the "H" extension which supports virtualisation has been under development for several years, and has neither been ratified nor received many recent updates.

The Linux RISC-V maintainers generally don't like to merge code into Linux for non-final extensions, because changes to them later would break hardware/software compatibility and cause Linux not to run on some platforms. At this stage, there are no public RISC-V platforms on the open market implementing the "H" extension, although there are **qemu** patches and other "soft" designs that can be tested using FPGAs and hardware emulation platforms (that can also be updated if the extension evolves prior to ratification) ahead of real silicon.

It's actually commonplace in the silicon world to test out different proposed hardware designs ahead of building the actual silicon. Such emulation platforms run slowly, but they're versatile and they can be updated. Generally, across the industry, great effort goes into not having hardware designs implement things that aren't carefully locked down and agreed upon, due to the cost of any potential breakage, but it seems there's a real risk of this happening within the RISC-V ecosystem, with hardware coming ahead of official standards.

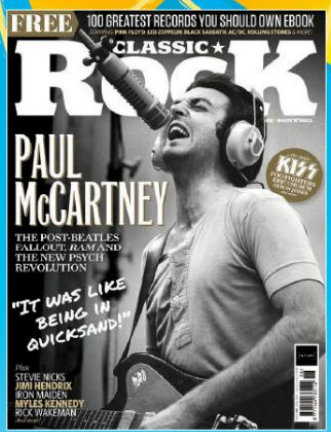
Without an obvious path forward for the "H" extension to become official, code implementing KVM support cannot officially be merged into the kernel. An attempt to get it in via the "staging" driver work in progress tree maintained by Greg K-H (Kroah-Hartman) was met with a predictable backlash because he felt that it was inappropriate to do so ahead of hardware. He wished all involved good luck getting this through the committees and into hardware (and thus Linux). LXF

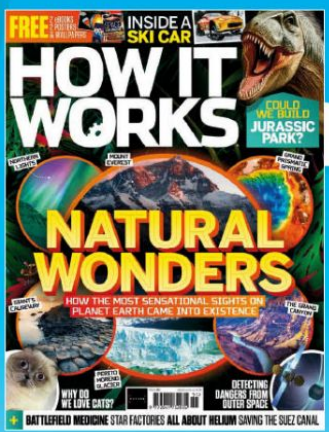
SUMMER SALE

6 ISSUES

for just

£9.99





SEE THE FULL RANGE

www.magazinesdirect.com/summer21

Answers

Got a burning question about open source or the kernel? Whatever your level, email it to **lxf.answers@futurenet.com**

Neil Bothwick is a timely triple-page troublesome Tux tinkerer.

Modem muddles

I live in the US, and I had to upgrade my modem to one supporting DOCSIS 3.1 to access the faster speeds from my cable provider. I have two Linux Mint PCs and two Windows 10 PCs. After changing modems, the Windows PCs would connect to the internet, but neither of the Linux PCs would. I tried two different modems: an Arris SB8200, and Netgear Nighthawk. I called support to my ISP, and the modem manufacturers, but nobody could help. I was never able to get the internet to work, so I had to put the old modem back. After doing that, everything started working again. I kept the Arris modem, just in case someone could figure it out.
Larry

A Looking at the documentation for the Arris modem, it should work. The manufacturers state that Linux is supported, so this is likely to be a configuration issue. One point to bear in mind is that although the modem has two Ethernet ports, you can only use one of them unless your ISP is providing you with two IP addresses, because this is only a modem and not a router.

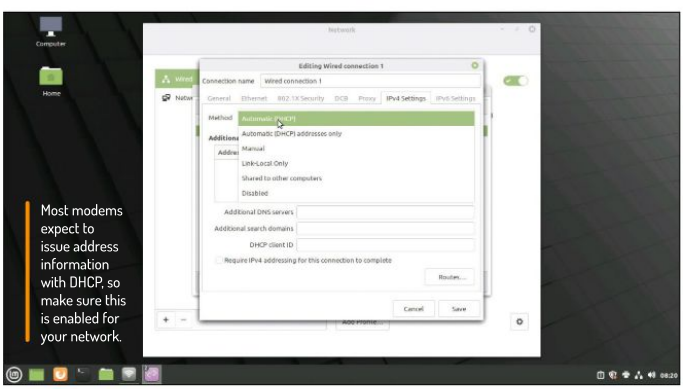
These devices usually use DHCP to assign an address to a connected computer and to provide it with routing details, so make sure your Linux systems are set up to use DHCP rather than statically assigned addresses. Then see what the system logs says about your attempts to connect. Check the modem is powered on and connected to the Internet, then open a terminal on Mint and run

```
$ sudo journalctl -f
```

This will show your system log as added entries. Now connect the Ethernet cable to the computer and modem and watch the log output. It may help to maximise the terminal window to see as much as possible. This should display the attempt to set up a network connection and show where it's failing, and hopefully give a clue as to what is causing it.

However, with four computers, you'd be better off with a router connected to the modem, then connect your computers to the router. That would mean they could all access the Internet at once with the added benefit that they can also communicate with one another, making file transfer between them much simpler. Many routers also include wireless connectivity. Even if you don't use a laptop, this could save on data costs for your mobile phone.

Just about any router intended for home or small office use will be suitable. The router than takes care of providing the DHCP information to the computers, while the modem sees only the one device connected. This approach is recommended in the Arris documentation.



Iffy Wi-Fi

I need help to get an old laptop to automatically start Wi-Fi. I have an old Dell Inspiron 1300, which was running Ubuntu 16.04 but slowly. As a 32-bit CPU with 2GB of memory I tried to load what I thought was going to be a smaller/lower overhead version of Linux. I got into a bit of a mess and after a lot of web searching (on another PC) decided to reload Ubuntu 16.04 (32-bit).

I now have the old laptop installed with Ubuntu 16.04, but can't get the Wi-Fi to start up automatically. After again looking online I now have a workaround, but it means that when I want to use the laptop with a Wi-Fi connection I have to carry out the following. With the computer powered on and with Ubuntu running I open a Terminal and type

```
$ sudo apt install firmware-b43-installer
$ sudo apt install linux-firmware
$ sudo modprobe b43
```

The Ubuntu system then finds and connects to our Wi-Fi, and all works well. I'm at a loss to be able to ask the right questions of how to get Wi-Fi to start automatically at startup. Which files need to be edited, what needs to be written in to which file and where should these files reside in the FSH?
WH

A Is Ubuntu the best distro for your laptop? Something more lightweight, such as Lubuntu, may be more suitable for something with 2GB of RAM, or you could install a lighter desktop on your existing Ubuntu system.

Normally, you'd expect Linux to identify your hardware and load the appropriate module. Your first two steps are only needed once: to install the files required for your wireless card. Then the module should be loaded automatically at boot. However, Broadcom wireless adaptors have been a little hit and miss in our experience and it looks like this is the case here; the system needs to be instructed to load the module.

There's a place to include modules that need to be loaded, on distros using systemd, which is in **/etc/modules-load.d**.

Create a file in this directory – the name doesn't matter as long as it ends in .conf but use something descriptive like **wifi.conf** or **b43.conf**. The modules you want to load are simply listed one per line.

Older systems put it all in the file **/etc/modules**, but the syntax is the same. Once you've done this the module will be loaded when you boot and the wireless interface should be available.

Many laptops have the wireless adaptor on a mini-PCI card, so it's possible to swap it out for something a little more Linux-friendly. We'd replaced the Broadcom adaptor with an Intel one in this laptop.

Your computer's BIOS has no concept of time zones, so set wake-up times in UTC rather than local time.

Q Missing hours

I've set the BIOS option of wake-up by time, but the computer turns on exactly two hours afterwards. Both my BIOS clock and the Linux clock show the same time. I'm on Sparky Linux rolling (Debian testing). I'd appreciate any suggestions on this. Could it be a locale issue?

Jørgen Brandt

A It is indeed a locale issue, but probably not quite as you imagine. Linux, unlike certain other operating systems, keeps the system clock at UTC and makes adjustments for your time zone whenever you use it. This means that a multi-user system can comfortably have users in different time zones and the clock will always be correct for each of them. However, the BIOS has no concept of time zones or locales – it just knows the time stored in the clock. So if your time zone is at UTC+2, which it seems to be, when you tell the BIOS to power up at 8am, it's actually 10am in your locale.

There are two ways to deal with this. One is to set your system to set the hardware clock to local time rather than UTC, then when your clock says 8am, so does the hardware clock. This was needed to dual-boot with Windows in the past because it was unable to cope with a hardware clock on UTC. The other option is to recognise that your hardware clock is always two hours behind local time and simply set the wake up time in UTC.

You may also need to adjust the setting twice a year to cater for the start and end of daylight savings time.

Q Sign here please

I have installed *Let's Encrypt* on my koozali mail/web server exactly according to the instructions at https://wiki.koozali.org/Letsencrypt. However if anybody (including myself) tries to access the server, they are told that the certificate is invalid because it's self-signed.

How do I get round this problem apart from paying megabucks annually for a commercial certificate?

Dusty Pulver

A *Let's Encrypt* shouldn't generate self-signed certificates. The main point of the service is to give everyone access to authenticated certificates without shelling out wads of cash. There are two main possibilities: either the certificate was generated incorrectly or your web server is pointing to the wrong certificate.

Did you use a self-signed certificate for testing before generating one with *Let's Encrypt*? if so, it's possible that your web server is still trying to use this. Check all of your server configuration files for references to this certificate. You should also delete or move the directory containing the certificate before restarting the server.

If it's still trying to use the old certificate, it should output some error messages as it can no longer find it. Once you have made sure everything points to your *Let's Encrypt* certificate, restart the server (or reboot) and try again.

If you continue to experience the same error, you can use the `openssl` command to check the certificate, like this:

```
$ openssl x509 -text -noout -in /etc/
letsencrypt/live/HOSTNAME/chain.pem
```

You'll need to change the path if your certificates are stored in a different location. The output should contain something along the lines of:

```
Authority Information Access:
  CA Issuers - URI:http://apps.identrust.
com/roots/dstrootcax3.p7c
```

This will be present in a correctly signed certificate but not in a self-signed one. If it's not present, you should delete the certificate files (if this is the only *Let's Encrypt* certificate on the system, delete the whole **/etc/letsencrypt directory**) and start over with creating a new one. If you

» A QUICK REFERENCE TO... DAEMONS!

Many programs, especially background daemons, can get very chatty, sending all sorts of details about what they're doing to log files. This can either be the system log or individual log files, generally stored in **/var/log**. As the weeks and months go by, this can build up to a substantial amount of data, and disk space consumed.

If you use systemd's journal as the system log, it takes care of this by compressing or discarding old data to keep disk usage within limits. By default, this is to use no more than 10 per cent of the filesystem's space and to always leave at least 15 per cent of that space free. You can adjust these settings in **/etc/systemd/journald.conf**.

For other log files we have *logrotate*. Log files are text only, so very compressible. *Logrotate* compresses each of the current files, saving it with a .1.gz suffix, and start a new file. The defaults are to do this weekly and to use *gzip* for the compression. *Gzip* isn't the most effective compressor, but it is fast, which is usually more of a consideration; you can change this in **/etc/logrotate.conf**. *Logrotate* keeps the last four compressed logs, but you can change this and number them with .1 being the most recent. You can set the dateext option in **logrotate.conf** if you want it to use a datestamp instead of a number. The rotate option with a number sets how many logs are kept.

end up with an unsigned or self-signed certificate again, you should raise the issue with *Let's Encrypt*.

Q My Brother ignores me

I have a shell script code that prints any new PDF generated to a Brother printer that's connected to a Apple Mac computer using

```
lp -d Brother_HL_L2350DW_series /var/
pdfs/11.pdf
```

After two or three days the printer stops to print and I see the printer queue growing with the error message 'The printer is not responding'. See below:

```
tail /var/log/cups/error_log

E [26/Apr/2021:13:38:45 +0200] [Job 15850]
The printer is not responding.
E [26/Apr/2021:13:39:45 +0200] [Job 15850]
The printer is not responding.

sudo journalctl -b 0 -u cups
-bash: journalctl: command not found
```

If I manually power off and power on the printer it works and starts the pending prints. Thus my requirement is to have Linux MacOS commands to reset/restart the printer using the command line; and reprint the Printer Queue.

Robert Pratt

A I have a similar problem with an HP printer that occasionally becomes unavailable. Fortunately, I also have a tried and tested solution. CUPS has command line tools to manage printers, two of which are *cupsdisable* and *cupsenable*. So all you need to do is run these two commands on the machine running the printer:

```
$ cupsdisable Brother_HL_L2350DW_
series
$ cupsenable Brother_HL_L2350DW_
series
```

See the man pages for these commands for their various options, but this should be all you need. There's no need to restart the printer queue – *cupsenable* does this automatically, unless you ran *cupsdisable* with the `--hold` option.

Because you need to run the commands on a remote server, the simplest way is to create a script on the server, say **/usr/local/bin/resetprinter**, containing the above commands. It may be helpful to add a `sleep 2` command between to allow time for the disable to complete before re-enabling.

Provided you've set up SSH keys to allow passwordless logins, you can then add this to your **.bash_profile** file:

```
alias lpreset ssh root@macserver /usr/
local/bin/resetprinter
```

Then you only need to run `lpreset` in a terminal to restart the printer.

This works for me, but if your problem is more deep-seated and you really do need to power cycle the printer, you'll need some extra hardware in the form of a remote-controlled power switch. Some of these run open source software, such as *espurna* or *tasmota*, and can be controlled from a web browser, the command line or a script. However, setting up something like this is too much to go into here.

The lack of a `journalctl` command is to be expected because this is part of systemd, and systemd is only used on Linux, not MacOS. Instead, you should use *grep* to search the system log file.

Q Parenthetical problem

I'm writing a simple script to let *bc* evaluate simple maths expressions and output on the screen. I write down a file called mybc as

```
#!/bin/sh
express=$1
echo "scale=6; $express" | bc
```

and run it on the command line as

```
mybc sqrt(6.0)
```

But I get the following error message

```
bash: syntax error near unexpected token
`('
```

If I run it on the command line as

```
mybc sqrt\(6.0\)
```

it works correctly, but I hope *mybc* can just run as the first way.

Alfie Schofield

A The "problem" isn't with your script but with the shell from which you're running it. Most shells, including Bash that most distros default to, interpret parentheses as special characters, so they're not even passed to the script. That's why the error message you're getting is from Bash, although this may be unclear because the script is probably also calling bash via **/bin/sh**. If you tried passing arguments to a Python or Perl script in this way, you'd also get a bash error.

Escaping the parentheses as you tried does mean they're passed to the script, but there's a simpler approach: not using parentheses at all. Bash scripts and functions accept arguments as a space separated list, and these available in the script as $1, $2 and so on ($0 contains the name of the script). The simpler solution is to write the script so that it's called as:

```
mybc sqrt 6 0
```

The number of arguments is available in `$#` so you can still cope with varying numbers of arguments. You should also look at the shift operator in the Bash manual, which can be useful in stepping through arguments. LXF

The CUPS web interface provides options to pause and resume a printer, but you can also do this from the command line with cupsenable and cupsdisable.

GET HELP NOW!

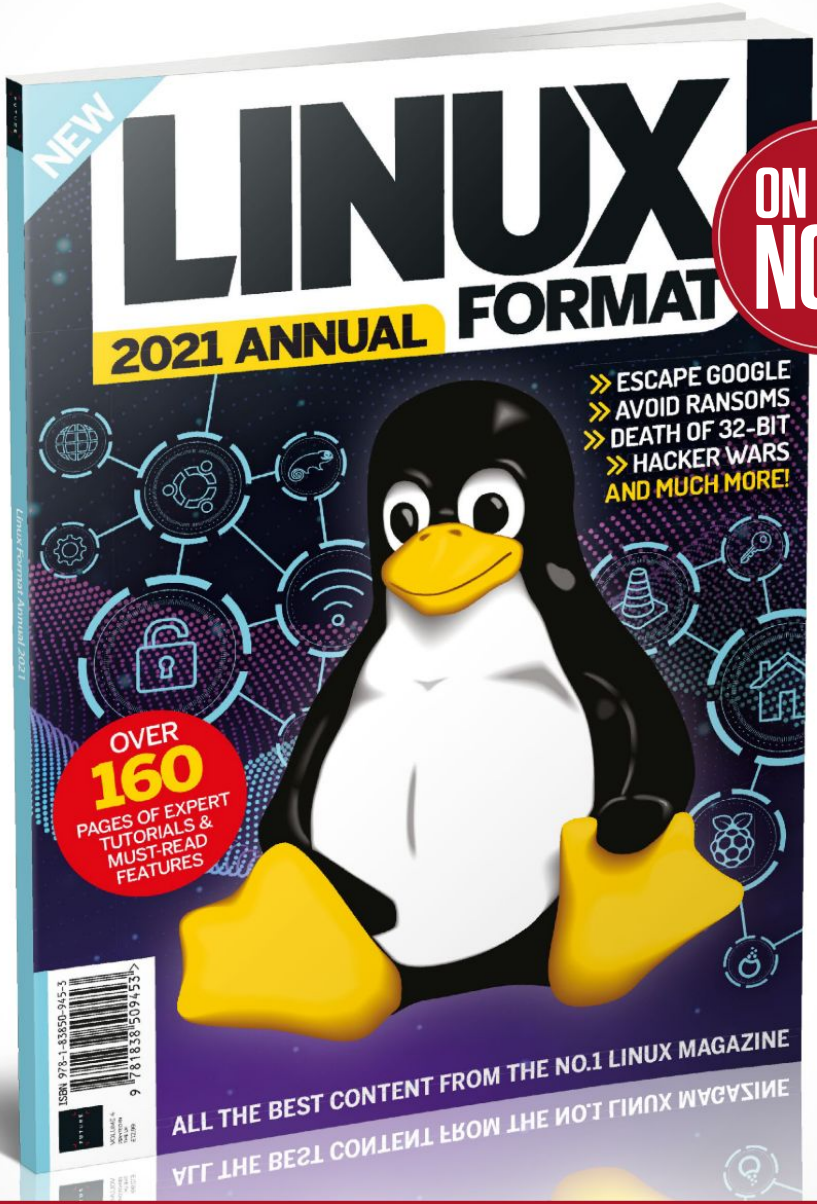
We'd love to try and answer any questions you send to **lxf.answers@futurenet.com**, no matter what the level. We've all been stuck before, so don't be shy. However, we're only human (although many suspect Jonni is a robot), so it's important that you include as much information as you can. If something works on one distro but not another, then tell us. If you get an error message, please tell us the exact message and precisely what you did to invoke it.

If you have, or suspect, a hardware problem, let us know about the hardware. Consider installing *hardinfo* or *lshw*. These programs list the hardware on your machine, so send us their output. If you're unwilling, or unable, to install these, run the following commands in a root terminal and send us the **system.txt** file too.

```
uname -a > system.txt
lspci >> system.txt
lspci -vv >> system.txt
```

TAKE COMPLETE CONTROL OF YOUR LINUX BOX

Got root? Then get the Linux Format Annual and get the most from your distro and its free and open source software today!



ON SALE NOW

FUTURE

Ordering is easy. Go online at:

magazinesdirect.com

Or get it from selected supermarkets & newsagents

Mailserver

WRITE TO US

Do you have a burning Linux-related issue that you want to discuss? Write to us at *Linux Format*, Future Publishing, Quay House, The Ambury, Bath, BA1 1UA or email **lxf.letters@futurenet.com**.

Evil squids

The download link for the code to *Escape From The Evil Space Squids!* on page 48 of LXF275 just gives a 404 error (https://github.com/lesp/LXF275-PiUserSpaceEscape/archive/main.zip) Can you help?
Brough

Les Pounder says...
Well that is odd. Looks like Github has updated the URL, so what we printed was correct at the time of print. The corrected link is **https://github.com/lesp/LXF275-PiUser-SpaceEscape/archive/refs/heads/main.zip**.

Nostalgic

Loved the article on the Commodore PET. I used a PET in my first year computer science lab to learn to program, using Pascal, in 1984. It had a 5.25-inch floppy disk and overheated if used for more than an hour or so. The uni replaced them the next year with Sinclair QLs with their horrible tape micro-drives and clunky keyboards. Personally I preferred the PET!

I also enjoyed the article on Forth. The monitor program on my old Sun SPARCstation was Forth based. One of my colleagues wrote a graphical *Pac-Man* style game using it during his lunch breaks.

The Commodore PET and its iconic shape.

CREDIT: Rama. CC BY-SA 2.0 FR. https://commons.wikimedia.org/wiki/File:Commodore_2001_Series_IMG_0448b.jpg

A very different language for sure. I also used a Novus calculator from 1976 to the mid-80s that used Reverse Polish Notation. So many memories in one issue!
Alan Gauld

Neil says...
I usually try not to have quite this much nostalgia in one issue – it's a heady drug that we can overdose on – but it was all proposed at once and so was hard to turn down! Glad you're enjoying it. It's lovely being able to relive this stuff from the safety of 2021?!

linuxformat.down?

I wonder why the IP-address of the *Linux Format* website has changed? For a couple of days I couldn't connect to www.linuxformat.com. What's happened?
Jan Pihlgren

Neil says...
As Jonni documented in great detail in **LXF277** the old *Linux Format* Archive server required a drastic upgrade. It wasn't entirely out of the blue, but hosting has switched from UKFast to AWS, which quite frankly was long overdue. It's never actually been done before, because every time the topic came up with Future's lovely Dev-Ops team, it seemed they had to run off with an urgent task of rebooting **TechRadar.com** or something similar. So Jonni did it instead.

So retro

In a recent issue (**LXF275**) Jonnie Bidwell says that RetroPie is arguably the best emulation platform available. What is the difference between RetroPie and RetroArch, and can you reveal why Arch-Lovin' Jonnie prefers RetroPie?
Chris Williams

Jonni says...
Did I say that? Incendiary. Fortunately, RetroPie is a Pi distribution that uses RetroArch (a cross-platform middleware that works with many emulators via the libretro library) under the hood and *EmulationStation* as a front-end. So the rabbit hole is deep, and I should've said that RetroPie is arguably the best Pi emulation

Helpdex

RETRO ARCH

Powering more things than you might expect.

platform available, and I should also have mentioned Lakka. Setting up game controllers, ROMs, network shares and all the different emulators by hand is quite a lot of work, so I really admire how RetroPie unifies it with friendly menus (while still enabling you to manipulate the underlying OS).

I do play the odd Amiga game via *FS-UAE* on Arch, in theory I could do this with RetroArch and the *puae* core (the libretro port of UAE), but I ran into some annoyances with multi-disk titles. Either way, the emulation of disk sounds is weirdly soothing. Grrrrr chug chug-clunk.

Totally sane

I find it so refreshing that you act on comments from readers, such as the subscription renewal this month. So many organisations either ignore suggestions or make excuses.

Sorry I've been slow getting back to say thank you for answering my question about SANE (LXF275). It still wouldn't work, but your answer pointed me in the right direction.

I finally recalled an issue with permissions that occurred some while ago which I thought I had resolve it by some suggestions found online. Clearly it had not fully sorted it. I bit the bullet and did a complete re-install and that fixed the issue.

My next step has been to order a book which I hope will be a good investment. *How Linux Works, 3rd Edition: What Every Superuser Should Know*, written by Brian Ward.

Dave Fisher

Neil says...

Thank you for the kind words. Honestly, we do want your feedback and do listen, although it can take a few months for suggestions to actually trickle through to finished articles because of how slowly the wheels of dead-tree print publishing take to turn. LXF

LETTER OF THE MONTH

More maps

Thank you for the article on OpenStreetMaps. I've been a contributor since I bought a GPSR in 2009. I do think there are a couple of related programs that deserve a mention. If you have a Garmin GPSR, you can produce your own maps using freely downloaded data and the *mkgmap* utility. With a little tuning, you can change the way roads, areas and points are rendered to suit your preferences. The maps produced are up to date, more detailed and don't include the copyright monitoring "artefacts" found in commercial mapsets.

And there's an OSM-based mapping application – *OsmAnd* – which operates from an offline internal map database, so there's none of the "phone home to the mothership" constant reporting of your location, as involved when using Google Maps or Waze. It's also possible to produce your own maps for *OsmAnd*.

Here are some useful links for those interested:
www.mkgmap.org.uk – for more details on how to use *mkgmap*.
https://osmand.net – how to use *OsmAnd* for offline maps.
http://katze.tfiu.de/projects/phyghtmap – use topographical data from NASA to create OSM contour lines.

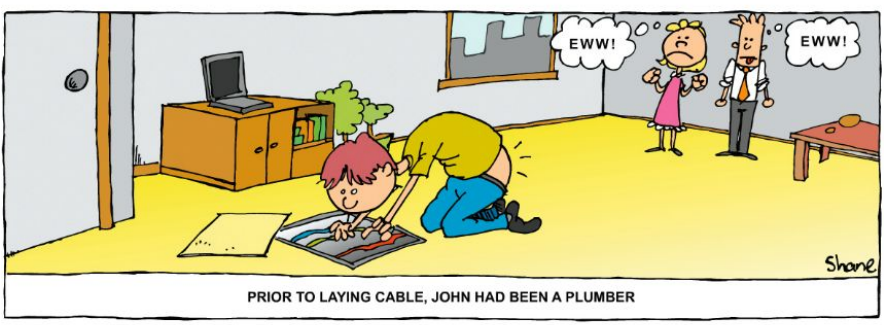
Finally, if you want to produce contour overlays then visit http://download.geofabrik.de for daily extracts of the OSM database.

Jon Hobden

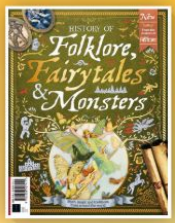
Neil says...

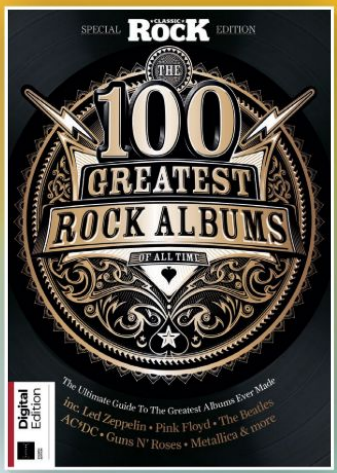
Thanks for the recommendations Jon! We were planning on running a few more mapping tutorials down the line, but it's one of those speciality topics that's hard to cover well without the specialist knowledge... but also because OSM is a crucial project that we've been badly neglecting, doubly bad as I use OSM for walks – I find the detail better than Google, though as Mike Bedford points out, the UK's Ordnance Survey maps are pretty good. Anyone else have any experiences to share?

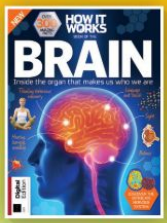
Making top-quality mapping available freely to everyone.

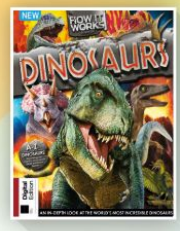


shane_collinge@yahoo.com



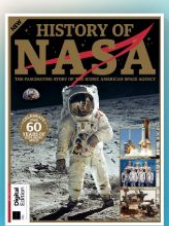






DISCOVER OUR GREAT BOOKAZINES

From travel and history to gaming and photography, you're certain to find something you're passionate about



Follow us on Instagram @futurebookazines

FUTURE

www.magazinesdirect.com

Magazines, back issues & bookazines.

Aorus Gen4 7000s

Gigabyte's SSD is the fastest storage this side of **Sean Webster's** oiled drawers.

SPECS

Capacity: 2TB (1TB)
Form: M.2 2280
Interface: PCIe 4.0 x4, NVMe 1.4
Logic: Phison PS5018-E18
DRAM: DDR4
Memory: Micron 96L TLC
Seq. read: 7,000MBps 2TB, 1TB
Seq. write: 6,850MBps (5,500 1TB)
Ran. read: 650,000 IOPS (350,000 1TB)
Ran. write: 700,000 IOPS 2TB & 1TB
Security: 256-bit AES
Endurance: 1,400TB (700TB 1TB)
Warranty: Five years

Gigabyte's Aorus Gen4 7000s SSD has a well-crafted heatsink that's decked out with plenty of fins. We'd expect nothing less for a device that can gulp down over 8.5 watts of power during the harshest of workloads. It's designed to compete with the best SSDs, and dishes out up to 7GBps and improves over earlier Phison E18 NVMe SSD models. Meanwhile, its nanocarbon-coated heatsink aims to reduce temperatures by 20 per cent.

Behold the aluminium heatsink.

The drive comes in two capacities, 1TB and 2TB. Each capacity is rated 7,000MBps read, but the 1TB delivers 5,500MBps write and the 2TB model 6,850MBps. Both have a five-year warranty and come with respectable write endurance ratings – up to 700TB per 1TB in capacity. It's packaged in a M.2 2280 double-sided form factor. The included aluminium heatsink measures 11.5x 23.5x76mm with the SSD sandwiched between two thick thermal pads. The device is powered by Phison's second-generation PCIe 4.0 x4 SSD controller, the PS5018-E18. It leverages DRAM and features a triple-core architecture that's paired with the company's CoXProcessor 2.0 technology (an extra two R5 CPU cores) for fast and consistent response. The main CPU cores are Arm Cortex R5's clocked at 1GHz, up from 733MHz on its predecessor, the PS5016-E16.

Setting the pace

The Aorus recorded our fastest read test ever and second-fastest copy, only just pipped by Samsung's 980 PRO 2TB. Synthetic sequential read/write performance was very good across all block sizes. At a QD of 1 and 1MB block size, the Aorus 7000s reads much faster than the InnoGrit-powered Viper VP4300, but lags behind the Samsung and WD. When writing, however, it scores top marks, writing at roughly 5,850MBps. In terms of its random responsiveness, the Aorus is just as fast as the Rocket 4 Plus, but not quite matching the Samsung, WD, or even the Adata at a QD of 1.

The sequential write test shows the Aorus is one of the fastest filling SSDs we've had our hands-on. While it doesn't show a full disk dynamic cache, it's seemingly more conservative behaviour enables fast performance. When tested, the SSD wrote 464GB of data at a speed of 6.9GBps before degrading to 1.8GBps for another 1.18TB before again degrading to a final 1,150MBps average for the remainder of the test. SLC cache recovers at a rate of roughly 88GB every five minutes.

Power wise it runs more efficiently than the Rocket 4 Plus, but trails the other Phison models. It gulps down the most power on average, with peak power consumption pegging 8.5 watts. At idle, power consumption is well regulated at 800mW, second-lowest to the Samsung 980 Pro. When idle the SSD measures 40C with no airflow, after copying half its capacity, the surface temperature measured 78°C with no thermal throttling taking place.

For those on the lookout for a heatsinked SSD that can keep up with and even trade blows with the best, Gigabyte's Aorus Gen4 7000s is a solid offering. LXF

Phison's second-generation controller keeps things in check.

VERDICT

DEVELOPER: Gigabyte
WEB: www.aorus.com
PRICE: £360 (£190 for 1TB model)

FEATURES	9/10	EASE OF USE	9/10
PERFORMANCE	9/10	VALUE	8/10

It comes with all the features you could ask for from a high-end enthusiast-grade NVMe SSD and is affordable, too.

» Rating 9/10

OSGeoLive 14.0

Being cooped up in his home office, **Mayank Sharma's** quest to travel the world leads him to a distro that brings the world to him.

IN BRIEF

A Lubuntu-based distro that offers some of the finest open source geospatial tools along with relevant datasets. Instead of simply piling them into the distro, the developers have configured them to be usable right out of the box. OSGeoLive is supported by several reputable projects and organisations, and supports multiple languages.

SPECS

Minimum
CPU: 1GHz
Memory: 1GB
HDD: 20GB
Build: 64-bit ISO and VM

OSGeoLive is a Lubuntu based distro that pulls together a library of open source tools and applications to support all kinds of geospatial use cases. The distro isn't designed as a general-purpose distro, but rather as a daily driver for individuals who work with geospatial tools.

Geospatial refers to all data and the associated tech that's used to acquire, manipulate and store geographic information. Geospatial tools are used by data specialists, science researchers, cartographic professionals and map hobbyists.

There's no dearth of robust open source geospatial tools. The OSGeoLive distro ships with several of them, which the developers claim are pre-configured to suit a range of geospatial use cases. The distro also caters to new users by helping them sort through the myriad options and find pre-configured tools to suit all workflows.

The OSGeoLive project is backed by the not-for-profit OSGeo Foundation, which also supports the development of several open source geospatial projects. It also promotes the use of the distro for educational purposes and for use in the real-world.

Desktop includes a file that lists the default usernames and passwords for the various pre-configured applications.

Charting the world

OSGeoLive is distributed as a traditional ISO image that can be copied on a DVD or a USB drive. The advantage of copying the image to USB is that you can enable persistence and save any changes you make in the Live environment. The project also puts out a vmdk virtual disk that can be attached to VMware and VirtualBox virtual machines.

OSGeoLive only accepts established open source projects, but still manages to include about 50 geospatial tools, along with several free world maps and sample datasets. The specialised programs are all housed within the Geospatial meta category, which is further subdivided into relevant groups such as Desktop GIS, Navigation and Maps, Web Services and more.

Each category lists a handful of tools. While they wouldn't be of use to the casual user, the fact that these are pre-configured goes a long way in helping the geospatially inclined users to use them for productive purposes without delay.

There are traditional desktop GIS programs for viewing, editing and analysing geospatial data. Then there are browser-based clients, some of which offer functions that were previously only available in desktop applications. The distro also includes web services that can be used to fetch map data in various formats.

The Navigation and Maps category has some familiar tools such as the *Marble* virtual globe, as well as *OpenStreetMaps* along with its *JOSM* and *iD* editors.

Supplementing the geospatial tools are several sources of free mapping data. For instance, there's one that provides cartographers with public domain maps that helps them create small-scale world, regional and country maps, and another that includes both historical and predicted temperature and precipitation details.

According to the release notes, the latest release has added a couple of new tools, besides improvements to the distro's documentation and translations.

For package management, OSGeoLive relies on the *Muon* package manager, which uses the default Ubuntu repositories along with the OSGeoLive's PPA. For installation, the distro uses the standard Lubuntu installer, which is *Calamares*.

OSGeoLive is one of those distros that's more than a sum of its parts. It isn't just a collection of the best open source geospatial tools, but has instead been carefully put together to ensure users can access the tools straight after booting into the distro. LXF

VERDICT

DEVELOPER: OSGeo Foundation
WEB: https://live.osgeo.org/en/index.html
LICENCE: LGPL

FEATURES	**8/10**	**EASE OF USE**	**8/10**
PERFORMANCE	**8/10**	**DOCUMENTATION**	**8/10**

The perfect daily driver for anyone who works in the geospatial space, either as a hobbyist or professionally.

» Rating 8/10

Bodhi Linux 6.0

Mayank Sharma isn't embarrassed to admit that there's something about Enlightenment-based distros that makes him go weak in the knees.

IN BRIEF

An Ubuntu-based distro that combines Ubuntu's long-term support (LTS) release with its custom Enlightenment-based Moksha desktop, to create a lightweight-yet functional installation. Bodhi isn't exactly designed for Linux newbies, but doesn't involve a steep learning curve, either.

SPECS

CPU: 1GHz
Memory: 1GB
HDD: 5GB
Builds: Currently only 64-bit; 32-bit remains TBC

Bodhi Linux 6.0 is the distro's first major release under the new management after its founder handed over the reins of the project to its community in 2019. Although built on top of Ubuntu releases, this one being on the Ubuntu 20.04.2 LTS Focal Fossa release, Bodhi Linux isn't just another Ubuntu derivative.

For starters it's one of the handful of distros that uses the Enlightenment desktop. Actually, that's not factually correct, since Bodhi uses its custom Moksha desktop that's based on Enlightenment, with certain changes. For one, Moksha has tamed Enlightenment's peculiarities to draw a desktop that adheres to the traditional desktop metaphor and doesn't look alien even to first-time users. Importantly however, Moksha has retained Enlightenment's ability to draw a classy desktop without taxing the hardware. Indeed, its Moksha desktop means Bodhi pitches itself as a viable alternative for older, underpowered PCs. The desktop environment has had numerous improvements and a few new features added in this release, with Bodhi's iconic Arc-Green theme having been overhauled under the hood.

All three Bodhi Linux editions are currently available as 64-bit releases only, although a 32-bit version based on Debian is currently in the works.

Elegance in motion

The one major visible difference is that it now sports an animated background. The release also sports a new Plymouth theme. This drops you to the login screen, which uses an elegant new greeter.

The other major change is the replacement of the *PCManFM* file manager the project borrowed from the lightweight LXDE desktop environment, with a custom-patched version of Xfce's *Thunar*. While *PCManFM* is lighter than *Thunar*, the latter offers a far richer set of modern futile management features.

Another highlight of the release is the new notification applet in the panel at the bottom, or shelf in Moksha/Enlightenment parlance. The gadget helps keep track of all notifications from the various programs as well as the distro itself, along with their timestamps. By default, it records notifications of all programs, but it has some useful configuration options, including the ability to blacklist certain tools and components that you don't want to send you notifications.

Configurability is the hallmark of the distro and virtually all aspects of the Moksha desktop including all its components offer useful tweakable parameters. The developers have piled on some additional options to the existing applets to add to their configurability.

Bodhi Linux developers go to great lengths to maintain the distro's lightweight nature without overly compromising on its usability. One of the core tools that's key to delivering Bodhi's vision is the *Leafpad* text editor. When *Leafpad* was dropped by upstream Debian and Ubuntu repositories because of certain issues, Bodhi Linux developers took it upon themselves to patch it in order to continue shipping the program.

In another similar show of defiance, Bodhi includes the *Chromium* browser by default. The web browser is key to Bodhi, thanks to its unique web-based app store, and the developers feel *Chromium* does a better job than other alternatives. Notably however, the distro doesn't use Ubuntu's *Chromium* snap.

Bodhi Linux is available in three editions. The regular edition is a minimal installation that's meant for users who want to craft their installs from scratch. Unless you have a fairly dated machine, you should use the HWE (Hardware Enablement) release that ships with a newer kernel. For a more usable installation, there's the AppPack release that's pre-installed with lots of useful everyday desktop productivity programs. LXF

VERDICT

DEVELOPER: Robert Wiley and others
WEB: www.bodhilinux.com
LICENCE: GPL and others

FEATURES	**8/10**	**EASE OF USE**	**8/10**
PERFORMANCE	**9/10**	**DOCUMENTATION**	**8/10**

Despite its lightweight nature Bodhi Linux should be seen as a viable daily driver on modern PCs as well as on older kit.

» **Rating 8/10**

GeckoLinux Rolling

With openSUSE being the only real KDE distro that he's ever found tolerable, **Mayank Sharma** wonders if a spin can add anything to that experience.

IN BRIEF

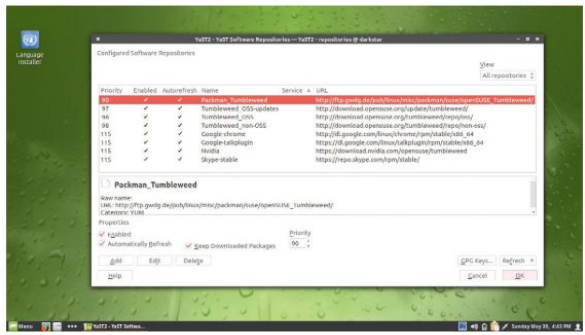
GeckoLinux turns a stock openSUSE release into a desktop oriented spin. Broadly speaking, GeckoLinux is to openSUSE what Linux Mint is to Ubuntu, or what Korora was to Fedora. Managed by a single anonymous developer, GeckoLinux is surprisingly polished and available with all the popular desktop environments.

SPECS

CPU: Any 64-bit processor
Memory: 1GB
HDD: 10GB
Build: 64-bit

OpenSUSE is one of those distros that regularly slips through the cracks, thanks mostly to its once-a-year release cycle. Despite our infrequent coverage of the distro, we've always praised it as one of the most polished KDE renditions.

GeckoLinux promises to offer the stability and behind-the-scenes architectural benefits of one of the few well-funded open source distros, tailored for the average Linux desktop user, even if that means shipping with proprietary drivers and programs.



The Cinnamon edition is the recommended and most tested edition of GeckoLinux, because it serves as the daily driver for the distro's lone developer.

The distro is available in two editions. There's GeckoLinux Static, which is based on the one-release-a-year openSUSE Leap branch that's stable and suited for use cases that need long-term support. Then there's GeckoLinux Rolling, which is based on openSUSE's rolling release known as Tumbleweed. Here we're looking at GeckoLinux Rolling, which makes periodic releases to enable users to jump into the distro.

One of the major differences between Tumbleweed and GeckoLinux Rolling is that the latter includes packages from the Packman repository as well. Packman is a set of third-party software repositories that gives openSUSE users access to a lot of restricted and encumbered packages that the distro can't package in its official repos. Including packages from Packman makes GeckoLinux Rolling a more usable and functional distro straight out of the box.

Both GeckoLinux releases have several editions each with different desktop environments including Cinnamon, Xfce, Gnome, KDE, Mate, LXQt and Budgie. Unlike the Static branch, the Rolling release also offers a version based on elementaryOS' Pantheon desktop as well. Besides these, there's also a bare-bones edition with a vanilla IceWM desktop, the *Firefox* web browser, the *YaST* configuration tools, to give advanced users the perfect platform to custom-build their installations.

Greenhorn geckos

Besides the inclusion of extra tools, the other notable difference between GeckoLinux and its base distro is the use of the *Calamares* installer. We really like openSUSE's installer, especially the System Roles function that enables users to tailor their installations for particular use cases. However, the installer's dexterity goes against the desktop-oriented focus on GeckoLinux, which is why the distro has gone with the distro-agnostic *Calamares*, which is more tuned to the sensibilities of a desktop user. Starting with this release, the *Calamares* rendition in GeckoLinux Rolling will default to using the Btrfs file system, with transparent Zstd data compression.

The release also ships with zRAM swap enabled along with the EarlyOOM daemon that'll help prevent installers from experiencing responsiveness issues due to the installation running out of memory. Apart for the installer, GeckoLinux includes all of openSUSE's *YaST* configuration tools for all kinds of system tasks including package management.

All editions of the distro are available as Live installable images for 64-bit machines. The developer also highlights the multilingual nature of the distro, and has shared illustrations to enable non-English users to use GeckoLinux in their native languages. There's not much else in terms of documentation. However, documentation about the other components can be accessed from other sources, though first-time users would appreciate a link or two to direct them to these sources. Importantly, if you do run into issues using GeckoLinux, the project has a fairly lively forum board that sees active participation from the developer. LXF

VERDICT

DEVELOPER: sb56637
WEB: https://geckolinux.github.io
LICENCE: Various

FEATURES	**8/10**	**EASE OF USE**	**8/10**
PERFORMANCE	**8/10**	**DOCUMENTATION**	**5/10**

GeckoLinux Rolling 999.21 uses its openSUSE foundation to deliver releases that'll appeal to all kinds of desktop users.

» Rating 7/10

NomadBSD 130R

While **Mayank Sharma** has tried a couple of BSD desktop spins, none of them has offered a compelling reason for him to keep them around… until now.

IN BRIEF

A FreeBSD-based live persistent OS that's designed for USB flash drives. The OS has tuned the vanilla FreeBSD for use by regular desktop users and ships in a ready-to-use state. It's preloaded with tools and thanks to FreeBSD's famed hardware support, is available for both 32- and 64-bit setups.

SPECS

Minimum
CPU: 1.2Ghz
Memory: 1GB
HDD: 8GB USB flash drive
Build: Both 32-bit and 64-bit

NomadBSD 20210508 is a FreeBSD spin with two unique aspects. First is the fact that it's a BSD aimed at average desktop users. Second, NomadBSD is designed to operate from a USB disk, which is by no means a small feat and something that only a handful of Linux distros have successfully mastered.

Unlike popular FreeBSD-based spins such as TrueOS, NomadBSD doesn't make any architectural changes to the base FreeBSD. In essence, everything that NomadBSD does and offers could be replicated on top of a vanilla FreeBSD installation. The advantage of this approach is that utilities from FreeBSD will work in NomadBSD, and users have the perfect platform that they can use to learn the ropes of FreeBSD.

NomadBSD can be installed on to the hard disk as well using its rudimentary single-screen installer that currently takes over the entire disk.

In the latest NomadBSD release, the developers have upgraded the base system to FreeBSD 13 released in April 2021. The other major change is that NomadBSD's partition alignment has been changed to 1M, which the developers say will improve write speeds on flash drives.

NomadBSD relies on a first-boot wizard to configure the language, locale, keyboard and timezone, and also helps set the password for the default nomad and root users. Very helpfully, the wizard also offers the ability to encrypt the data partition on the USB using FreeBSD's geli block-level device encryption utility.

Next up, the wizard prompts users to choose some core default utilities. This is particularly useful for Linux users as it will help introduce a little familiarity in what is otherwise a completely different operating system. For instance, you can choose to use the bash shell on NomadBSD, which will appeal to CLI warriors.

The first boot wizard then uses the selection you've made to prepare the USB. This involves creating a new partition for the **/data** directory, which will take over the remaining space on the storage device. Depending on the size, creating the file system can take several minutes. Once it's done, NomadBSD will reboot into the fully configured persistent environment.

Clean as a whistle

NomadBSD boots to the minimal *SLiM* display manager, which is odd since the project appears to have been abandoned. However, it still works as advertised. On the desktop, NomadBSD sports the lightweight Openbox desktop environment with a Plank panel at the bottom.

The OS borrows various components from the Desktop Suite for FreeBSD (DSB) project, which is a meta project that packages some of the most popular desktop programs, tools and utilities for FreeBSD. DSB is maintained by one of NomadBSD's core developers.

To manage wireless and wired connections, NomadBSD uses the *Netmgr* network manager, which is a Python GTK3 network manager for FreeBSD developed by the GhostBSD project. You can also use the NomadBSD environment to interact with files on your disk as it ships with support for several popular filesystems including FAT, HFS+, NTFS and Ext2/3/4.

The OS ships with several popular open source tools, so Linux users shouldn't have any problems using NomadBSD. There's also the *Linux Browser Installer* tool that enables users to install Widevine-capable Linux versions of the *Chrome*, *Brave* and *Vivaldi* web browsers that can then be used for popular streaming services.

You can also flesh out your NomadBSD instance by installing and upgrading software packages using the *OctoPkg* utility, which is a graphical front-end to FreeBSD's *pkg* package manager. Even more impressively, you can enable FreeBSD's Linux binary compatibility and install and use virtually all Linux tools on top of NomadBSD without any issues. LXF

VERDICT

DEVELOPER: The NomadBSD Team
WEB: https://nomadbsd.org
LICENCE: BSD 2-Clause "Simplified" License

FEATURES	**8/10**	**EASE OF USE**	**8/10**
PERFORMANCE	**8/10**	**DOCUMENTATION**	**7/10**

Not only does NomadBSD work great from removable devices, it's also a wonderful lightweight option for older PCs.

» Rating 8/10

» PLUS: Exclusive access[1] to the *Linux Format* subs area!

1,000s of DRM-free PDF back issues and articles! Get **instant access** back to issue 66 (May 2005) with tutorials, interviews, features and reviews. At **linuxformat.com**

1) Only available to www.magazinesdirect.com subscribers

DON'T MISS! Now with 5 years of *Linux User & Developer* issues

NOT FROM THE UK? Turn to page 67 for more great subscriber deals!

» CHOOSE YOUR PACKAGE!

ANNUAL PRINT EDITION

Only **£65.00**

13 issues of *Linux Format* in print over 12 months by Direct Debit

ANNUAL PRINT + DIGITAL EDITION

Only **£70.00**

13 issues of *Linux Format* over 12 months in print and digital by Direct Debit

DIGITAL EDITION ONLY

(no gift) Only **£50.00**

13 issues of *Linux Format* in digital by Direct Debit

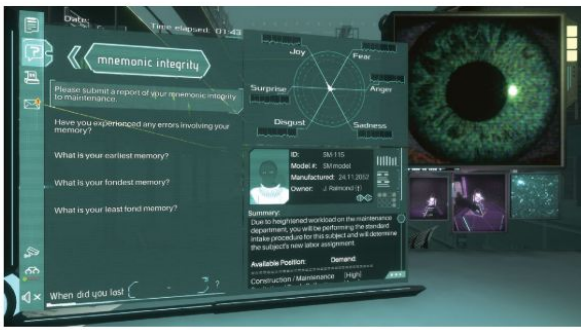
Silicon Dreams

Management loves to treat the staff like robots. A defiant **Luke Kemp** can confirm that they're certainly not human…

SPECS

Minimum
OS: Ubuntu 12.04+
CPU: 2GHz
Memory: 4GB
HDD: 4GB
GPU: Shader Model 4 capable

Although 1982's *Blade Runner* has inspired three trillion and six films, games, books, comics, albums and cereals, only a tiny percentage of them have understood (or even tried to understand) the film. Hurling neon signs and Japanese words around does not constitute telling a cyberpunk story. Here, however, we have a game that not only understands *Blade Runner*, but thoroughly deserves to be mentioned alongside it. Y'know, like we just did.



This is one memory assessment that doesn't involve MemTest86.

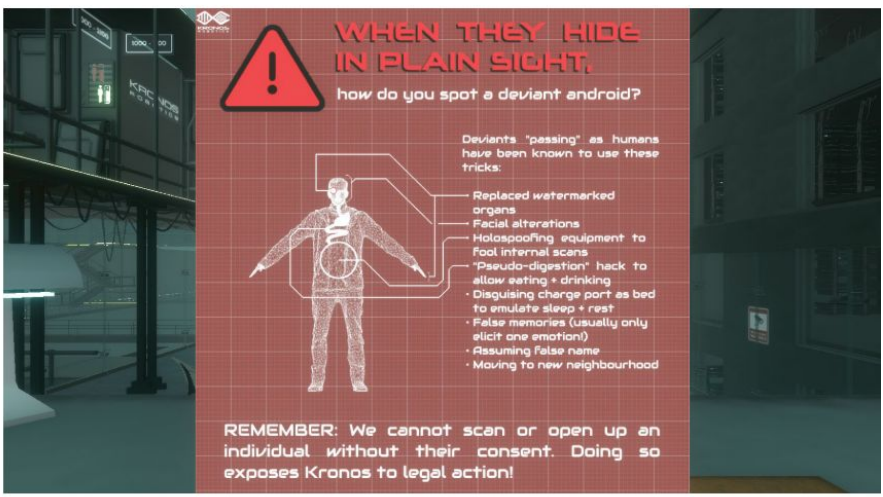
The Voight-Kampff test is how blade runners determine whether an individual is human, or a human-seeming android. *Silicon Dreams* takes the idea of this test, and especially the machine involved in conducting it, and runs with it. It runs hard, it runs fast, and it runs with a hell of a lot of style.

Your character is an android created with the sole purpose of running what is for legal reasons absolutely not a Voight-Kampff machine. *Silicon Dreams* (mostly) consists of a half dozen or so interrogations conducted with this machine. Usually you'll be speaking with androids, but occasionally with a human. While at some point you'll have to determine whether somebody is human or android, the experience for the most part takes sharp turns into territory you'll never see coming.

With no voice acting and relatively simple graphics, there's enormous pressure on the script. Which it lives up to: the writing is incredible. Somewhat ironically, each and every android met seems more human than most of the other characters you would have met in other games.

The depth and texture of the interview subjects is a vital part of the experience. You'll find yourself being pulled in several directions at once. Every interview has an accompanying report to be filled in and, for androids, one of three choices must be made: release, send for maintenance (which guarantees a memory wipe), or destroy. Things are not nearly as simple as they appear, of course.

The company you work for is the manufacturer of these androids, and holds no sympathy for androids that begin to evolve emotionally. Unsurprisingly, the individuals themselves have no desire to be memory wiped or, indeed, to die. Yet as an android yourself, you're expected to toe the company line at all times. It's made clear that



Forewarned is forearmed, but it's no guarantee of success.

Take in a panoramic view of the future (flying cars just out of shot).

any attempt to defy your employers (*heaven forbid! <looks around nervously> – Ed*), or provision of information that contradicts their expectations (intentionally or otherwise), will cast suspicion on you. You're then assessed for your performance after every interview. If the rating falls too low then it's tears in the rain time, as you're destroyed and replaced.

Double Deckard

It's more than possible to fail in this manner before the end, but we found ourselves more than invested enough to do everything possible to avoid that. During one sequence that you'll probably see coming a mile off, but is no less powerful for that, we were constantly on edge until it was made clear that we were allowed to carry on. Whether deciding the fate of somebody else, or worrying about our own, *Silicon Dreams* is full of dread.

Emotions are central to play. The interrogation machine you use displays the emotions that accompany the answers of interviewees, exposing lies and confirming truths. Certain android models are also supposed to have certain emotions limited or disabled, and the company expects you to report any deviations. Ingratiate yourself with the company enough, and you even unlock the ability to slowly induce an emotion of your choice.

Trying not to spoil any of the surprises, there are a few experiences we crossed during the emotional journey *Silicon Dreams* took us on. During one interrogation, the order was to get a truthful answer to an important question, needing to induce fear in the subject. Hating ourselves for doing so, but considering it necessary, we locked down the shackles in the interview chair. The surprised interviewee immediately became distressed and we disengaged the shackles as soon as we had the info needed.

That same interrogation took us dangerously close to the scrap heap, so we sucked up to the company's wishes more than usual to save our shiny metal ass. That included wiping the android's memory, a decision that fills us regret even as we type this. As a final kick in the moral teeth, even then the end result was only just a narrow positive assessment.

Conversations twist and turn in unexpected ways. We decided to trash, save and then at the end trash one android, but ultimately felt terrible about it. In another case we sympathised with the subject, to the point where we decided a memory wipe was the best option, only to be gutted when the company destroyed it anyway. Finally, the story that will stay with us the longest was bittersweet. Through our actions, this android retained its personality but had its life expectancy suddenly blunted. The reason, which was revealed later, made us smile.

While typically subtle and smart, with allusions to working-class life, *Silicon Dreams* could've done without the cringe-worthy references to *Blade Runner* and *Do Androids Dream of Electric Sheep*. It also gave us mixed feelings about the first ending we reached, which didn't lead on smoothly from our final actions. Nonetheless, it's an experience that will stay with us, and one that we'll be going back to. LXF

What are you looking at? Making small talk in the distant 2065.

VERDICT

DEVELOPER: Clockwork Bird
WEB: www.clockworkbird.net
PRICE: £12.49

GAMEPLAY	**9/10**	**LONGEVITY**	**7/10**
GRAPHICS	**9/10**	**VALUE**	**9/10**

An extremely intelligent sci-fi interrogation simulation that's unpredictable for all the right reasons.

» Rating 9/10

WE COMPARE TONS OF STUFF SO YOU DON'T HAVE TO!

Roundup

» Aqemu 0.92 » Gnome Boxes 40.1
» VirtualBox 6.1.22 » Virt-manager 3.2
» VMware Workstation Player 16.1.2

Neil Bothwick
A long-time lover of virtualisation, Neil's speciality is doing virtually nothing.

Desktop virtualisation

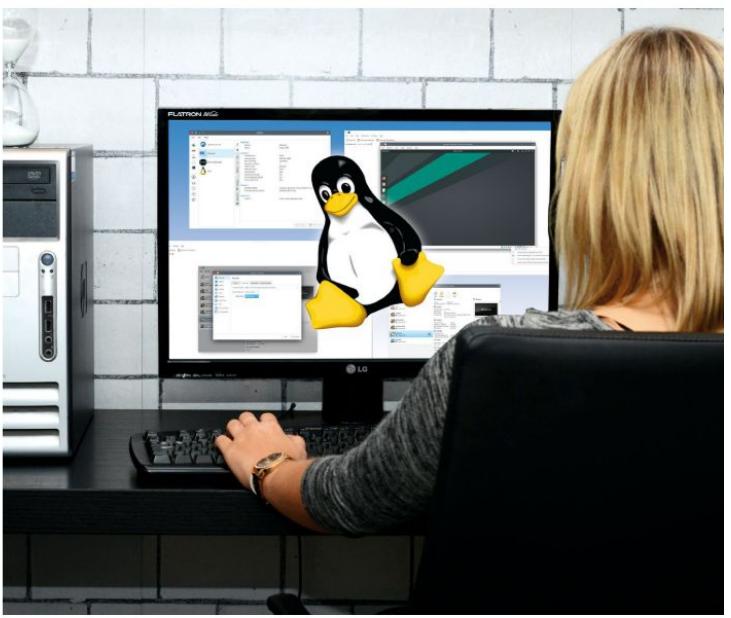
Need to run a different distro without rebooting, or run Windows without your friends knowing? **Neil Bothwick** looks at some virtual options for you.

HOW WE TESTED...

Each virtual machine you run will take a certain amount of memory, whether it uses it or not, and can be allocated a number of CPU cores. How much of each you allocate affects not only the performance of the VM, but also of any other VMs you may be running – together with the host OS if you take too much. We tested on a system with an eight-core processor and 32GB of RAM, and limited each VM to two cores and 4GB.

VMs can either use a dedicated hard disk partition or they can use a file as a virtual disk. We chose to take the latter approach when testing, because that's the most common method. As such, we didn't run any I/O speed tests for the VM because that would be determined more by the filesystem in the host.

Similarly, CPU performance was very close to native in most cases because the kernel's KVM subsystem gives the software almost direct access to the host's CPU cores. That means we concentrated on how well they did the job, as how fast was a given.



Virtualisation is a big thing these days. It's used by big data companies to spin up extra resources when needed, because it's much quicker to boot a VM on a running OS than to boot a hardware box from scratch. However, we're looking at the other side of virtualisation here: running virtual machines on your desktop. This may be because you need to use a different distro from time to time – either to see if the grass on the other ISO really is greener or because you're testing software on multiple distros.

It could be that you have to use a certain piece of Windows software that won't run in *WINE*, but you can't or don't want to reboot. It could be that you need to keep some software isolated from the rest of your system for security reasons (although containers are sometimes sufficient, and more efficient, for this). You might need to use old software that won't run on your modern distro. We could go on, but you get the point. So let's look at some of the alternatives in virtualisation and see which may be best suited for your needs.

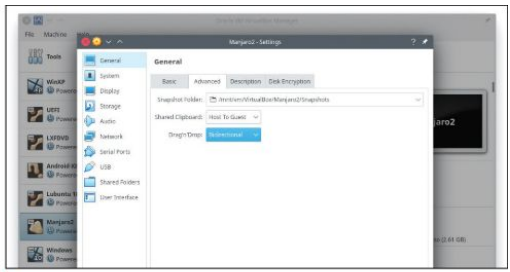
Talking to your guests

Communicating across the divide.

You might be interested in using a virtual machine for development, in which case you may want it to be isolated from the host. Alternatively, you could well need to transfer files or information between host and guest. With *VirtualBox* you first need to install the *Guest Additions* inside the VM, which in turn means you need a compiler suite installed on the VM and a kernel built with the necessary options, so this may be a simple task or a more complex one. After that it's simply a matter of enabling the shared clipboard in the settings to be able to copy on one side and paste on the other. Similarly, you can enable drag and drop to copy whole files.

The situation with *VMware Workstation Player* is similar. Install the *VMware Tools* option and, if it works, you get the shared clipboard and files. However, when we tried it on a test system running Linux Mint, it failed with an undiagnosed error.

With *Virt-manager*, the situation is much simpler. Install **spice-vdagent** from the guest distro's repositories (or **spice-guest-tools.exe** for a Windows guest) and clipboard sharing just works. You may even find, as we did, that the guest distro you're using already had **spice-vdagent** in place – it couldn't be simpler. It was also possible to copy files to the guest just by dropping them onto the VM window. *Aqemu* uses the same *Qemu* backend as *Virt-manager*, but the Spice configuration meant there was no display when we tried to use it, so clipboard sharing didn't work. However, *Gnome Boxes* just worked, like with *Virt-manager* – sharing clipboards and files couldn't be simpler.



Once you've installed the guest utilities package VirtualBox can share files and the clipboard between the host and a guest.

While you can copy files this way, sometimes it's easier to share a folder on the guest. *Virt-manager*, *VirtualBox* and *Aqemu* all make it possible for you to select a folder that's shared over Samba (you'll need the Samba server running on the host, though). *Gnome Boxes* and *VMware Workstation Player* don't have this option, although you could run Samba on the guest if you really needed it.

VirtualBox, *Aqemu* and *Virt-manager* also enable you to run headless, making a display available over VNC if needed. This is particularly useful if you want to transfer a VM to a headless server once you have it set up, but still administer it from a GUI.

VERDICT

AQEMU	5/10	VIRTUALBOX	8/10
GNOME BOXES	8/10	VMWARE WORKSTATION PLAYER	6/10
VIRT-MANAGER	9/10		

Once set up, Virt-manager, VirtualBox and Boxes make transferring information a breeze.

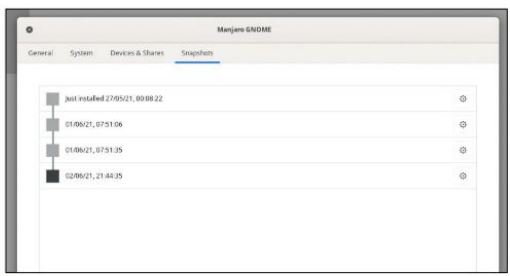
Happy snaps

Keeping your guests safe.

Even though a virtual machine isn't a real computer with physical value, the data that's on it can still be real and have some value. So it can be as important to be able to back up your virtual computers as the host. You could run a backup program on the guest, but there's another option: make use of snapshots.

A snapshot is a view of your VM at a moment in time, which means you can roll back to that point when needed – for example, after an upgrade that breaks something. Importantly, snapshots can be taken while the VM is running. Taking a snapshot can be near instant, making it an easy way to keep safe. So how do the various programs handle this?

Gnome Boxes has a snapshots tab where you can create a new one or revert to a previous snapshot. It conveniently creates an initial snapshot for a newly installed VM, something none of the other do, but it makes it easy to get back to a freshly installed state. *Aqemu* is similarly simple to work with. Snapshots are managed from a control window, although it only allows snapshots of running machines. *Virt-manager* enables you to manage snapshots of running and stopped guests, and even include screenshots of snapshots. *VirtualBox* also allows snapshots of running and stopped guests. It also makes it possible for you to clone a VM from a snapshot. *VMware Workstation Player* has no snapshot facility. What it does have, in common with the others, is the ability to suspend the guest OS, which would enable a quick restart or even the option to copy the VM to another computer.



Gnome Boxes makes taking snapshots easy, and rolling back to them if you need to, as do all the open source alternatives.

VERDICT

AQEMU	7/10	VIRTUALBOX	9/10
GNOME BOXES	8/10	VMWARE WORKSTATION PLAYER	2/10
VIRT-MANAGER	9/10		

Instant snapshots at the click of a button can help recover from those dreaded "oops" moments.

Ease of setup

Installing the software and a guest OS.

Creating virtual machines is often the most complex task you do with these programs, so how easy and flexible are they in this task? Any program will enable you to create a virtual disk and then boot from an ISO image as if it were an optical disc, from which you then run the standard installation process.

Beyond that, some of these make the process simpler, or give you more choices. How easy are they to use when creating a new virtual machine and how flexible are they? Before that, how easy are they to install in the first place?

Most of the candidates in this month's *Roundup* can be installed from your distro's package manager, but some need extra steps to be performed before they're ready to use. If you're a serious user of virtualisation then a little preparatory work is fine, but not so much if you just want to spin up a live DVD to see how a particular distro looks – or fire up Windows to run the occasional program.

Aqemu 9/10

Aqemu is the simplest of the programs here, being simply a front end to the *Qemu* command line software, but it makes setting up a VM pretty easy. The creation wizard can either generate a typical virtual machine, requiring little input from the user, or it has a custom mode that enables you to choose most settings.

Unlike most of the other programs on test, it also makes it possible to choose the location of your virtual disk file, which is useful if you need to store the disk files somewhere other than your home directory – either for space or organisational reasons. It gives a choice of emulated hardware for some devices, such as sound and network cards, which is useful if the OS you're trying to install only supports certain models. The networking options also enable you to share folders over Samba, providing the server is running in the host.

Gnome Boxes 7/10

When you first start *Gnome Boxes* it shows a list of ISO images within your home directory hierarchy and enables you to select one to boot. If you want a different distro, you can select it from the predefined list; *Gnome Boxes* will then download and boot the ISO. Either way, you then proceed with the usual installation process for that distro.

If the ISO you choose is one that *Gnome Boxes* is configured to handle, which includes Ubuntu 20.10, Debian 10 and Fedora 33, but not Windows, select one of those and *Gnome Boxes* will do some of the work for you, but you still need to go through the usual questions about locale, user name and so on. It makes recommendations for memory and virtual disk size, but you can change those, either now or later. It does create a VM using all available CPU cores, which may not be ideal and can affect performance.

Documentation

You know you'll only look for it when you're stuck.

Virtualisation is a complex process, so you would expect extensive documentation to come with each of these programs. However, that's not always the case. The documentation for *Virt-manager* consists of a man page listing the startup options and a four-question FAQ on the website. About the only real help you get are bubbles that appear over various GUI options.

VirtualBox goes the other way, with a 400-page PDF manual, which seems scarily huge for first-time users, but does have an easy primer on creating your first virtual machine. This really is commercial-level documentation, and not a man page in sight!

Aqemu takes a similar approach to *Virt-manager*, but without the helpful bubbles. This is a previously abandoned project that has been resurrected, but it would be nice if someone could write some documentation for it. *Gnome Boxes* is similarly lightweight on helpful information – there seems to be something of a trend here. These programs are supposed to provide an easy way into virtualisation, and they do, but the time will come when you want to do more than boot from an ISO image and install it on a virtual disk. Yes, the man page for *qemu*, which they use behind the scenes, covers all the options, but the point of an easy graphical interface is to avoid the need to delve into man pages. Open source software is often criticised for its documentation, and these programs prove the point.

VMware Workstation Player is a commercial product and it shows in the documentation available from its web site, which puts all but *VirtualBox* to shame. Good though it is, it's neither as comprehensive nor as easy to navigate as *VirtualBox's* manual.

VERDICT

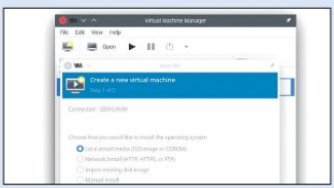
AQEMU	2/10	VIRTUALBOX	10/10
GNOME BOXES	2/10	VMWARE WORKSTATION PLAYER	8/10
VIRT-MANAGER	3/10		

All or nothing documentation options, VirtualBox is the only one that's worth mentioning.

Virt-manager 8/10

Virt-manager is a graphical control panel for *Qemu* virtual machines, usually under the control of *libvirt*. This means you can create a VM in *Virt-manager* using the GUI options, then run it from the command line later. It also means you can import existing *Qemu* VMs directly into *Virt-manager*.

One annoyance with *Virt-manager* is that it asks for your root or sudo password each time you start it. To avoid this, you need to add yourself to the libvirt group, and possibly add a polkit rule if your distro's package hasn't already done that, and log in again. You'll still need root permission to create a new virtual disk for your VM and *libvirt* keeps these in **/var/lib/libvirt**, which is owned by root. This seems unnecessary for simple desktop virtualisation, although it does fit in with *libvirt's* wider remit. Apart from the root hassle, *Virt-manager* makes setting up a new VM very easy.



VirtualBox 9/10

Creating virtual machines in *VirtualBox* is a relatively straightforward process. Only a minimal number of questions are asked by the program during creation, then it's up to you to use the Settings window to make changes as you see fit. You'll need to use this to add an optical disc (usually using an ISO image) to boot an OS installer – otherwise most of the settings have sensible defaults.

One handy feature is a warning displayed at the bottom of the settings window if *VirtualBox* detects settings that it thinks are inappropriate. It doesn't stop you shooting yourself in the foot with a poor setting, but it does let you know where the gun is pointing – you're still free to break things.

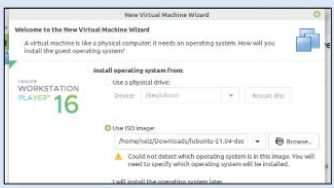
Virtual disk images are stored in **~/.VirtualBox** by default, but you can put them elsewhere if you want. You can also set an alternative location for snapshots.

VMware Workstation 4/10

VMware Workstation Player is a commercial product that's free for non-commercial use. It's supplied as a binary bundle that you run to install.

A drawback of *VMware Workstation Player's* approach – which is the only way to do it for closed source software – is that it'll only run if you have compatible libraries and so on, which not all distros do. It also needs a compiler present because it has to build a couple of kernel modules the first time you run it. It's really a matter of trying it on your distro and hoping that it works, because there's no useful output to analyse when it fails, even if run from the command line.

Providing it works, creating a new virtual machine is straightforward: just answer the questions, point it at an ISO image to install from and off you go. Of course, one of the reasons for this simplicity is the lack of choices.



Compatibility

Will it fit in with your computer system and beliefs?

Not all of these VM options are completely open source and that can affect compatibility with your system, both in a philosophical way and at a more pragmatic level. *Aqemu*, *Virt-manager* and *Gnome Boxes* are all based on *Qemu* and are completely open source. Their hardware interface is through modules included in the standard Linux kernel and should work on just about all distros. The only time you may run into problems is if the KVM hardware acceleration isn't enabled on your computer, which is turned on and off in the BIOS settings menu.

VMware Workstation Player is completely closed source, meaning it can only work with the distros it supports. It also requires kernel modules to be compiled when it's first run, and after any kernel updates, which means that you need a compiler toolchain installed on the host.

VirtualBox is somewhere in between. The core is open source but some parts, such as USB 2.0 and 3.0 support, Remote Desktop Protocol and disk encryption are closed source. This also means that *VirtualBox* needs external kernel modules to be built, but this is often taken care of by distro package managers. So this should only be an issue if you install direct from the *VirtualBox* web site. Both of these are under a licence that permits non-commercial use for free.

Performance of the various programs should be identical because all use the KVM extensions in modern CPUs to give performance close to native running, although *Gnome Boxes* did seem sluggish at times. *VirtualBox* did experience some slowdowns while accessing the disk, but it was annoying more than harmful.

VERDICT

AQEMU	**10/10**	**VIRTUALBOX**	**7/10**
GNOME BOXES	**10/10**	**VMWARE WORKSTATION PLAYER**	**5/10**
VIRT-MANAGER	**10/10**		

VirtualBox is compatible with your hardware, but do you want to run closed source software?

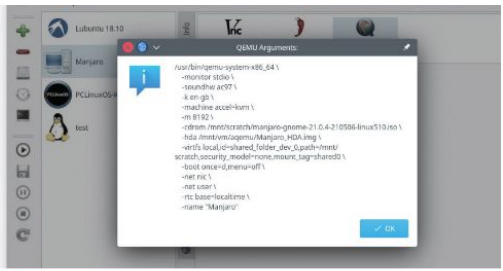
Useful extras

Additional titbits to tempt you…

Some of the programs have their own unique (or fairly so) features that may make them particularly suitable for your own computing needs. For example, all of these programs provide an x86 CPU (either 32- or 64 bit) when run on an x86 host. That's why they're almost as fast as running the software natively, but *Virt-manager* can also emulate ARM processors. This is much slower, of course, because it's carrying out full CPU emulation. On a powerful-enough host, it means you can develop or test software for other platforms, such as the Raspberry Pi and other SBCs.

Gnome Boxes, *Aqemu* and *Virt-manager* are all frontends for the command line *qemu* program. *Aqemu* has a button to show you the exact command and arguments used to run a guest. This means you can use *Aqemu's* GUI to set up the VM as you want it, then copy and paste the command string into a script to run it without the overhead of a GUI, or even from a startup script.

VirtualBox is able to create encrypted disk images (this requires the closed source extensions) to keep your disk images private. Not only can it apply encryption to new images, it can convert existing ones, too. It's closed source encryption, so you have to trust that there are no weaknesses, but it uses open source ciphers so you can be fairly confident. However, it's more convenient than separately encrypting disk images, or encrypting them within the guest OS. It also makes a lot of sense if you're transporting VMs on a removable drive.



Aqemu doing itself out of a job, by showing how you can run a virtual machine without the Aqemu GUI.

Virt-manager is able to manage other types of virtual machine, not just the *Qemu*/KVM ones that we've covered here. It can also work with Xen machines and Lxc containers, which gives it an advantage if you use different types of virtualisation.

VERDICT

AQEMU	**8/10**	**VIRTUALBOX**	**9/10**
GNOME BOXES	**1/10**	**VMWARE WORKSTATION PLAYER**	**1/10**
VIRT-MANAGER	**9/10**		

Encrypting your virtual disk may be a good idea if it's sitting on a removable drive that can be easily misplaced.

Host interactions

How the guest uses the hardware.

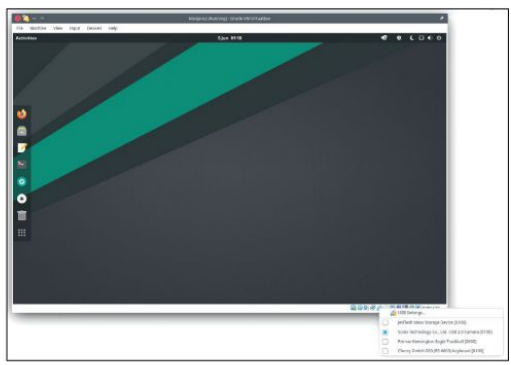
While all of the software here can use a physical disk partition for its storage, it's more common to use a file as a virtual disk. Not only does this save having to manage lots of partitions, it also makes creating and moving virtual machines much easier.

Each of these programs will create such a file when setting up the virtual machine, but they each use their own format. However, some of them are able to work with different formats. *VMware Workstation Player* uses its own format and that's pretty much it for the free version. *Gnome Boxes*, *Aqemu* and *Virt-manager* all use *qemu* behind the scenes and so are able to work with *Qemu's* qcow2 format, *VirtualBox's* vdi files and *VMware's* vmdk files, although some features are only available when working with native qcow2 images.

In a similar fashion, *VirtualBox* is able to use qcow2 and vmdk files as well as its native vdi. All of the *qemu*-based programs as well as *VirtualBox* are able to convert disk images between various formats, including the more or less universal raw format, by using their command line tools.

VMware Player works well with peripheral devices. Not only the obvious ones such as USB storage devices, but it also identified a laptop webcam when booting a guest and offered to give the guest access to it. *VirtualBox* give access to USB devices through an icon displayed in the window border. This worked with both a flash drive and a webcam.

Adding external devices in *Virt-manager* was also possible, if not as streamlined a process. *Aqemu* was able to add a storage device to the guest OS, but not share USB devices in general. *Gnome Boxes* has a devices tab in a guest's properties window, but no option to add one on the fly. It's a little disconcerting that *Gnome Boxes* has this look at times: space for a feature but nothing there yet. Let's hope the gaps are filled soon because the features are available in the underlying *qemu*.



VirtualBox makes accessing the host's USB peripherals from the guest as simple as clicking on button menu in the window border.

VERDICT

AQEMU	**7/10**	**VIRTUALBOX**	**9/10**
GNOME BOXES	**5/10**	**VMWARE WORKSTATION PLAYER**	**6/10**
VIRT-MANAGER	**8/10**		

Being able to transfer VMs between the different programs is a key feature.

The Verdict

Desktop virtualisation

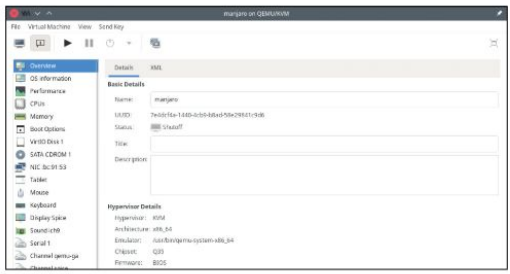
As always, the best choice for you is the one that fits your needs, because in this month's *Roundup* there is no one clear winner for everyone. Based on general usage we have picked one that came top, but it's close – very close. For its range of features, ease of use and adaptability, we choose *Virt-manager*. It does just about everything the others do, with the obvious exception of *VirtualBox's* disk encryption, and does it in a way that is generally easy to get on with.

Very close behind is *VirtualBox*. It has plenty of features yet remains reasonably easy to use, and it comes with superb documentation – a manual bigger than some O'Reilly books. The downside is that it comes with closed source baggage. If you're not concerned about that, it may be a better choice.

Aqemu is a much simpler offering, essentially because it's not trying to be the ultimate virtualisation tool. For some uses that's a good thing, the interface isn't cluttered with options that you neither need nor understand. Despite its recent development hiccups – the project ceased development and was then started under new leadership – *Aqemu* is a good choice for those that just want to run a Linux distro, or Windows, in a virtual machine without any hassle.

Gnome Boxes is also aimed at the simple/easy way of doing this. While this works to a point, that point is when you want to do more than the basics. This is because the program has gone too far in its simplicity and is missing some basic features that should be there. However, it's relatively early days for this project and it's worth keeping an eye on for future developments.

Where does that leave *VMware Workstation Player*? Nowhere, really. It's restricted, both in features and freedom. This author used to have an expensive *VMware* licence years ago when it was the only good option, but the *Qemu*-based and *VirtualBox* options have made huge strides since then. There seems little reason to use a restricted proprietary solution with limited features that may not even run properly on your distro when the current libre options are so good, they do everything that's required of them nowadays. Unless you really need a solution compatible with *VMware's* commercial products, there's little to recommend it, although what it does it generally does well.



1st Virt-manager 9/10

Web: http://virt-manager.org
Licence: GPL-2 Version: 3.2.0
Powerful and flexible, maybe more than some people need.

2nd VirtualBox 8/10

Web: www.virtualbox.org
Licence: GPL2 and PUEL Version: 6.1.22
Full featured and well-documented, but not entirely free.

3rd Aqemu 7/10

Web: https://sourceforge.net/projects/aqemu
Licence: GPL-2 Version: 0.92
If you want the simple, open source option then this is it.

4th Gnome Boxes 6/10

Web: https://wiki.gnome.org/Apps/Boxes
Licence: LGPL2+ Version: 40.1
Still very basic, but easy to use as far as it goes.

5th VMware Workstation Player 3/10

Web: www.vmware.com/uk/products/workstation-player.html
Licence: Proprietary Version: 16.1.2
Capable, but has all the limitation of a restricted binary program.

» ALSO CONSIDER

We have mentioned *Qemu* a few times here, it's the engine that powers three of these programs. *Qemu* is command line only, but may be more suitable for some needs. The learning curve is steeper but you do get access to all of its features, not just those that the GUI exposes. *Libvirt* is another command line option, which is used for managing *Qemu* VMs and the layer behind *Virt-manager*. It may be that this program could suit your requirements if a GUI isn't necessary.

It's also worth considering whether you actually need a fully virtualised computer in the first place. If you're trying to run software away from your existing OS – perhaps for security reasons – or want different versions of software, then you may find that some type of containerisation, such as Docker or Lxc, is a better option for you. Of course, if you want to boot your *Linux Format* DVD to try out the latest distros then this is not a viable option.

FASTER, BETTER SERVERS!

As the dust settles following Red Hat's winding down of CentOS, **Jonni Bidwell** looks for alternative server distributions.

The winter of 2020 will be remembered by many as 'not a particularly festive time', but spare a thought for CentOS (the community-powered distro based on Red Hat Enterprise Linux), who were told unequivocally in December that further development would be axed.

Contrast this with the headlines of rose-tinted 2014, when CentOS announced that it would be partnering with Red Hat, giving the company access to vast resources while still enabling it to remain an independent, community-driven project. Red Hat recognised the value in CentOS, which at the time of the union was the world's most popular server distribution (Debian later reclaimed that prestigious title).

A lot can happen in a few years (Red Hat being bought by IBM, for example). Issues of governance, development and other foibles led to the partnership's demise. Red Hat wanted its own CentOS Stream offering (and we'll cover the intricacies behind this later) to be the go-to community enterprise distribution, and for it to function like Fedora used to, as a proving ground for new RHEL features. The Fedora umbrella now encompasses much more than servers, being a leader for Gnome and Wayland on the desktop, and taking on the skies with Fedora Cloud. CentOS Stream would then be a true, free upstream for RHEL, and as such would make the project more accessible to developers.

There was only one problem: the brigades of CentOS users who quite liked their OS and didn't want to migrate to something which sounded very much like an experimental version of RHEL. But all is not lost. The lead CentOS developer is now heading up the Rocky Linux project, which aims to continue the CentOS legacy, and for users who want to give Stream a chance, upgrading from CentOS 8 is easy. Join us as we peer into the nature of RHEL, study the events leading to CentOS's demise and have a look at today's top server distros for a better, faster server...

CentOS remembered

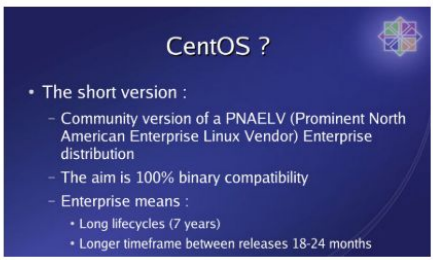
Delve into the illustrious history behind one of the most respected server-targeting Linux distributions.

By the mid-90s, Red Hat had irrefutably demonstrated that it was possible to make money from Linux. Together with SUSE it showed how the burgeoning free software ecosystem, combined with premium support, was a boon for enterprises, small businesses and anyone else that thought that Microsoft site licenses were too pricey.

That legacy continues, and today whenever you think about 'corporate' Linux distros, chances are Red Hat Enterprise Linux (RHEL) will spring to mind. The RHEL title was introduced in 2000, but it was just a new name for the Red Hat Linux it had been supporting since its inception. Alongside this name change came the inauguration of Fedora Core: an experimental, community-supported distro that would function as the upstream for RHEL. In other words, new features would stabilise in Fedora Core and then be adopted by RHEL.

A number of RHEL clones sprang up in the early 2000s, among them Tao Linux and CAOS (not to be confused with KaOS Linux, the Arch-Based desktop distro). When the former ceased development, the project announced that rather than leaving users unsupported it would roll into CentOS (the Community Enterprise OS) that had grown out of the former. CentOS. This saw its first release in 2004 and was more than just a clone – it was entirely binary-compatible with RHEL. It was appealing for anyone who wanted to tinker with RHEL, and useful for developers aiming to deploy to commercial RHEL environments.

The RHEL source code is GPL-licensed, so CentOS was well within its rights to copy and paste code from the Red Hat project. And initially Red Hat had little objections. CentOS was probably driving at least a few RHEL support contracts. Things soured when CentOS founder Lance Davis left the project in 2008, taking with him the CentOS domains and access to infrastructure.



A slide from Tim Verhoven's 2008 FOSDEM talk. The long life-cycle part did not age well...

Control was eventually relinquished the next year, but some in the Red Hat camp felt that this mishap had damaged its brand. Later, counsel for Red Hat contacted CentOS asking that it remove the Red Hat logo and trademarks from its website, but noting it had no objection to the existence CentOS (or other RHEL-rebuilds). If you use the Wayback Machine to view the front page of the CentOS website as it was in September 2013 you can see a tongue-in-cheek response that (avoiding infringing trademarks) makes reference to a "Prominent North American Enterprise Linux vendor".

Nonetheless, relations were soon cordial again, and in 2014 the partnership between Red Hat and CentOS was signed. This fitted with Red Hat's strategy of establishing a 'community beyond the server', and senior CentOS developers probably enjoyed getting paid to work full-time on their baby. The CentOS Governing Board was established at Red Hat and everything seemed copacetic. There may have been whispers when Red Hat introduced CentOS Stream in 2019 (*see below*), but things got very loud in December 2020 when CentOS 8 was canned. With that noise now calmed, join us over the page as we examine the wherefore and why of CentOS Stream.

A TALE OF TWO CENTOSES

CentOS 7 was released in 2014 and if you're still using it you're safe. Red Hat will honour the original support period and so you'll receive maintenance and security updates until 2024.

CentOS 8 was launched in 2019, with the same stated 10-year support period. So imagine the furore when users of CentOS 8 learned that their OS would instead be sunsetted not in 2029, but at the end of 2021, in order that Red Hat focus on its own CentOS Stream offering.

CentOS Stream was introduced in 2019 with the goal of being a truly open source upstream for RHEL. This would bring development of RHEL, which hitherto had taken place behind ominously closed doors, out into the open. When the distinctively unfestive (December 2020) announcement was made that this would be the only CentOS, people were livid: "Fedora is already an upstream for RHEL!", "I want my CentOS downstream from RHEL, not some sort of beta". Worse, Red Hat even corroborated this argument, describing CentOS Stream as "not a replacement for CentOS", but a "rolling preview of what's next in RHEL". That doesn't sound like something you'd want to use in production. And to add insult to injury CentOS Stream support period is only papered until 2024. Be that as it may, there's a simple upgrade path from CentOS 8, and three years of support is much better than one. There are other options, too: read on to check them out.

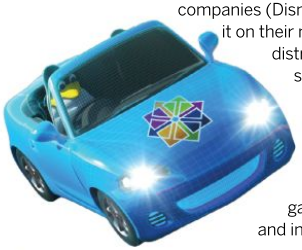
Behold CentOS Stream

Red Hat's new vision is CentOS Stream and caused a family feud, but would a CentOS by any other name smell so sweet?

A large part of CentOS development involved removing proprietary branding and trademarks. This, alongside managing repositories, documentation and package-building infrastructure, is painstaking work, but not comparable to the development effort behind a truly independent distro. Such as RHEL, for example.

Red Hat is generous in the sense that source RPMs for all RHEL packages, including media creation and installation tools, are all made available. So any budding hobbyist could eventually get their own RHEL clone up and running pretty easily. So long as you remove all references to Red Hat, and all pictures of red hats, then you're free to distribute this (providing, per the GPL, you make your changes available too). It's what CentOS did, and it's what Scientific Linux did. And it was good.

CentOS users were plentiful, and they weren't just hobbyists motivated by impecuniousness. Big companies (Disney, GoDaddy, Rackspace) deployed it on their networks, and it's often one of two distros (the other being RHEL) officially supported by professional software, particular within the 3D rendering rubrick where GPU vendors work closely with software bods. It's big in particle physics, too.

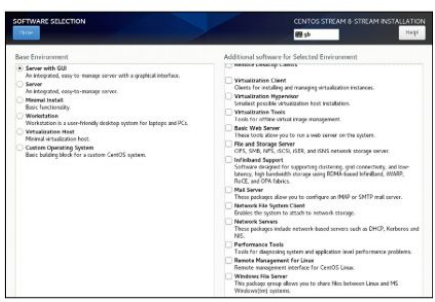


Competition is fierce in the tech game, and companies need to adapt and innovate it they're to thrive. Consider Google's development model for Android. Start with the Android Open Source Project (AOSP) that anyone can use (and indeed do via, say, the mostly open source Replicant and the mostly proprietary *One UI* by Samsung). Then add some fancy tools and services (forget about the data-mongering aspects for a second, lest the analogy will fail) and bundle them into a product which can be licenced to customers (in this case rich device manufacturers). This is a good model, because it drives interest and ultimately revenue. It's similar to how features are stabilised in Fedora before being adopted by RHEL, which is also a good model, but for different reasons (namely, not angering paying customers by breaking things).

Building up the buzz

The tech landscape has a whole bunch of new buzzwords now – DevOps, containers, things-as-a-service, continuous integration, GitOps – and this evokes new ways of working, So if you're a business resting your laurels on the buzzwords of old (servers, firewalled development and homogenous architectures), then you're doomed to fail. We provocatively mentioned IBM earlier, but IBM has always said it wasn't going to meddle in Red Hat affairs and it's almost certain the shift to CentOS Stream was underway before Big Blue bought it. There's plenty of conspiratorial posts to the contrary, but ultimately it's a business decision and, like it or not, from Red Hat's point of view it makes sense.

Red Hat's CTO Chris Wright laid it out pretty straight in an interview (**www.itprotoday.com/linux/community-concerns-prompt-red-hat-drop-centos-centos-stream**): "I would say the big one for us was that CentOS itself was not actually providing that much usefulness to Red Hat. Most of the communities we set up, Fedora for example, do have a lot of bidirectional community involvement. Unfortunately, CentOS was never like that.

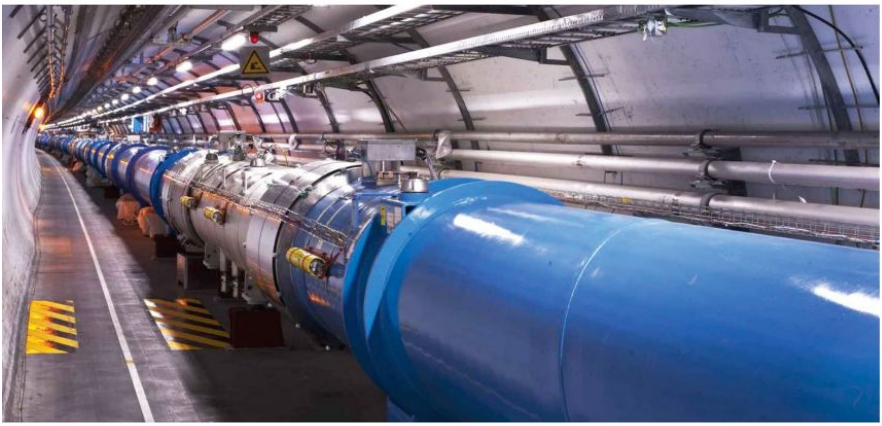


CentOS, and its successor, can set up common roles and software straight out of the box.

» TAKE OFF EVERY SIG

CentOS Stream has its fans (besides Red Hat), though. Facebook is already running the distribution on millions of servers. Facebook and Twitter have in fact teamed up to form a Special Interest Group (dubbed Hyperscale) for large organisations wanting to deploy Stream on their giant infrastructure. Special Interest Groups (SIGs) were introduced in 2014 when Red Hat and CentOS partnered. Red Hat want to nurture these, especially since this is where the idea of CentOS Stream really germinated.

To expand on this slightly there wasn't much point having a virtualisation SIG if users didn't have access to the latest virtualisation stacks, rather than whatever was currently in RHEL. They needed to target what was going to be in RHEL and so SIGs were, according to their purpose, given access to the once-firewalled development sources. In this way you could view CentOS Stream as some sort of omni-SIG, which again is probably a better name than CentOS classic. The introduction of CentOS Stream means SIGs have only one release to target, rather than having to test against both CentOS and RHEL. There are SIGs for artwork (CentOS Stream still needs beautiful, trademark-free branding), Cloud, Storage and more. See the entire list on the wiki page at **https://wiki.centos.org/SpecialInterestGroup**.



CERN, and the LHC (pictured) make extensive use of Linux, see https://linux.web.cern.ch.

CREDIT: Maximilien Brice (CERN), CC BY-SA 3.0, http://cds.cern.ch/record/1211045

It was always a community of users, so that contribution model was mostly one way". Ouch.

The problem was that all these big-name CentOS consumers, apart from not contributing to Red Hat coffers, were unable to contribute code to RHEL (even if they wanted to,). Naturally, CentOS flourished from this userbase, but due to RHEL's development model and tracking ahead of CentOS, Red Hat saw little benefit from the partnership.

Looking back at said partnership (which stipulated that all CentOS trademarks were now Red Hat's), one can easily dredge up references to Microsoft's "Embrace, Extend, Extinguish" strategies of yore. These are, for the most part, wholly inappropriate though. As Red Hat-CentOS liason person Brian Exelbierd succinctly puts it (in an interview with *The Register*): "The CentOS board doesn't get to decide what Red Hat engineering teams do". If CentOS hadn't partnered with Red Hat, then maybe they could have kept on keeping on. But one can imagine a couple of less-than-ideal scenarios too: Red Hat making its life harder; or an independent CentOS struggling to find relevance in the face of Red Hat's own, repositioned open source offerings, for example.

Earlier we mentioned Scientific Linux, the RHEL-derived distro developed by subatomic collision specialists CERN, Fermilab, DESY and ETH Zurich. Scientific Linux was used on thousands of machines at Fermilab and CERN, with development being sponsored by the former. In 2019 Fermilab announced that, in order to have a uniform scientific computing environment across partner labs, it would be switching to CentOS and focusing its development there. So there would be no Scientific Linux 8. Perhaps with the benefit of hindsight there are some pangs of regret at this decision, because in six months there'll be no CentOS 8, either.

As of January 2021 CERN had some 40,000 machines running CentOS 7 and a further 4,000 running CentOS 8 (see the slides at **http://bit.ly/lxf278-cern-pdfs**). In May, alongside the decision to switch the CentOS 8 machinery to CentOS Stream, it announced, "We are evaluating a number of scenarios for future Linux distributions such as community editions or academic licence options over the next 12 months as the shorter Stream lifecycle is not compatible with a number of use cases for the scientific program of the worldwide particle physics community."

It's important to note that RHEL 8 introduces a no-cost subscription that can be used on up to 16 (physical or virtual) machines. And paying customers can use this to add unlimited developers to an existing subscription, so that bringing in a new team doesn't involve further expenses. Essentially, new devs gain access to prestigious RHEL on their personal machines and clusters, which sort of obviates one fairly sizeable aspect of CentOS.

The Red Hat Developer Subscription for Individuals, to give it its pedigree name, is well worth checking out, and for SMEs not tempted by CentOS Stream it might well be the best option. Red Hat won't mind if companies choose to invest in a commercial subscription, but its literature assures us that individual subscribers won't be contacted by salespeople.

RED HAT'S CTO TELLS IT LIKE IT IS

"I would say the big one for us was that CentOS itself was not actually providing that much usefulness to Red Hat."

Any large company dabbling in new technologies would surely be interested in a distro combining the innovation of Fedora with the stability of (and binary-compatibility with) RHEL. And individuals will like something that tracks ahead of RHEL, so that they're not overly inhibited by its slow release cycle.

Under the classic CentOS model, it wasn't possible for a minor bug to be patched until the corresponding bug was fixed in RHEL (unless the bug was introduced by CentOS itself, which is unlikely given it stays as true to RHEL upstream as possible). So CentOS users had to hope the bug was picked up by RHEL gatekeepers in time for the next release. You can read a real-life account of this happening in the *Nmap* package, and why this wouldn't happen with CentOS Stream, at **https://medium.com/swlh/centos-stream-why-its-awesome-5c45d944fb22**.

Rocky Linux

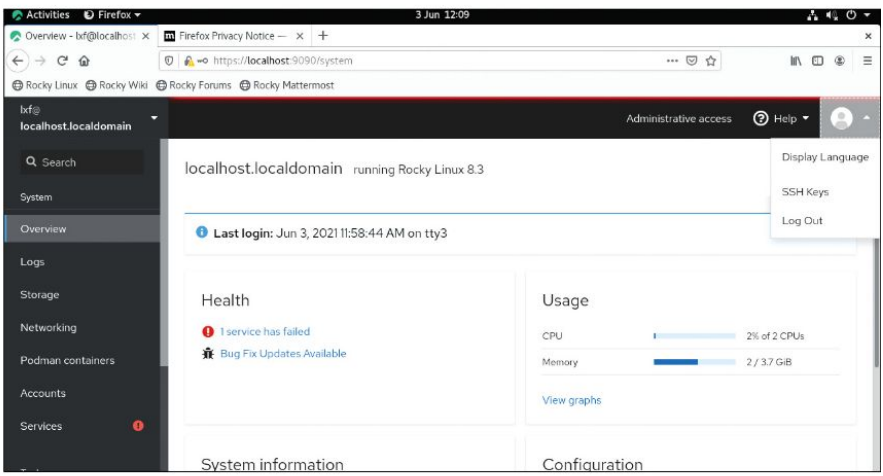
CentOS's co-founder has made his own RHEL-downstream, and the community is behind him.

A petition to save 'classic' CentOS at http://bit.ly/lxf278-centos-petition seems to have petered out at around 12,000 signatures. But we've no doubt that some of our CentOS-using readership will want to resist the pull of the currents and streams Red Hat-ward.

CentOS co-founder Greg Kurtzer was sufficiently motivated as to start a new project, Rocky Linux, with the aim of providing an enterprise distro positioned downstream from RHEL. The name honours Kurtzer's fellow co-founder Rocky McGough, sadly now passed. Thinking about it, even if CentOS wasn't a RedHat trademark, that dedication still makes it a better name than Classic CentOS.

Just after Rocky Linux launched it enjoyed some engagement-metric glory by way of being the top trending repository (for a time) on GitHub. At the time of writing a not-inconsiderable 350 people (all potential contributors) have forked the main distro repository, but more importantly a Release Candidate for Rocky Linux 8.3 has been available since May 1. Before anyone gets too excited, let's just quote the official FAQ before going any further: "Under no circumstance should you use a release candidate in a production environment. A release candidate is provided for testing and validation purposes only". Okay. Good.

One may question why it took so long from the project being announced to this initial RC. After all, one of CentOS's finest achievements was its Community Build System, which behind the scenes is powered by Fedora's Koji, a tool whose raison d'etre is to build and track source RPMs. Again, the answer lies in the FAQ: "If



One of the stars in RHEL's crown is surely Cockpit. And, naturally, it's in Rocky Linux, too.

EXTENDING ROCKY

As with desktop distributions, where sooner or later you'll find yourself adding a PPA or other third-party repository, it's possible to add new package sources to RHEL. The most popular of these is the EPEL (Extra Packages for Enterprise Linux) repository. Since packages here aren't subject to enterprise licencing, anything or anyone compatible with RHEL is free to use them. This comes with the caveat that packages here don't come with the same kind of stability guarantees as the main repo, but you knew that already.

To enable EPEL is a simple matter of:

```
$ sudo dnf update
$ sudo dnf install epel-release
```

You can find a full listing of the 3,000 or so EPEL packages with:

```
$ dnf --disablerepo="*" --enablerepo="epel" list available
```

but if you're looking for the desktop indulgences like KDE and Chromium, or command line gems like *ImageMagick* or *Hashcat*, or other things you feel should be in a distro ,then the EPEL should be your first port of call.

One last thing. Lots of EPEL packages depend on things from the PowerTools repo, so if something doesn't install then try adding it with

```
$ sudo dnf config-manager --set-enabled powertools
```

our only goal for Rocky Linux was to debrand and repackage RHEL, we would have been done much sooner. However, what we had to do differently is figure out how we could keep Rocky Linux in the hands of the community. Carefully devising this strategy ensures that Rocky Linux will never meet the same fate of CentOS."

Testing times

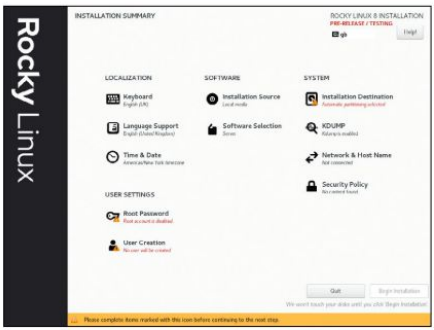
If you want to test out Rocky Linux on a virtual machine, a spare server or in the clouds, then now is a great time to do it. The release aims to be exactly what the CentOS 8.3 was, and as such anyone running the previous CentOS 8.2 release has an easy-ish migration path. There won't be an official Rocky Linux 8.3 release, but rather another release candidate based on the RHEL 8.4 sources. Since RHEL 8.4 was released in late May, that should happen soon. Indeed, by the time you read this Rocky Linux 8.4 may well have already enjoyed its first General Availability release. At which point it's probably safe to start deploying on your computer. There won't be any official way to upgrade from either Rocky Linux 8.4 RC to the GA release, so bear in mind that you'll either have to do a fresh install or sidestep from RHEL 8.4.

Rocky Linux is funded by Ctrl IQ, a company set up by Kurtzer, who is something of a performance computing Don. During his term as technical lead and architect for HPC Services at the Lawrence Berkeley National Lab he developed the Kubernetes-based Singularity, which enables container workloads to run on HPC hardware (see **https://singularity.hpcng.org**) for more information. The Open Science Grid consortium, which provides distributed compute resources for scientific research estimates that some 1.2 billion CPU hours of compute time has been enabled by Singularity (see **http://bit.ly/lxf278-osg-singularity**). Kurtzer's also responsible for the Warewulf cluster management system.

Ctrl IQ as an entity has no governance over Rocky Linux though and, citing the FAQ once again, neither will any corporate interest. Still, Ctrl IQ has already secured $4 million to fund initial Rocky Linux development, so it's good that the not-for-profit foundation which governs the distro can invest in its infrastructure and community.

Rocky Linux is available for 64-bit x86 and ARM architectures and comes in three flavours of ISO: network booting, minimal (which provides the core Server profile and downloads the rest) and DVD (you'll need a 9GB DVD to make a physical disc, though). There's no live environment, but we'll go out on a limb and assume you don't need a step-by-step installation guide. The *Anaconda* installer (*pictured above right*) is friendly enough, and offers to customise your installation to suit various roles. You have to manually enable the network (even for a wired connection) which might seem odd, but remember that Rocky aims to be deployed in a places DHCP might not be the norm. Oh, and if you're planning on installing on a drive with insufficient space you'll need to check the "I would like to make additional space available" box, otherwise the install will fail.

CentOS had plenty of desktop users, in large part those who wanted to mirror the toolchain they had on their CentOS (or RHEL) servers. And as such desktop users are in no way second-class citizens in Rocky – just don't expect the latest in desktop glitz. We were pretty sure we'd selected the Gnome desktop in the installer, but when Rocky booted we were greeted with a textual



The Rocky Install screen, despite the pre-release warning it worked just fine for us.

login prompt, and having entered our details found no sign of Gnome. Perhaps it was a bug with the Minimal ISO image, or perhaps it was a lapse in our memory. Anyway, installing Gnome from the bare-bones install (which occupies about 1.8GB) is easy:

```
$ sudo dnf group install "Server with GUI"
```

We could have used *DNF* (Red Hat's package manager) to install the `gnome-shell` package explicitly, but this lands you an extremely minimal desktop without even a terminal emulator to console yourself. So we make use of the default package groups on offer. To see what else can be installed this way, it's just a matter of

```
$ sudo dnf group list
```

If you don't want a desktop, or only want to interact with your Rocky Linux install remotely, then it's worth checking out the *Cockpit* (*see tutorial* **LXF276**) application which enables you to administer your machine from the comfort of a web page. To start the *Cockpit* service (and have it start with each boot) is a simple matter of:

```
$ sudo systemctl enable –now cockpit.socket
```

Then point your browser to port 9090 of your server's address, or **https://localhost:9090** if you're running Gnome on your Rocky box (the *DNF* group we used pulls in *Firefox* and everything you need to get started with desktop work). Ignore the scary warning about the self-signed certificate. You can add an exception if you don't want to see this again; we're running *Cockpit* locally so it's really not as dangerous as the warning suggests. Log in with the account you set up at install time, and you'll see a glorious overview of your system's health. Delve deeper and you can interrogate logs, display some nice storage graphs and manage SSH keys. From the Applications section of the menu on the left, you can enable the Podman service, which makes easy work of managing the many containers you're surely planning on running. You can also manage SELinux (one of the many reasons security conscious people use Fedora-derived distros) and configure automatic updates from here.

Rocky's *Firefox* package includes bookmarks for the Rocky Linux manual and social media channels. If you do get stuck you'll find answers here. To finish up, we'll look over the page at some more general server setup.

Building better servers

Want to shift to CentOS Stream? Done with RPM-distros entirely? We have options that will ensure your boxes are firmly ticked.

On the wake of the sudden demise of CentOS, a number of poorly written articles appeared on the web purporting to answer the complex question of "What can I use now that CentOS is gone?" in simple, SEO-friendly, oft' listicle-ised form. We hope to do better here, and while we do hope you give Rocky Linux a try we really don't want you to feel pressured into nixing your long-running CentOS machines so early in the game. There's a summary of most of the other RHEL-derived distros in the box (*right*), and thanks to them all aping RHEL (the goal is to be bug-for-bug compatible, remember) they all should work much the same as Rocky.

But besides staying downstream from RHEL, or taking advantage of the free licencing and shifting straight to it, it might be worth considering a different approach. Red Hat's advice is, of course, to switch to CentOS Stream, and if you're okay with being a little bit closer to the edge, then this is sound advice. Remember that CentOS Stream packages are all tested thoroughly in Fedora, before being tested again by Red Hat, so the chance of breakage is pretty low. For the CentOS Stream 8 (there are no point releases as in RHEL) lifecycle, you'll probably find it tracks very close to Rocky Linux (or any descendent of RHEL), but as development accelerates, and certainly once CentOS Stream 9 is released (*Q2 2021 according to Red Hat, are you waiting for it to be released before you submit this feature? – Ed*), it'll start to look a little fresher, version number-wise. CentOS Stream 8 is a transition release. If you're reading this in the future you'll note a major shift in the contribution workflow with CentOS Stream 9. To that end may want to hold off making judgements about the stability or suitability

Upgrading from the final CentOS 8.3 release to CentOS Stream is easy (Red Hat would be laughed out of town if it wasn't). Just swap the repos and let DNF do the hard work:

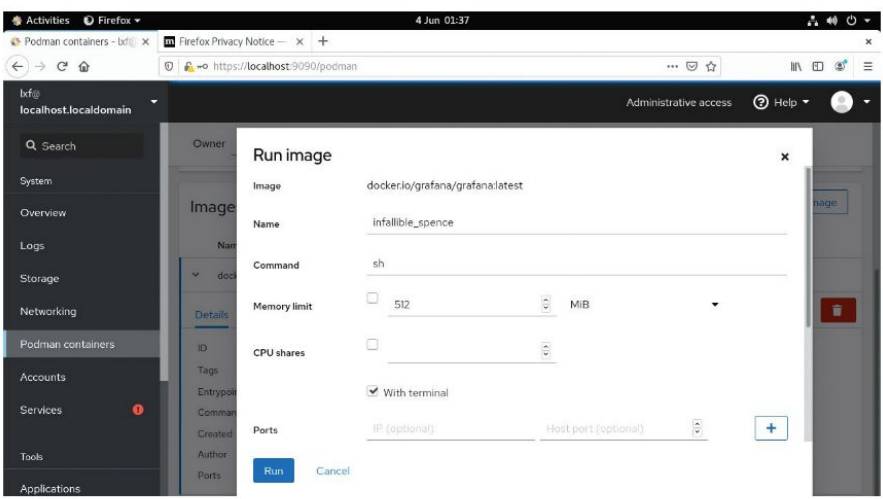
```
# dnf swap centos-linux-repos centos-stream-repos
# dnf distro-sync
```

There's a certain amount of confusion and concern as to how 'rolling' CentOS Stream as a distribution is. Brian Exelbierd puts it quite nicely: "I think people are forgetting CentOS Stream tracks ahead of RHEL, it doesn't track ahead of the universe. RHEL is prided for the fact that it's a stable operating system over that 10-year lifespan. How much change goes in from minor release to minor release? That's the delta you could expect out of CentOS Stream."

Incidentally, the install process for RHEL, CentOS Stream, Rocky Linux and all the distributions mentioned in the box is more or less identical (but for the branding

LOOKING BEYOND CENTOS

"In all the confusion, lots of individuals are choosing to migrate away from the whole class of Red Hat relatives entirely."



Installing the Grafana container with Podman through Cockpit is almost too easy.

in the former). To check out RHEL, head on over to **https://developers.redhat.com/products/rhel/download**. You'll need to create an account (for book-keeping, GDPR-type reasons) and 'buy' yourself a free developer subscription.

Moving on and moving up

In all the confusion, lots of individuals are choosing to migrate away from the whole class of Red Hat relatives entirely. Debian and Ubuntu are seeing a lot of attention from this exodus, which is understandable because these are proven server platforms. We're often accused of being Ubuntu-centric, and this often comes with the implication that we're concentrating on novice Linux users. But people that are deadly serious about their computing run Ubuntu, too. It's running on millions of servers thanks to its awesome administration tools. Having the industrial clout of Canonical behind you is one way to ensure your tooling is good.

If you prefer a more hands-on approach, then Debian is a fine choice, too. It's what we use on our new server (okay, it's more a slice of Amazon resources) and it hasn't broken yet. There's no commercial support for Debian, and it doesn't have Enterprise in its title, but its reputation for rock-solid service is largely indisputable.

Whatever you choose to run on your server, there are some bits of digital hygiene that you should pay attention to. On a desktop machine you'll notice when something is broken: there'll be error messages, beeps or the system will slow down. On a server (especially one that's out of earshot) things can be wrong for a long time before you start to notice. This is why tools like *Cockpit* are useful, because they can tell you at a glance what's wrong. Still, sometimes it's worth checking the old-fashioned way: get thee to the command line and use *Top* to check for resource-hogging programs, check your disk space with `df` and `du`, and look for log files growing out of control in the **/var/log/** directory.

Setting up further monitoring beyond what *Cockpit* offers is a smart move, especially if you're running lots of services, containers or other errands. The *Prometheus/Grafana* combination is highly regarded, but setup can get complex. If you're running a web server there are a number of simple scripts that can analyse your log files for you. Our favourite is *GoAccess* (**https://goaccess.io**), which can run interactively in the terminal or generate a static HTML report with lots of graphs. With just a tiny bit of effort it can even update the HTML, via WebSockets in realtime. *GoAccess* is self-contained and is packaged up for most distros. No RHEL-like packages are available, but it's easy to download and compile the latest version from Git.

We use GoAccess to monitor the millions of bots and occasional human that visit linuxformat.com.

If you're using Gnome and apply updates from the Software app you'll notice they're applied post restart, for safety.

Alternatively use *Podman*, which by the way is Red Hat's own container runtime. It's entirely capable of running Docker containers, and you can manage them all from the comfort of a web interface (like our good friend Portainer). The only slight gotcha on CentOS Stream is that in order to search and use images from the **docker.io** repository, which on Docker you wouldn't need to mention, for example `docker run grafana/grafana`, you'd need to explicitly specify it, so the Grafana image would be `docker.io/grafana/grafana`. You'll notice there are RHEL8 Docker repositories set up in CentOS Stream too, but you can't use them until you sign up for that developer subscription.

It's a complicated business this new age of CentOS, but we hope that this feature has brought some clarity to the situation. This isn't the whole story though, so we've tried to summarise the other RHEL-derived projects below. LXF

» FORKING RHEL

Besides Rocky Linux there's another RHEL-downstream on the scene, which like Rocky was announced pretty much as soon as classic CentOS got the axe. That Linux is Alma Linux (from the Latin word for 'soul') and unlike Rocky it offered a stable 8.3 release, and the stable 8.4 release has just dropped. Check it (and its lovely logo) out at **https://almalinux.org**. Alma's initial release was prepared by CloudLinux, but now it's entirely in the hands of the community, in the spirit of the original CentOS. CloudLinux already has its own commerical CloudLinux OS offering, aimed at hosting providers. It's based on RHEL which perhaps gave them a bit of a head start preparing the first Alma release, but time will tell if it can build the community required to sustain it.

Here's a recommendation some people won't like: Oracle Linux. But it's always been a RHEL clone, and it's free, and there are easy migration scripts from previous CentOS versions. Amazon Linux 2 – though you have to dig deep to find this out – is based on CentOS 7, and at a pinch might be an option for your cloud and virtual CentOS boxes. Probably just another option people won't like.

Finally for those missing Scientific Linux, there's Springdale Linux (**http://springdale.math.ias.edu**), a joint effort between Rutgers and Princeton universities. Springdale has been used by the HPC community for some time. The first major release, when the project was known as PUIAS Linux, predates both CentOS and Scientific Linux. In a sense, then, Springdale has outlasted both of them, but its use appears largely confined to academic circles and specialist installations. Still, if this is you, then maybe Springdale is the one.

THE X FILES

Jonni Bidwell wants to know why a middle-aged display server is still at the heart of the Linux desktop.

Last issue we studied Ubuntu 21.04, which marks the distro's shift from the X.org display server to Wayland (on amenable hardware). But this isn't the first time Canonical tried using Wayland by default.

Ubuntu 17.10 shipped with this configuration, but users were displeased. Come to think of it, Ubuntu 17.10 (Artful Aardvark) also had an artful habit of bricking Lenovo laptops and had to be re-released. And it was disliked by fans of the Unity desktop who saw their favourite desktop replaced by Gnome. Anyway, that Wayland experiment never made it to the next Ubuntu LTS (Breezy Badger), and nor is it present in the current LTS (Focal Fossa). Things have stabilised by now, so barring any mishap the next Ubuntu LTS will finally make the shift to Wayland.

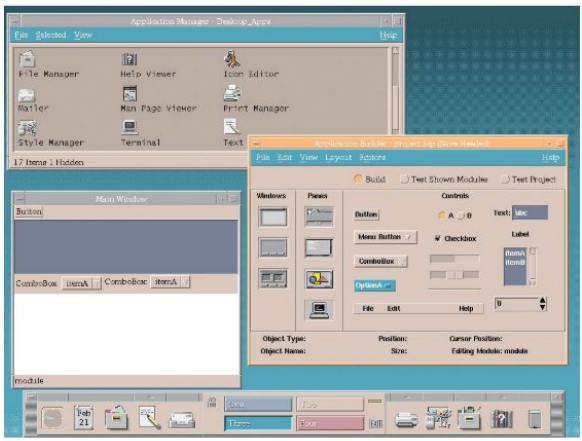
But why do we need Wayland? And what exactly is this X thing anyway? The answers are complex and have lengthy stories behind them. But hopefully this feature adds at least a small amount of clarity to the situation.

Ultimately, X has been with us for over 30 years, and it has no business serving the displays and running the applications that modern workloads entail. Wayland has been with us a while, too. It was first dreamed up in 2008 and has been powering devices (just not necessarily Linux desktop PCs) from the get-go. It's been the default on Fedora for ages, and probably works just fine on other distros, too. Maybe even better than fine…

The venerable X Window System has been powering Linux desktops since the 90s, and powering UNIX (and Ultrix) machines since development began in 1984. It was formed as part of Project Athena, a joint effort between MIT and Digital Equipment Corporation (DEC) with the goal of providing students with computing resources.

The X name, and indeed design, was inspired by a predecessor, the W Window System, itself built for the experimental V operating system (which had a distributed, micro-kernel architecture, but until W had no means of talking to a local display). The main change between W and X is the use of synchronous versus asynchronous drawing primitives, in keeping with Unix design ideals. Functionally, X provides a client/server interface between physical objects (monitor, keyboard and mouse) and imaginary things (the desktop).

One reason why there weren't desktops as we know them until the 1980s was because memory wasn't fast enough to act as a back buffer. Once it could handle sending that buffer to the scan-out port, things became more tractable and enabled systems like X, Sunview, NeWS and Display Postscript to display at 60Hz.



CREDIT: Huihermit, Wikipedia/LGPL

Before Gnome, before KDE, there was the proprietary CDE. It and the accompanying Motif toolkit were open sourced in 2012.

Sharing the knowledge

There's a story behind everything and X's genesis story is about being constantly sidetracked. Bob Scheifler was working on his masters thesis at MIT, on the subject of Argus (a multithreaded language environment). He wanted to debug multiple threads at the same time, and since there was no notion of a window manager or terminal multiplexer in the world of 1984 Unix, this required multiple physical terminals. There also was no notion of Free Software, but there was a culture of sharing between universities and research institutions.

So the W code (originating from Stanford) ended up in Bob's hands and he set about porting it to Unix. Two weeks later he had the world's first version of X running. That and the five subsequent releases were used internally at MIT, but again the sharing culture meant it soon escaped those confines.

X9 was released under the permissive MIT License and a number of components only shipped in binary form. In his 2020 **Linux.conf.au** talk veteran X developer Keith Packard described X10 as 'almost useable' and, thanks to IBM sponsoring a porting effort to their new RT PC (RISC Technology Personal Computer) models, included nearly all the source code for that architecture. Development took place at opposite sides of the country, with tapes being FedEx-ed nightly between MIT and Stanford. This proved unsustainable so further development took place on America's East coast.

In 1987 the underlying protocol was, after a rewrite, more or less standardised and became known as X11. A term which you'll still see today because the protocol

NETWORK OPAQUE

One of the things that makes X unique is its network transparency. This is about more than running programs remotely with `ssh -X`, although that's a convenient side-effect. An X application has no awareness of whether it's running locally or remotely. This was a decision that made sense at the time, given the relative youth of desktop computing, but it adds an unnecessary layer of complication to today's workloads. And that's why there's no network transparency in Wayland, as Collabora's Daniel Stone explains.

"It's deliberate, although *Xedit* or Motif applications work over the network in Wayland as well as they ever did. That's because it's using the server's core rendering and accelerated rectangles and what have you. But since GTK2 I think all that happens is that pre-rendered buffers get sent to the server. So you're just shoving raw pixel buffers over the network, which is pretty inefficient.

"That's why things like *NX* (used in the *NoMachine* remote desktop program) are really useful, because they look at that data more intelligently and compress it accordingly. With Wayland we recognised that it doesn't make sense to have the server doing all the rendering built up from client primitives anymore, because it's so complex. Rather than send these huge buffers, or have the client detect whether it's running locally or remotely and use a different pathway, we eschewed network transparency."

Modern X (using DRI3 and shared memory and everything) is far from network transparent anyway, so this argument doesn't settle any X vs Wayland debates.

Daniel Stone smiled as he corrected Jonni's views on network transparency.

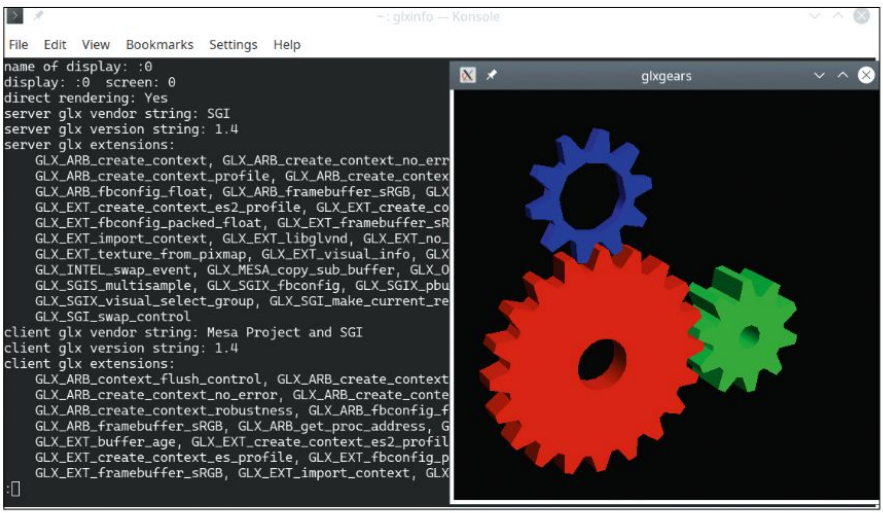
hasn't changed since then. If you've been using Linux since the early days, you might well have used X11R6, which first appeared in 1994. By this time there was a more formal notion of Free Software and being a protocol, a number of different X servers, free and proprietary were available. XFree86 originated as a fork of one of these, and became the dominant fork on Linux thanks to its support for PC graphics cards. XFree86 turned out to be not free enough though, and in 2004 a licencing change was found to be unsuitable for lots of projects and deemed incompatible with the GPL2 by the Free Software Foundation.

So a new freedom friendly fork emerged, called, slightly confusingly given it's also the name of the steering group to promote all things free desktop, X.org. Even more confusingly the display server component of X.org is today referred to as xfree86. An October 2020 blog post calling for X.org to be labeled 'abandonware' attracted a great deal of attention, but it later clarified it was only referring to the xfree86 component

Exactly how X11 has managed to stay useful for over three decades can be answered in a single word: extensions. Pretty much as soon as there were X people started writing extensions for it. The Ximage extension and Low Bandwidth X (LBX aimed at improving performance over slow networks) were among the first. If you're running X now you can see which extensions are in use by running `journalctl -b` and scrolling down to the lines of the form:

```
gdm-x-session[1149]: (II) Initializing extension Generic Event Extension
gdm-x-session[1149]: (II) Initializing extension SHAPE
gdm-x-session[1149]: (II) Initializing extension MIT-SHM
gdm-x-session[1149]: (II) Initializing extension XinputExtension
gdm-x-session[1149]: (II) Initializing extension DAMAGE
gdm-x-session[1149]: (II) Initializing extension RANDR
gdm-x-session[1149]: (II) Initializing extension GLX
```

The Damage extension does sound interesting, but is fairly prosaic in that all it does is allow applications to keep track of which parts of their windows have been updated. By itself, X takes abstract notions like windows, boxes, lines and curves and draws them to a physical screen. *Randr* enables the root window to be resized, rotated and reflected, so that resolution changes or additional displays can be accommodated. And that's why these things no longer require you to log out and log back in again.



Glxgears is not a benchmark, but there's something majestic about those silken, silent cogs.

MECHANISM, NOT POLICY

The positioning of X has always been at odds with other GUIs. The Apple Lisa, in 1983, was the first machine to ship with such a thing and then something called Windows became awfully popular (we're not sure why) after its v1.0 introduction in 1985. But both of these GUIs are much closer to the OS than X is to the Linux Kernel. Yes, we know early Windows ran on DOS, but X's positioning (and the fact that it's only a low-level part of what most people consider a GUI) has always made it more complicated.

Its existence is made even more complicated by the mess of legacies and (probably, with hindsight) bad ideas that it's inherited. One of the most regretted of these is "mechanism, not policy". The idea was that X wasn't going to tell you how to be a GUI. Instead, it would accommodate (through many, many mechanisms) any and all kinds of different appearances and behaviours, and enable application developers and desktop environments to make these design decisions. You might think that the Linux desktop situation is bad today, but spare a thought for the hard-working people at **freedesktop.org** who somehow have enabled so many different desktop environments with so many different applications to somehow interoperate.

The Composite extension was introduced in 2004 and enabled compositing window managers (like *Compiz*) to do effects like transparency and wobbly windows, which became terribly popular around this time. Direct Rendering Infrastructure (DRI) enabled applications to access graphics hardware directly and enjoy acceleration through the Mesa OpenGL drivers. This meant some exciting games could run on Linux; we in particular remember *Unreal Tournament 2004*. The GLX extension enabled OpenGL applications to draw directly to an X window, so that users could entertain themselves with the mesmerising rotation of three coloured cogs. The *glxinfo* program (from the `mesa-utils` package) gives you information on OpenGL extensions and drivers. A key diagnostic step to test your 3D setup was working correctly was to run and look for the line

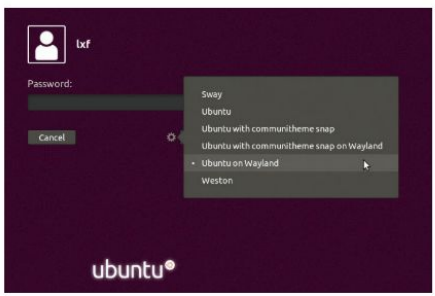
```
direct rendering: Yes
```

No need for hardware configuration

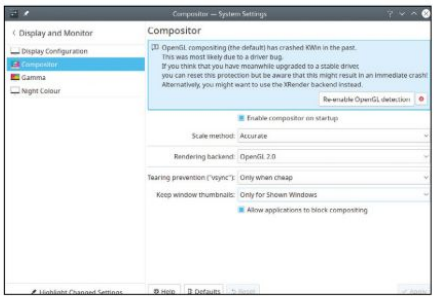
In August 2008 we interviewed Keith Packard (see **LXF108**, The X Factor) at Fosdem. Back then X11 was on the cusp of being able to start without at least a minimal **xorg.conf** configuration file describing your GPU and monitor. Indeed, if you look at forum posts from the era you'll see questions about which options to specify to keep the Nvidia driver happy, or how to get 3D acceleration so you can play *Quake* or *Glxgears* faster.

It's often said that misconfigured monitor timings in **xorg.conf** risk the magic smoke escaping from your monitor, but the last monitor we broke released no such smoke. Only a persistent whining sound, gently increasing in volume until such time as our survival instinct kicked in and we unplugged it. Anyway, you might think this phenomenon is the stuff of the past. And hopefully it is, but as Keith pointed out it's not a problem that went away with analogue displays: "It's interesting that X used to be able to destroy CRT monitors, but they've now got pretty bomb-proof. If you program internal LCDs wrong you can get them driven at DC, which tends to boil the fluid inside the LCD... it can catastrophically destroy your panel."

Fortunately most drivers today will get this information from your monitor. They will even work around any known misreportings for buggy hardware. And the magic smoke will remain safely within. Collabora's Daniel Stone, another contributor to the free graphics stack, told us (back in **LXF243**) about some of the latter developments and reshapings of X:



"On Ubuntu this tiny cog is how to choose between ancient display tech and the future."



KDE's desktop compositing enables all kinds of pretty effects, and also (when things go awry) this helpful warning.

"A lot of the evolution since configless X has been moving as much as possible out of the X server. So Kernel Modesetting (KMS) was pulled out from the userspace drivers and input handling was pulled out into separate libraries. Even though this was improving X at the time, a lot of the idea was proactively trying to make it possible to replace X. It's similar with Wayland: the core is super small and all the interesting bits people have been doing recently are things like PipeWire, for video streaming, and making it possible to work with Flatpak and stuff."

Even after all this decoupling and refactoring and removing of unnecessary features (why did a display server need a print server?) the X.org codebase, after three decades of plugins and extensions, is nightmarish. Very few people alive understand all of it, and a quote from Gnome developer Thomas Thurman describes the situation nicely: "Programming X is like reading one of those French philosophers where afterwards you start wondering whether you really know anything for sure". The display server had become something akin to its own operating system.

X MARKS THE KABOOM!

"It's interesting that X used to be able to destroy CRT monitors, but they've now got pretty bomb-proof."

Another concern about X is security. It's possible for one application to manipulate another's windows, or harvest keystrokes. Long-time kernel contributor Matthew Garrett made a blog post about this back in 2016 (showing this was a problem even for then-new Snap applications, see **https://mjg59.dreamwidth.org/42320.html**), as well as a proof-of-concept tool called *Xevilteddy*. There are extensions that can lock down X to some extent, such as SELinux. But it's hard to use these effectively without losing functionality.

With Wayland an application's memory is its own business. There's no peeking into or poking another program's memory space. There aren't a million different legacy drawing commands to deal with. In fact there's no drawing API at all. All Wayland cares about is buffers filled with pixels. And all we care about right now is getting this feature off to the printers because the clock seems to be ticking faster than usual this day... LXF

Pete Lomas is the co-founder of the Raspberry Pi Foundation and co-creator of the Raspberry Pi.

» SALUTE TO THE MAKERS

The best part of my roller-coaster journey with Raspberry Pi has been, and continues to be, the fantastic community that has grown up around it. It resonates loudly with my experience as a young college student. Graham Beech, a laboratory technician, introduced me to computers in the form of a PDP8. His belief, help and encouragement changed my life for the better.

The Raspberry Pi mission has galvanised makers, teachers, professional engineers, enthusiasts and folks taking their first steps in "digital making" to all come together and help each other move forward and create something truly wonderful.

I've been hugely fortunate to meet so many of the Pi community at the hundreds of events I've attended over the years. From São Paulo to Stockholm, I've been in awe of your creativity and unbounded enthusiasm for making with Raspberry Pi and other hardware. I vividly remember the "Paddington hard stare" from the family when I announced bunking off two days of a skiing trip to chat to Vancouver hackspace!

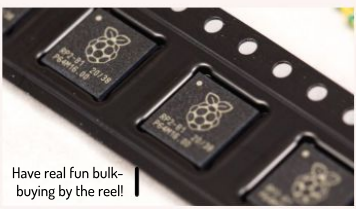
I'd like to call out some stalwarts of the community, many who have given a huge amount to the mission and getting the original Raspberry Pi up and running and still do today, but I fear missing someone out. If you created a mega project or have just been able to get that LED to flash, you're all heroes to me. You've helped democratise programming and electronics in a way that would have been impossible without you – long may it continue!

Huge demand pushes up RP2040 SoC sales

A silicon-starved world demands its integrated circuits and Raspberry Pi is trying to play catch-up.

The Raspberry Pi Pico went on sale at the start of the year and is powered by the RP2040 SoC. Furthermore, the RP2040 surface mount device (SMD) integrated circuit (IC – the chip itself) has been available to select industry patterns. Hence Pimoroni, Adafruit et al being able to develop spin-off RP2040-packing products.

It turns out that because of unprecedented demand the programme to provide the RP2040 IC package was accelerated by three months to the beginning of June 2021. This resulted in 40,000 units being pulled out of general circulation and offered to makers and other businesses at the price of $1.



Have real fun bulk-buying by the reel!

CREDIT: Raspberry Pi Foundation

Eben Upton of the Raspberry Pi Foundation explains, "This will give people time to develop their projects and products, while we clear out the rest of the Pico backlog and scale up production of RP2040. In the autumn we'll have serious volume available to serve any resulting demand."

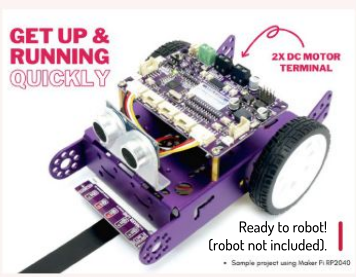
There are no shortage of eager makers looking to incorporate the RP2040 into their projects. Arturo182's RP2040 Stamp is a one-inch square breakout designed to drop into a project, Seeed's Wio RP2040 is the cheapest way to get your RP2040 online and Guido Burger's RP2040 CO2 sensing ring is a novel means to measure your environment. All of these projects are powered by the RP2040.

More details at **www.raspberrypi.org/blog/raspberry-pi-rp2040-on-sale**.

Pico robot board

A Cytron for sore eyes.

Talking of RP2040-based projects, here's Cytron with its $10 Maker Pi RP2040. Unlike the other Pico-clones this goes for its own robot-targeting 88x64mm form factor. With two motor terminals, four servo ports, seven I/O grove ports, power and reset switches, three push buttons and li-ion battery port it's ready to rock. See **http://bit.ly/lxf278cytron**.

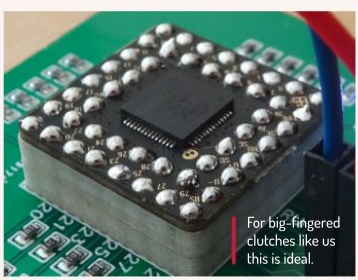


CREDIT: cytron.io

All-access fun

Breakout RP2040 board.

Shortly after the RP2040 SoC announcement Pimoroni tweeted an image showing an RP2040 breakout board, named PGA2040, that provides access to all of the GPIO pins. Pimoroni said "The PGA2040 is a fairly minimal breakout for the RP2040 with standard 2.54mm (0.1-inch) headers in a Pin Grid Array."



For big-fingered clutches like us this is ideal.

CREDIT: Chris Parrott/Pimoroni

Ubuntu 21.04

Les Pounder has long been a fan of Ubuntu and Hippos, but had to look up Hirsute in the dictionary.

IN BRIEF

The latest Ubuntu release and the first to use Wayland by default. We have a marginal increase in overall performance with Wayland, but "The Raspberry Pi Experience" is far from stellar. If you want to build cool electronic projects then Raspberry Pi OS (Raspbian) is still the best on offer. But for a modern Linux desktop, Ubuntu 21.04 covers the basics.

Ubuntu has been the first Linux distro for many new users, but not for the majority of Raspberry Pi users. Ubuntu 20.10 was the first to release a simultaneous Raspberry Pi and server/desktop image. Six months on and the latest release – 21.04, codename Hirsute Hippo – arrives for those same platforms and so we took it for a test drive on a Raspberry Pi 400. The 64-bit image is also compatible with the Raspberry Pi Compute Module 4, but we've lost compatibility with older models of Pi.

Installation was via the official *Raspberry Pi Imager* tool and following an initial setup script we were faced with the familiar Ubuntu desktop running on top of Linux Kernel 5.11. But not everything is as it seems. Ubuntu 21.04 saw the distro come with Wayland as the standard display server, with X11 as a backup, and our Raspberry Pi release has the same choices. To take advantage of this you'll need to reboot the machine because the post-install/setup default is X11.

The Ubuntu desktop experience is as pleasant as it is on more powerful devices, and we can get some work done, which is always welcome.

BooTube more like!

Ubuntu 21.04 Wayland on the Raspberry Pi is a solid desktop experience and with the Pi 400's 1.8GHz quad-core CPU we had ample power to push things along, but a fast micro SD card or USB 3.0 SSD will produce the best results. We tested out basic web browsing and the usual desktop applications and all went rather well, with the exception of YouTube, which dropped frames even at 720p. Worse still, 1080p 60 was a slide-show with 95 per cent of the frames being dropped. We repeated the test on X11 and that saw 480p video dropping frames even over a 5GHz Wi-Fi connection.

The Pi is famed for its GPIO and we quickly connected up our LED test circuit. Our software choice for this test was Python 3 and the *GPIO Zero* library. The obstacle here was that *GPIO Zero* wasn't installed, nor the *pip* Python package manager. The reason why is clear: software should be installed via the default system package manager, but *pip* is ubiquitous in the Python community.

After installing *pip* and *GPIO Zero* we tested our LED, and hit another issue: our user didn't have permission to use the GPIO. We could have forced the issue with *sudo*, but after a little research we found that we could add our user to the dialout group to achieve the goal. Something that Arduino users have been aware of for some time.

```
les@les-pi400: ~
les@les-pi400:~$ sudo usermod -a -G dialout les
[sudo] password for les:
les@les-pi400:~$
```

Want to hack around with the GPIO? You'll need to add your user to the dialout group.

If you'd like to use your favourite Pi HAT/add-on then your mileage may vary. We tested Pimoroni's Unicorn HAT, which uses PWM to control the 64 WS2812B LEDs, but we were unable to get it to work correctly.

Ubuntu 21.04 officially supports only the 400 and CM4 models of Pi, but that won't stop many from using it on their Raspberry Pi 4. However, you'll need to overclock the Pi 4 to achieve the best results. A quick edit to the **config.txt** file found in **/boot/firmware** is all it takes to squeeze a little extra power from your Pi.

Ubuntu 21.04 for Raspberry Pi is a small step forward from 20.10. The inclusion of Wayland is not the breath of fresh air that it is on the desktop, but it's a welcome move. The issues with the GPIO and lack of compatibility with Pi accessories will prevent many from sampling Ubuntu. If you need a solid desktop machine then the Raspberry Pi 400 and Ubuntu 21.04 are a good combo. But Raspberry Pi OS is still the dominant distro for the Pi, despite being a 32-bit OS versus Ubuntu 21.04's 64-bit nature. LXF

VERDICT

DEVELOPER: Canonical
WEB: https://ubuntu.com
LICENCE: Free software; some proprietary drivers

FEATURES	**6/10**	**EASE OF USE**	**8/10**
PERFORMANCE	**6/10**	**DOCUMENTATION**	**8/10**

A small step forward for Ubuntu on the Pi. It looks great, but it still can't overcome the dominance of Raspberry Pi OS.

» Rating 7/10

SCRATCH

Scratch sound and motion detection

Our intrepid explorer **Les Pounder** has escaped from the planet of the evil space bats. But even in zero G he needs to lose a little weight.

OUR EXPERT

Les Pounder is associate editor at Tom's Hardware and a freelance maker for hire. He blogs about his adventures and projects at bigl.es

Continuing our series of Scratch projects, we've finally escaped from the planet of the snack-stealing space bats and are recovering in our starship's extensive leisure facilities. To be specific, we're keeping fit in the astro-gym with *Ball Blaster*, a game where you punch balls to score points, but you only have 30 seconds! To control the game we use a webcam that will look for our hands and react when we hit a ball. To start the game we need to shout "GO" and then the countdown starts!

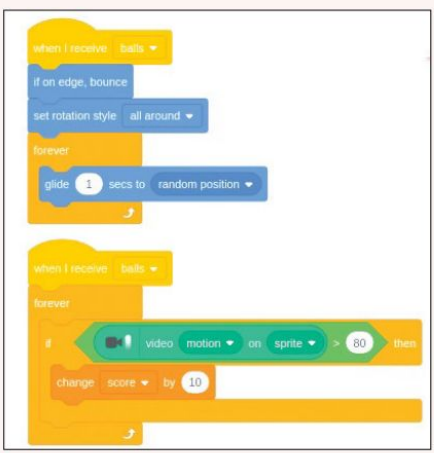
Plug in your USB webcam and it should just work. Scratch 3 should come pre-installed on your Raspberry Pi OS image, but just in case it's missing it can be installed from the main menu, under Preferences> Recommended Software. Scratch 3 is found in the Programming category – place a tick in the box and click Apply to install. Once installed, Scratch 3 can be found in the main menu under Programming.

Open Scratch 3, on first start Scratch may take a little while to open. We're going to assume that you have an understanding of how to code with Scratch. If not, we covered the basics in the previous issue. Before we write any new code we need to click the blue folder icon in the bottom left of the screen to load the Extensions menu. From their select Video Sensing and a palette of new blocks is added to our code. We shall use them later.

The game is best played full screen on a large TV, which will enable you to get the most exercise as you run around chasing balls!

YOU NEED

- **Raspberry Pi 4 or 400**
- **Raspberry Pi OS**
- **A USB webcam**
- **Play it at https://scratch.mit.edu/projects/527621210**



The code for the balls has two sections: one for moving the balls and another for reacting to user input.

Create a sprite for the game

We're going to start with a new sprite. In the bottom right of the screen click the Cat sprite and then click the trash can icon to delete the cat. In the same section look for an icon which looks like a cat with a plus sign above it – click this and select `Choose a sprite`. Select the basketball sprite and it'll appear in the sprite box.

We're going to write code for the basketball, so make sure that it's selected. The first block of code is the trigger that starts the game. From the Events blocks, drag the `When Green Flag Clicked`. From the Looks block drag both `show` and `say hello for 2 seconds`. Place them in that order underneath the green flag block. When the flag is clicked, the basketball will appear, and we need to change `say` so that it instructs the player to `Say GO to start`.

Next we go to the Variables block and click `Make a Variable`. Call the variable `score` and make it available for all sprites. From Variables drag the `Set score to 0` – you may need to use the drop-down menu to select the variable. Now go to Control and drag a `forever` loop and connect it to the previous blocks of code.

The next set of blocks are inside the forever loop. From Control, drag `if... then` and place it inside the loop. This is a conditional test, a question. In this case we need to

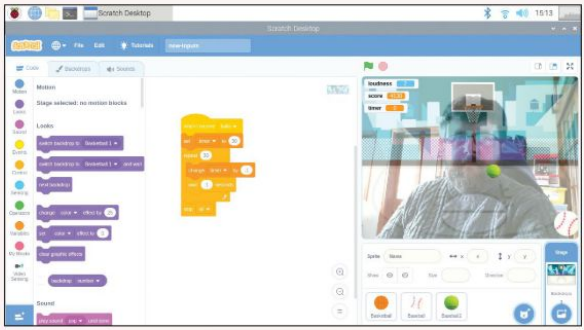
go to Operators and drag __ > __ and place it inside the hexagon-shaped blank of if... then .
In the second blank type 50, and in the first drag loudness from the Sensing blocks. Loudness uses the microphone to detect noise, and the noise level is given a value from 0 to 100. If we shout GO then it should be over 50 and trigger the game to start. But this number may need tweaking for your home setup. To start the game we need to send a message to all of the sprites and for this we use broadcast found in Events. Drag broadcast inside the if... then section and use the drop-down to create a new message called balls . Still inside the if... then drag say hello for 2 seconds and hide from Looks. Change the say block to read GO! . Now when the player shouts Go , the basketball will speak to the player and then disappear.

Create a new sprite using the Choose a sprite button in the bottom right of the screen. Select a baseball as the sprite and the coding area changes to enable us to write code for that sprite.
We start with when I receive balls from Events. Drag two of these blocks into the coding area. Under the first one drag if on edge bounce and set rotation style from Motion. Set the rotation to all around . Now from Control drag forever and place it under the previous block. Inside forever drag glide 1 secs to random position . When this sprite receives the message balls it'll set the sprite to bounce off walls and to reflect at realistic angles off the borders of the screen.

The ball will glide around the screen, similar to a ball bouncing. Under the second when I receive balls block we need to add a forever and if... then from Control. The if... then should be nested inside the forever . From Operators drag __ > __ and place it inside the hexagonal-shaped blank of if... then .

In the second blank type 80 and in the first blank drag video motion on sprite from the Video Sensing blocks. This will use the camera to see if we're waving/punching on a sprite and it'll check to see how fast we are waving. In reality this block checks for movement and assigns a value, so you may need to tweak 80 to match your goals. Inside the if block we need to drop a change score by block from Variables. The score should change by 10 points if the baseball is hit.

In the bottom right Sprite screen, right-click the baseball and select Duplicate. Now we have two balls bouncing in the game. But this second ball should give us more points and so be harder to hit. To make the balls faster, change the glide 1 secs to 0.5 seconds. To make the ball look different click Costumes and click the Choose a costume button in the bottom left of the screen. Select the tennis ball. Back in the Costume editor, click the tennis ball on the left side of the screen to make it the default. Click code when done.



The stage or backdrop graphic can be customised to match our game, and it has hidden code to run our game timer.

The final sections of code are for the stage, where our game takes place. Click the stage section, in the far right of the screen. This section of code starts with When I receive balls and under that we need to create a new Variable, called timer, and then drag set timer to 0 and place it under the block. Set the value to 30. From control drag repeat 10 and place it under the previous. Change 10 to 30.

Inside the loop drag a change timer by 0 from Variables. Set the value to -1. From Control drag a wait 1 seconds block. So now this loop will timer from 30 seconds to 0, the time for our game. When the timer reaches zero, the loop ends and the final block, stop all is dragged from Control to the end of the code. If you like you can also add a new backdrop using the Choose a Backdrop button found in the bottom right of the screen. This changes the look of the stage, but we can still see the live video preview overlaid on top.

Click the Green flag to run the game and get ready to score lots of points and keep fit. It's a win-win! LXF

QUICK TIP

If you need to see the current value of "loudness", click the tick next to the block in Sensing. A box will appear in the top left of the stage.

» CONTROL YOUR WEBCAM

Living through a pandemic has greatly increased the price of webcams. As more and more of us work from home, speak to loved ones using video chats, or record cute cat videos for YouTube, demand for good-quality webcams has pushed prices of even basic cameras. However, for this game you don't need the latest and greatest camera. We used a 10-year old Microsoft 720P camera and it worked with no driver installation. The official Raspberry Pi camera can't be used in this project because it doesn't have a microphone, which is necessary to detect sound and start the game.

If you have a cheap webcam lying around, you can use it with a Raspberry Pi and a quick way to test if it works is to use *guvcview*, a great webcam tool that enables us to tweak the camera settings for the best picture. To install, open a terminal and type the following:

```
$ sudo apt update
$ sudo apt install guvcview
```

Guvcview has two windows: the image preview and the controls. Under controls we can set the image, video and audio settings and then use Cap Image and Cap Video to grab images and video. It's a useful tool on any Linux machine.

MAKER BASICS

Getting to grips with breadboards

Mike Bedford provides a hands-on guide to using breadboards for those starting out with the Raspberry Pi or other single-board computer projects.

OUR EXPERT

Mike Bedford used to be more comfortable dealing with software than hardware, but his familiarity with the latter shows that you don't have to stick with just the one area of expertise.

One of the best things about Raspberry Pi or Arduino single-board computers (SBC), or similar, is that they provide a GPIO (general purpose input/output) facility that enables them to be connected to external devices. This offers the opportunity to learn about embedded computing and, once you're up to speed, use your SBC in so many exciting projects. Usually, though, you can't attach those external devices to pins on the GPIO. Instead, it'll be necessary to use some simple interfacing circuitry.

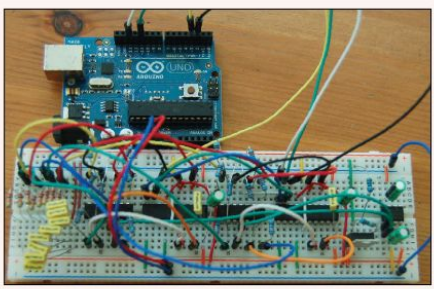
Without some experience of electronics this might sound daunting for a couple of reasons. First, it won't be immediately obvious what arrangement of components is needed. Even if you're presented with a circuit diagram, it might not be easy to convert that schematic into a real-world assembly of components that you can attach to your single board computer. It's that second issue that we address here by providing an introductory guide to building circuits on so-called breadboards. Why the word breadboard? It's because at one time electronics engineers used real wooden boards (even using actual dime-store breadboards) for prototyping circuits, using wood screws as binding posts.

Breadboard benefits

The traditional way of building electronic circuits involves soldering them together. That's exactly how the motherboard in your PC will have been manufactured, but it's built on a PCB (printed circuit board) which you're not going to want to design until you're certain that your circuit works as intended. An option that's more suitable during the prototyping and testing phase is to use a stripboard, but it still involves soldering, for which you'd need a soldering iron and a few other hand tools. That means that making alterations to the circuit isn't easy and the components can't really be re-used.

As an alternative to soldering, a circuit could be assembled using a piece of terminal strip, but although this enables changes to be made simply and components could be re-used, it's really only suitable for trivially simple circuits.

A breadboard provides a win-win solution for testing out circuits. Depending on its size, it even enables fairly complicated circuits to be built, and because it doesn't involve soldering, changes can easily be made. It's even possible to reuse the components, although you need to recognise these are just for early prototyping and are unsuitable for field work.



CREDIT: George P. Macklin, CC BY-SA 2.0, www.flickr.com/photos/gmacklin/5079459828/

You're not going to be building a complex circuit like this anytime soon, but this example shows the potential of breadboards for prototyping.

A breadboard is a plastic board that's drilled with holes on a 2.5mm (0.1-inch) grid. Clips inside the board will grab and hold a component lead or the rigid end of a patch lead that's pushed into a hole. Those clips are wired together, inside the breadboard, in a clearly defined pattern (*see top-right*), thereby enabling components to be wired together to form a circuit simply by pushing them into the board.

Boards vary in their size, but a typical board is illustrated on the facing page, and we'll use this diagram to explain the layout of a breadboard. For now, ignore the gold-coloured lines, because you won't see these when you look at a real breadboard, and we'll explain them later.

This particular board has 300 general purpose holes, with the columns labelled 1 to 30 and the rows labelled a to j. Any of these holes can, therefore, by uniquely defined. For example, the one at the top left is j1, although you don't usually need to concern yourself with the labelling of holes. However, if you anticipate rebuilding a circuit that you've developed, you can write down a description of the circuit by reference to these labels (for example, 10K resistor from j1 to j6) so you can follow those instructions next time. In addition, some descriptions of circuits that others have designed refer to breadboard labels.

QUICK TIP

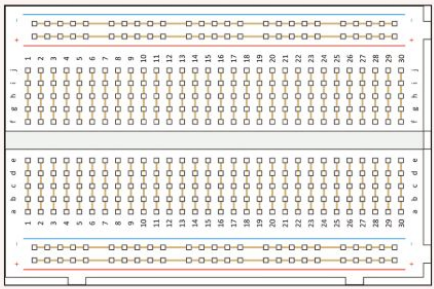
Some components have stranded wire leads because these are flexible, but they don't easily plug into a breadboard. Sometimes the manufacturer tins the end of the leads with solder. If not, the easiest solution is to extend each of the device's leads with a proper patch lead, using a small piece of terminal strip.

Next up are the two rows of holes at the top and bottom of the board that aren't labelled individually, but are labelled + or -, and often also carry red and blue lines. These holes are intended for the power supplies, typically the negative supply – called Ground on Raspberry Pis or GND on Aruninos, and which you may also see referred to as 0V, but which we'll refer to as "-ve" from now on. There's also the positive supply – which will either be +3.3V or +5V depending on the SBC, but which we'll refer to as +ve.

The gold lines represent the internal connections, and you'll see that the holes in each of these rows are connected together. However, the top and bottom +ve rows aren't connected together, and nor are the top and bottom -ve rows, so if you want to use both the top and the bottom rows you'll need to connect them together using patch leads. There are lots of holes for power supplies because they're used extensively in circuits. Typically, if you're using the breadboard for SBC interfacing, you'll need to connect these rows to the power supply pins on the SBC using patch leads.

Having now seen what the gold lines represent, the connections between the general-purpose holes will also be clear, and you'll notice that they're connected together in vertical groups of five. If you need to connect more than five things together, though, you can use a patch lead to extend the group of connections by another five.

You'll see there's an area across the middle of the board with no holes, and there are also breaks in the connections between the vertical columns at this point. This area is identifiable because commonly there's an obvious "valley" across the board. The main purpose of this valley is to enable the use of DIL (dual in-line) integrated circuits. DIL describes a common integrated circuit package that has two rows of pins along its two long edges, these two rows being separated by 0.3 inches or more, depending on the number of pins. These should be plugged into the board with the two rows of pins on opposite sides of the valley.



The thick gold lines are invisible on a real breadboard. They're shown here so you can understand the internal connections between the various holes.

Larger breadboards might have more than one valley to enable the use of more DIL ICs, and they often have breaks in the connections in the horizontal power supply rows. This means you can use multiple power supplies, but if you're using the same power supplies in all your circuit, you can bridge these gaps using patch leads. Also on the subject of boards for larger circuits, an alternative to using a large board is to stack smaller boards together and, to permit this, breadboards often have lugs on two edges that will clip into slots on the opposite two edges of adjacent boards.

Stocking up on components

Next, we need to introduce the components that you'll be using. In time you might need more, but here we suggest what you might need if you're just starting out and want to have enough to try things out as a learning exercise, and to prototype some simple projects.

Almost certainly you'll be interfacing LEDs in several colours, together with pushbuttons, so these are the first types of components you'll need. Interfacing the LEDs will require resistors, but the value will differ depending on the specific type and colour of the LED and the voltage of the power supply, so you'll need a variety. As you move on, you might need various capacitors, diodes and transistors, and possibly some of

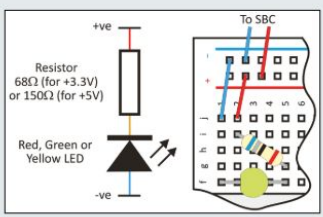
» APPLY YOUR BREADBOARD KNOWLEDGE

A simple exercise will help you to put into practice the principles you've learned in this article. The word simple is an understatement but, as they say, from tiny acorns mighty oaks grow.

First, build the circuit that's shown on the left of the diagram (using a red, yellow or green LED because blue or white ones will need a different value of resistor), but without making the connection represented by the red line. Although lines on circuit diagrams would normally appear black, we've coloured them in the same colours as the corresponding patch leads that appear in the diagram on the right, which is a portion of a breadboard.

Because the connection relating to the gold line is made via an internal connection in the breadboard, only connect the red lead between holes on the breadboard, and the red and blue leads from the breadboard to the SBC, which is only acting as a power supply. Having done this, the LED won't illuminate, because one end of the resistor is unconnected. To see it light, use the other red patch lead to complete the connection of the resistor to +ve.

Now, to use the SBC to control the LED, disconnect the upper end of red lead that runs from the resistor to the +ve row of the breadboard, and connect it instead to a pin on the SBC's GPIO that can be used as a digital output. Finally, write a simple bit of code to flip that digital output between 0 and 1 every second to see the LED flash on and off.



Our simple hands-on exercise involves building the circuit on the left by wiring together the components shown on the right.

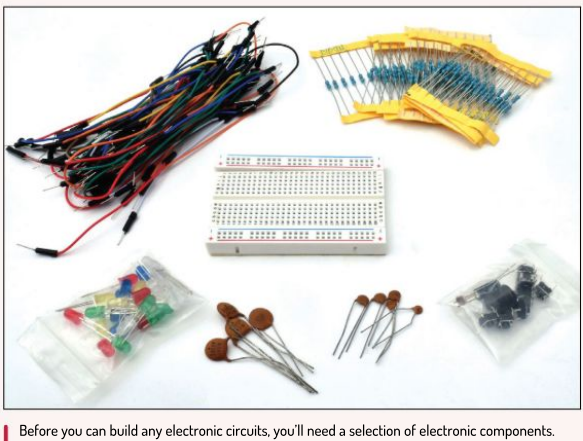
QUICK TIP

When you buy a set of patch leads, you'll probably be supplied with a variety of colours. Using specific colours for particular purposes will help you to spot mistakes if your circuit doesn't work. For example, you could use black or blue leads for -ve and red leads for +ve.

the more common types of integrated circuits. You'll also need some patch leads for wiring them up on your breadboard and connecting the breadboard to your SBC. For wiring between components on the breadboard you'll need male-to-male patch leads, and these will also enable you to connect the breadboard to an Arduino's GPIO. However, if you want to connect it to a Raspberry Pi, you'll need some male-to-female patch leads because the GPIOs on these SBCs have male pins.

If you're just starting out, the easiest solution is to buy an electronic component starter kit, which start at about £10. These are available from many sources but look carefully into what's included because they vary so much – some even include a breadboard and patch leads, but there might be some drop-offs. One that looks quite good is the 613-piece Electronics Starter Kit from **www.bitsbox.co.uk** (£14.99), which includes a breadboard, although it doesn't include patch leads or pushbuttons, and you'll probably need some additional values of resistor, but all these are available separately from the same supplier.

We suggest that you don't skimp on the starter kit.



CREDIT: pixabay.com

Before you can build any electronic circuits, you'll need a selection of electronic components. Buying a starter kit is highly recommended.

Yes, electronic components are cheap, but if you discover that you're short of something, you'll end up paying a lot more for delivery (there's no Maplin you can pop to) than the cost of the few extra components.

A final comment on components concerns resistor values. These are printed on to resistors as coloured bands instead of numbers, but this can be confusing to the uninitiated. Some component kits include a "cheat sheet" to help you read resistor values. If this is missing from the kit you buy, take a look at **http://bit.ly/lxf278resist**.

Polarised components

Many components have just two leads and can be connected either way round. This is the case for resistors and many types of capacitor. However, there are also some components with just two leads, yet they have to be connected a particular way round because they're polarised.

A common example of a two-lead polarised component is an LED. Since this is probably one of the first components you'll be using, we need to explain this. In particular, it has a positive terminal (the anode) and a negative terminal (the cathode). In following a circuit diagram (otherwise known as a schematic or a wiring diagram) you need to be able to identify the anode and the cathode on the symbol for the LED in the circuit diagram, and you also need to identify these two leads on the actual component.

The diagram (*above right*) shows the symbol for an LED, with the anode and cathode identified, and also shows the physical component, again with these two leads identified. You'll notice that the cathode on the component is identified by a flat edge on the LED's round body, and also by having a shorter lead than the anode. Fortunately, if you wire an LED the wrong way round you won't burn it out – unlike some components – but it certainly won't illuminate. As a word of warning, though. Even though it's not related to polarity, if you use a current limiting resistor that has a much lower resistance than the recommended value, you will destroy the LED.

Other two-lead polarised components that you'll encounter sooner or later are diodes and those high-

» MAKE USE OF TEST EQUIPMENT

Two pieces of test equipment will help you check out your breadboard circuit and find any faults. The first is a logic probe. It looks rather like a pen with two wire leads at one end and a thin metallic point on the other end. Clip the two wires onto -ve and +ve and then, if you touch part of the circuit with the metallic point, the red or green LED will light up to show logic 1 (+ve) or logic 0 (-ve) respectively.

The other is a test meter, sometimes referred to as a multi-meter. As the second name suggests, it can measure several electrical characteristics. First, it can detect continuity, bleeping if there's a dead short between the meter's two leads. This will prove if there's a connection between two points in your circuit. Second, it can show you the resistance between whatever you connect to the meter's leads. So, if you clip the leads onto the leads of a resistor, it will confirm if the resistor is the value you expect. However, if you try to measure the resistance of a component when it's in circuit, it's possible that the resistance will be affected by other components in the circuit. And finally, you can measure the current. However, to measure the current flowing in a particular part of a circuit, you have to make a break in the circuit at that point so you can wire the test meter into the circuit in series.

A test meter enables you to measure continuity, resistance, voltage and current. It'll simplify fault finding and a basic one won't cost much.

value capacitors that are described as electrolytic or tantalum. These components also appear on the same diagram as the LED. You'll see that the symbol for a diode is similar to that for an LED (which is a type of diode) but without the two arrows, so once you can identify the anode and cathode on the symbol for a LED you'll be able to do the same for a diode. And the cathode is identified on the physical component by a band printed on its body. We show two different symbols for polarised capacitors because standards differ. Take care when wiring electrolytic capacitors: if they're the wrong way round they could explode.

Building a circuit

Now we're going to see how to build a circuit, but we aren't describing how to design the circuit in the sense of deciding what components are needed and how they should be connected together. Designing the circuit is a major subject in its own right, but it's too involved to get embroiled in here as just an aside from our main theme.

Initially, therefore, you'll be building up circuits that other people have designed, but for which they haven't necessarily published a breadboard layout. Please don't give up on the idea of eventually designing your own circuits, though, because it isn't too difficult to interface commonly used components such as LEDs and switches, and you can work your way up from there.

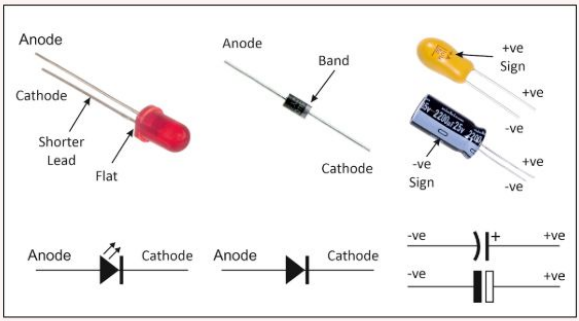
Knowing a breadboard's internal connections, the size of the components you'll be using and the spacing between their leads, it's fairly simple to figure out the arrangement of the components to implement a particular circuit. Despite the simplicity of the process, there are a few practical aspects to bear in mind.

LEDs have radial leads, which means they both protrude from the same side of the component's body. This means that they can be plugged straight into a couple of nearby holes on a breadboard. However, if the two holes are too far apart the leads won't reach both the holes, so you'd need to create two 45-degree bends in one or both of the leads.

Resistors are different in having axial leads, which means that they protrude from opposite ends of the component's body. There's no way, therefore, that you could plug one into a breadboard without bending at least one of the leads. One way is to bend just one of the leads through 180 degrees so it runs parallel to the other lead that's just a short distance from it, enabling the resistor to be plugged into two nearby holes with its body vertical. The other way is to bend both leads through 90 degrees so they both point in the same direction, allowing the resistor to be plugged into two more widely separated holes with its body horizontal.

Bending leads that are close to the body of a component risks damaging it unless you do it properly. You should clamp the lead close to the component's body using a pair of small-nosed pliers while you make the bend on the opposite side of the pliers from the body. When you take apart the circuit on the breadboard, we suggest you don't straighten out the leads of the various components. After all, you might need some of the components with the same bends next time you build a circuit, and if you bend and straighten a lead too many times, the lead will eventually snap.

Something else to remember is that it often won't be possible to build a circuit just by plugging components into the breadboard because component leads only stretch so far, however you bend them. But connections between components don't have to rely on the breadboard's internal connections alone because you can also make connections using patch leads. Patch leads are also used, of course, for making connections between the circuit on the breadboard and external equipment, most commonly a Raspberry Pi or Arduino or a similar single board computer.



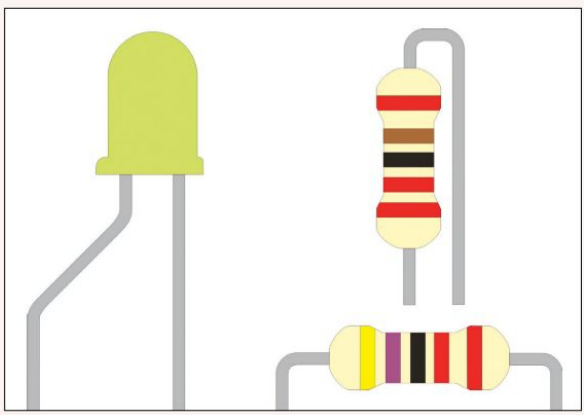
Some two-lead components have to be connected in a particular way. The identification of the leads on the physical component and the circuit diagram symbol are shown for LEDs (left), diodes (middle) and polarised capacitors (right).

Before you try out your circuit, and because it's possible to burn out a component if you get it wrong, be sure to check your circuit carefully. Even if the circuit passes your visual check, if it doesn't seem to work when you try it out, remove the power immediately before trying to figure out what's wrong. At this point, in addition to doing another visual check, if you have some simple test equipment, and especially a test meter, it could simplify the fault-finding process – see the box for our guide to test equipment (*facing page*).

When you start to build more complicated circuits, software can help you design the breadboard layout. We plan to provide some instruction on using this sort of software in the near future. LXF

QUICK TIP

We don't recommend it, but complicated circuits have been patched together on breadboards. You might be interested to learn, for example, that there are reports of a graphics card being prototyped this way, even though it looks rather like the aftermath of an explosion in a spaghetti factory.



You'll probably have to bend some component leads. An LED (left) will fit into two holes if they're close enough, but for more distant holes you'll need to bend one or both leads. You'll always need to bend at least one lead on a resistor (right) and there are two ways to do that.

BACKUPS

Back up and clone your Raspberry Pi

From disk cloning to backing up specific directories and partitions, keeping your Pi backed up saves time and heartache, says **Christian Cawley**.

OUR EXPERT

Christian Cawley is known as The Gadget Monkey because he eats a banana for breakfast, microwave chips for his lunch, and a Raspberry Pi for dinner.

While a flexible, affordable and much-loved piece of computing hardware, the Raspberry Pi isn't without its problems. One particular bugbear is the device's reliance on flash storage – potentially unreliable SD and microSD cards that are prone to data corruption following an unscheduled shutdown or restart.

One way to avoid this is to ensure that the Pi is shut down carefully. But it also pays to have a backup of your data, just in case the worst happens. Raspberry Pi has several options for backup, from cloning the microSD card to remotely backing up over SSH.

Clone Wars

The ability to switch Raspberry Pi projects simply by swapping microSD cards means that if a card is found to be corrupt, you can easily flash an image of its contents to a replacement. Cloning the SD card is a smart option that enables you to keep a library of ready-to-use Raspberry Pi images on your PC's hard drive, ready for deployment.

Just think of the time you can save. All your favourite software installed just once. Wi-Fi networks already configured. SSH enabled, ready to use. Where once you would reinstall and reinstall, changing numerous settings to get things just right, cloning the SD card means that you can customise Raspberry Pi OS, get things working how you like them, and then make a copy to use over and over again.

This approach isn't without its shortcomings, however. For a start, there's the volume of storage space required for multiple images. You should also consider the amount of time required for the initial image creation. Simply having a clone of a fresh Raspberry Pi OS install is pointless, because that's essentially what you can download from **raspberrypi.org**. Instead, the image should be the underlying operating system together with the key software that you want to use, along with the data you've saved to the device.

Some challenges might be faced with cloning your Raspberry Pi. For example, a 16GB disk image can't be cloned and then written to an 8GB SD card. However, you may also run into problems cloning a 16GB image to another 16GB SD card, because of differences in quality-control standards.

There may also be some difficulty booting a cloned disk image on a different system. For example, a backup that's been created on a Raspberry Pi 4 will almost certainly not work on the Raspberry Pi Zero. However, if you keep in mind basic issues of compatibility and file size, then cloning your Raspberry Pi for back-up purposes should be plain sailing. Cloning the SD card can be done on a desktop PC or on a Raspberry Pi.

Rather than fiddle around with the dd command in the terminal, you can clone Raspberry Pi SD cards with Etcher.

The clone heads

To create a clone of your Pi's SD card on a desktop computer, all you need is the cross-platform flashing tool, *balenaEtcher* (**http://bit.ly/LXF287-etcher**). Download and install this on your desktop system and then insert the Pi's microSD card into the PC's reader.

You should also have a second microSD card connected ready to write to. Failing this, a USB flash stick. Writing to a system drive isn't possible without first creating a dedicated partition.

Click Clone drive and select a Source. Click Select, then Select target to choose the target drive. This should be the second microSD card. Once again, click Select to confirm, then Flash to commence. Wait while the software writes and verifies the data. You'll notice that it's possible to flash the data to multiple devices if you need to. You can also clone the Pi's SD card directly to your HDD. This is useful if you only have one SD card but plan to run multiple Raspberry Pi projects.

With the SD card inserted in your desktop computer, open the terminal, and enter the following:

```
sudo fdisk –l
```

Identify the SD card and make a note of its name. This can be tricky, but it should be obvious from the size of the SD card. It will typically be sda or sdb followed by a number.

Next, use the `dd` command to create a clone. You can name the target file however you like, but it should have the .img extension:

```
sudo dd if=/dev/sdb of=~/raspberry_pi_clone.img
```

Wait as the process completes. When you're ready to write the disk image to a new microSD card, insert a fresh card and enter the following:

```
sudo dd if=~/raspberry_pi_clone.img of=/dev/sdb
```

You might also use the *Raspberry Pi Imager* tool, the mouse-driven Raspberry Pi disk imaging software available from **raspberrypi.org** (**http://bit.ly/LXF287-piimager**) for Ubuntu, macOS, and Windows. After installing the program, click Choose OS, then scroll to Use Custom and browse for the cloned .img file before writing it to a fresh SD card.

Clone your Home

For a simpler approach that's less time-intensive, archive the Pi's Home folder. This method enables you to browse and then restore directories and files to a reimaged Raspberry Pi as and when required. Open a terminal and enter:

```
cd /home/
sudo tar czf pi_home.tar.gz pi
```

With the archive created, copy **pi_home.tar.gz** to another device for safekeeping. The next time you need to reimage a microSD card, once complete simply unzip the **/home/** directory and copy the files you need to the Raspberry Pi.

Network backup

If you prefer using the Raspberry Pi remotely and have SSH enabled, then it's possible to back up the Pi across your network.

This isn't ideal for backing up the entire SD card, however. Some files that are in use while the Raspberry Pi is running won't be correctly cloned, leaving you with a disk image that may not boot correctly. The best option with this network approach is to back up only vital data, such as the home directory.

Begin by starting an SSH session and connecting to the Raspberry Pi. Then enter the command:

```
ssh pi@[IP_ADDRESS] "sudo dd if=/dev/mmcblk0 bs=4M | gzip –" | dd of=~/Desktop/[BACKUP_NAME].gz
```

Be sure to replace `[IP_ADDRESS]` and `[BACKUP_NAME]` as appropriate.

Wait while the backup completes. Of course, the backup cannot be restored across the network. The Pi would lose connectivity part-way through. To restore the disk image, you would need to insert the blank SD card into your computer's card reader and then use the `dd of` command:

```
gzip -dc ~/Desktop/[BACKUP_ NAME].gz | sudo dd of=/dev/rdisk1 bs=4M conv=noerror,sync
```

Remote backups using SSH will save time and wear and tear on the Pi's SD card. It's also a useful option if you run multiple Raspberry Pis.

The Raspberry Pi Imager tool has various hidden features. Among them is the ability to write any IMG file to a microSD card.

When in clone...

Backing up your Raspberry Pi files through cloning can take up quite a bit of disk space, especially if you're making regular disk images. By default, the disk images are the exact same size as the disks, rather than the data used. So, an 8GB cloned SD card would take up 8GB of hard disk space. Fortunately, the files can be compressed. All you need to do is pipe the output from *dd* to *gzip* to create a compressed .gzip file.

```
sudo dd bs=4M if=/dev/sdb | gzip > raspberry_pi_clone.img.gz
```

To restore:

```
gunzip --stdout raspberry_pi_clone.img.gz | sudo dd bs=4M of=/dev/sdb
```

The cloned file is smaller using this technique and will take up less space on your HDD. This, of course, means you can make even more Pi backups... LXF

QUICK TIP

Use the fastest rated microSD cards for your Pi, with A1 performance. This will ensure fast, reliable system cloning, which will help when it comes to writing a fresh disk image. Also, avoid unknown brands and stick to well-known names: SanDisk, Kingston and Samsung, say.

» JUST BACK UP THE CONFIG FILE

If your Raspberry Pi setup is unusual, perhaps due to the hardware you've connected to the device, but you don't feel the need to regularly back up, there is an alternative. Instead of backing up the full SD card, or even archiving the **/home** directory, why not simply make a copy of **config.txt**? Existing in place of a system BIOS, **config.txt** stores various system parameters required for the correct running of the Raspberry Pi. Every Raspberry Pi operating system features a **config.txt** file, which is read by the system's GPU as the device boots.

While the Raspberry Pi is running, **config.txt** can only be edited as root. You'll find it in the **/boot** directory as **/boot/config.txt**. With the microSD card inserted in other systems, **config.txt** can be edited using a standard text editor. A whole bunch of options can be stipulated in **config.txt**. It enables the use of options for every possibility, from RAM use to camera settings, display options, overclocking, GPIO, and much more. For example, if you used a Raspberry Pi Zero with a small display, you would need to make some changes to the default options in **config.txt**. By backing up the file after the changes are saved, you can save time after a reinstallation by simply replacing the default **config.txt**.

FYDE

Credit: https://github.com/fydeos

Turn a Raspberry Pi into a Chromebook

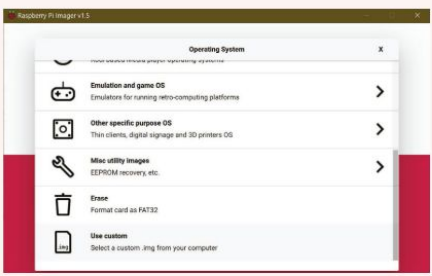
We're not saying **Ian Evenden** only likes the simple things in life, but we can't get him to return our Raspberry Pi for love or money...

OUR EXPERT

Ian Evenden is a freelance writer and editor, contributing to gaming, computing, science magazines, and books. He still disapproves of Oxford commas.

The beauty of single-board computers is that they can be switched between operating systems easily – it's just a case of swapping a microSD card over and switching it back on. The operating systems are easy to install too, flashed to a card from your PC with a user-friendly piece of software.

So we're going to swap in Chromium OS – an open-source version of the Chromebook OS – it's as minimal as things come without being a command line, because it's just the *Chrome* browser. But that doesn't mean it's limited. There's a full office suite in there, cloud storage, image editors, e-book readers and other tools that are available as web apps. With no flashy interfaces or 3D games to distract you, working like this is a low-powered way to get everyday computing tasks done.



The Raspberry Pi Imager is the ideal way to get your OS written.

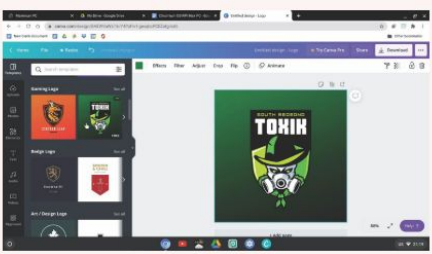
It's also quite responsive, which isn't something you can say for some of the heavier operating systems available for the Pi. We're using a Raspberry Pi 4 with 8GB of RAM here, but it will also run on a Pi 400, 3B or 3B+ model, so grab your Pi and get installing.

You'll need the *Raspberry Pi Imager* tool to flash the ChromeOS build to a blank microSD card (Chromebooks typically come with 16 or 32GB of local storage, but this version only gives you 4GB, no matter how large a card you use – there's a way around this, though). The tool runs on Ubuntu, Raspberry Pi OS, macOS or something called Windows, and can be found at **www.raspberrypi.org/software**. Once that's installed, head to **http://bit.ly/lxf278chrome** and download the Chromium OS image for your particular Pi model.

If you want to build the OS image yourself, head to the project's main Github page. You'll find plenty of discussion about that sort of thing, along with some known issues, one of which is important and we'll come to later. In the *Imager*, choose a custom OS (see screenshot, *left*), then point it toward the IMG file you downloaded from Github. Choose the card you want to use, and set it working.

We like to boot it, boot it...

Put the newly flashed card into your Pi's slot, and switch it on. The first boot can take a little while, but you'll soon be taken through the basic setup routines for the OS,

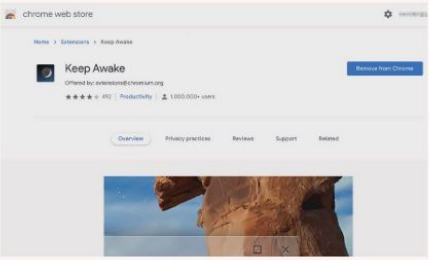


Canva is just one of a range of extensions available for installation.

» SILLY SETTINGS

You can connect a mouse or touchpad to the Pi. Touchpad use is almost exactly the same as in Windows. Tapping with two fingers gets you the right-click menu, and scrolling involves dragging two fingers down the pad. If you want to reverse this, there's an option in Settings>Mouse, from where you can also adjust the pointer speed.

Pressing the Windows key brings up the Launcher, from where you can choose between recent applications or search for a new one, and there's a circle button in the bottom-left corner that does the same thing. Settings>Keyboard contains a great many options for mapping a Windows keyboard to the Chromebook standard, enabling you to change the behaviour of Ctrl, Alt, Esc, Backspace and more. Finally, Google Assistant can be activated, but is turned off by default, because Raspberry Pis don't have a built-in mic. Add one along with speakers and you can turn your Pi into a digital assistant.



Use the Keep Awake extension to stop the Pi going into coma mode.

including connecting to Wi-Fi. Chromium OS requires a Google account to sign in with, and if you're a *Chrome* browser user you get all your bookmarks and extensions synced across. We often find that there's a period of non-responsiveness on the password screen, but it usually clears after 30 seconds so you can log in. It's a minimalist UI, with a Mac-like dock at the bottom and the browser taking up the rest of the screen – but what more do you need?

Remember that important issue we told you about? The first thing you should do is open the *Chrome* Web Store and install the Keep Awake extension (see screenshot, *above*). When that's done, set it to Sunny Mode, which stops your Pi from going to sleep. The developers of this Chromium fork haven't quite worked out the Pi's sleep system, and if it goes to sleep it won't wake up again. Or if it does, the keyboard input doesn't work, and you can't type your password.

Browse the Web Store and see if there's anything else you need to install. The *Chrome* Web Store is packed with extensions, though not all of them are compatible. The image editor *Polarr* wasn't when we checked, but *Photopea*, which is reminiscent of *Photoshop*, runs fine, albeit with a large gray bar down the right-hand side. Graphic design program *Canva* works too.

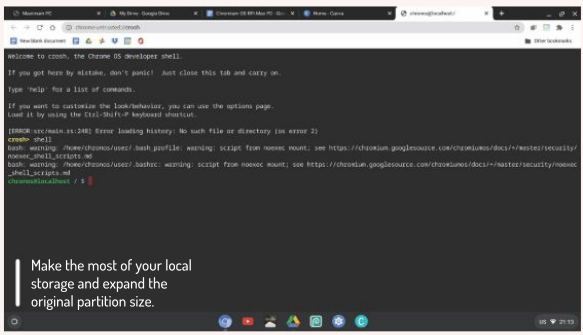
Working with *Google Drive* and *Docs* is exactly the same as in any other *Chrome* browser. Just open them in a new tab, and you'll get the familiar experience. A Chromebox is very much an online entity, with little local storage, so you'll need to get used to storing documents in the cloud – especially as there's such a small amount of local storage included. If you find you need to increase the amount of storage available to the Chromebox, you can expand the filesystem to the full size of the card through the developer shell. Press Ctrl+Alt+T from the desktop, and a browser tab will open containing *crosh*. Type `Shell`, and you'll be in *Bash* (see screnshot, *right*). Type `sudo rm /mnt/stateful_partition/.autoexpanded` and reboot the OS.

Accessing media

You'd expect browser-based streaming services to be accessible through Chromium OS, but they're hit and miss. Being a Google project, YouTube is prominent, and we were able to watch a SpaceX capsule arrive at the International Space Station live and without jerkiness. Attempting to watch hilarious Eddie Murphy movie *Coming to America* on Amazon Prime, however, failed on every attempt with an unhelpful error. *The Matrix Revolutions* failed in the same way, but this felt like less of a loss. This was due to the Widevine DRM system for *Chrome* not being supported in Chromium OS. There's a way around it, which is linked to from the Issues page of the Chromium OS for Raspberry Pi Github.

The Settings tool is the blue cogwheel icon in the dock, and opens into a comprehensive set of options for your device. It's where you go to connect to a new Wi-Fi network, add a new user to the Chromebox, or pair a new Bluetooth device, as well as adjust the screen resolution and scaling. On a 4K display, we found things much easier to use if we boosted it up to 175 per cent of the usual scale. Settings also contains options for personalisation (see the box, *below*) and, under the Apps heading, you can choose which tools are pinned in the dock. Some, like *Photopea*, open in their own windows, while others, such as *Google Drive* and *Canva*, open in a browser tab.

Click the grey lozenge at the bottom-right corner, and you'll get a pop-up menu with quick settings, such as turning Bluetooth on and off, changing the volume, locking the screen, and turning the system off. Being a Raspberry Pi, the unit doesn't have the ability to switch itself off, so you'll have to cut the power yourself once the OS has finished shutting down. Most Raspberry Pi kits come with a handy switch on the power supply lead for this purpose. LXF



» CUSTOMISED WALLPAPER

Changing desktop wallpapers in Chromium OS gets its own tool, accessible from Settings>Personalization. It's filled with desktop backgrounds, and a few have the option to refresh daily. There are some beautiful options available, but sadly they don't appear in the Files tool. Right-clicking the desktop brings up a menu, with only three options: to show or hide the dock (actually called the "shelf"); to position the shelf on the side of the screen; and change the wallpaper, which brings up the Wallpaper tool. Image files saved to the Downloads folder can be set as a wallpaper by right-clicking them and choosing the appropriate option. Beyond this, there isn't much personalisation , apart from changing the picture that appears by your account name.

WEB APP SECURITY

Former Dark lord of network operations **Tim Armstrong** teaches the mystical arts of systems, security and keeping your stuff secret and safe.

Another day, another high-profile company gets hacked, and more customer data falls into the wrong hands. The company-du-jour will say how its security was up to all the latest standards, and how this was clearly a state-level attack. Yet upon detailed post-mortem it'll usually be revealed that some web-crawler got lucky and started pulling data from an unprotected endpoint.

It's a common meme among programmers that businesses never have time to invest in security before an attack. Once an attack has taken place, the managers are all at your desk asking how you let this happen. Forgetting entirely that you've been calling for more time to focus on refactoring and security for several months, now only to be ignored.

As sad as this reality is, it's not really the managers' fault. The pay-off of a new feature is tangible: it has a fixed start and end; it links to a business objective; it is, for lack of a better word, quantifiable. Security and refactoring, however, is quite the opposite… or is it? If you can convey the risks and benefits without sounding like a broken record then you can change your culture for the better.

This series will empower you with the tools to do just that and make security part of your culture. Kicking things off we'll discuss some fundamentals and motivation, which will be followed with detailed implementation tutorials in future issues.

Illustration: Kym Winters

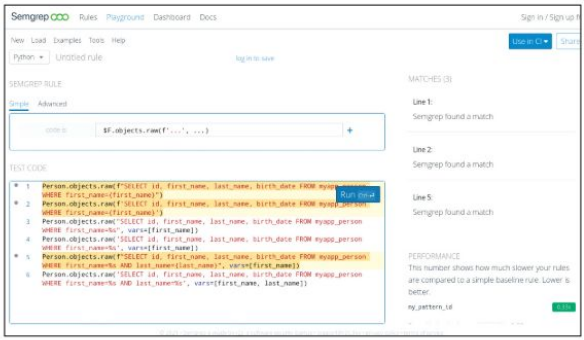
Security breaches are frequently more than just embarrassing, and can range from mild inconvenience to life-destroying for those affected. If a company leaks its customer's credit card details, the odds are that eventually you'll get that money back, either through the issuing bank's insurance scheme or through charge-backs against the fraudulent transactions. While the charge-backs can be difficult for any third-party businesses that get inadvertently caught up in this situation (generally through being the hacker's unwilling or unknowing cash converter), it's commonly an equally recoverable situation.

If, however, the leak contains copies of bank statements, a social security number, home address, and/or mother's maiden name, then the breach can be much more significant. It can lead to massive identity theft – devastating for those customers hit in the aftermath of the breach. As a result, the regulators, governments and class action lawsuits that follow frequently seek proportionate punitive damages against the company that suffered the breach. Which, unless it's a Fortune 500 company, can be significant enough that the company goes into voluntary administration and ultimately getting shuttered for good not long after.

Start by standing still

Your first step should be static analysis of your code, but what is it and why is it important to security? Essentially, it's the process of evaluating our code against a set of rules. The specifics of these rules, however, is where it gets interesting.

At face value, it might seem far-fetched that a suite of rules, which enforce things like code style and formatting, can protect us from various code-injection attacks, denial of service attacks and other vulnerabilities. Static analysis isn't limited to style and formatting compliance, though. It can be surprising but detecting issues like SQL injections, hard-coded passwords and unhandled exceptions are remarkably well-covered by free and open-source tools maintained by some of the best security researchers in the world. Using these tools is like having a team four to five times larger helping you get through your code review, pointing out any mistakes or vulnerabilities you might have introduced before you even post your review request. All while simultaneously checking your libraries are up to date and that you haven't introduced any known vulnerable code in your app.



Hop in the Semgrep playground and give it a whirl!

Static analysis comes in a wide variety of flavours, some of which you've probably come across before (common Python examples include *Black*, *Dependabot* and *PyLint*) and some of which you might not have (again from a Python perspective: *Prospecter*, *McCabe*, *Dodgy*, *Bandit* and *Horusec*). These fit loosely in a few categories: Linters, Complexity Analysers, Dependency/Container Analysers and Semantic Analysers. A number of these tools blur the lines a little, though.

Let's take *Semgrep* for example. *Semgrep* an open-source tool that supports a range of languages. It's designed to make it easy to write your own rules in essentially the same syntax as the language you're writing the rule for. This enables you to efficiently define rules that, for example, could identify the use of f-strings in SQL statements (for Django object raw queries this would be `$F.objects.raw(f'...',...)`). Simple rules like this are easy to write and, as you can see, very powerful.

Linting? Like the fluff in the dryer?

Pretty much, yeah, linting borrows its name from the first real tool of its kind – *Lint* – written by Stephen C. Johnson for C over 42 years ago. It was named after the fluff that tends to form little bug-like shapes on new coats and clog up the exhaust filters of your tumble dryer. The primary goal of *Lint* was to identify and help you remove this "fluff" and unnecessary complexity from code, thereby preventing bugs before they happen. The by-product (commonly mistaken to be the primary goal) of this tends to be cleaner, easier-to-read code.

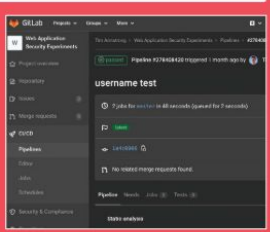
The base set of linting rules implemented in

» THE CONSTANT GARDENER

GitLab's CI solution and integrations are significantly above the competition. It is open source, easy to use, and provides 400 minutes for free in its hosted solution.

Gitlab CI consists of two elements: stages and jobs. A stage is like a tag that groups jobs into parallel executions. Each stage is performed sequentially, and if any of the jobs in that stage fail then the pipeline will stop progressing. Getting started with Gitlab CI configs can be as simple as this (we'll go into more detail on this in later tutorials):

```
stages:
        - prebuild
lint:
  stage: prebuild
  image:
    name: "ckleemann/prospector:latest"
    entrypoint: [""]
  before_script:
    - pip install -r requirements.txt
  script:
    - prospector ./src
```



Gitlab CI's run summaries are helpful to quickly check on your pipeline.

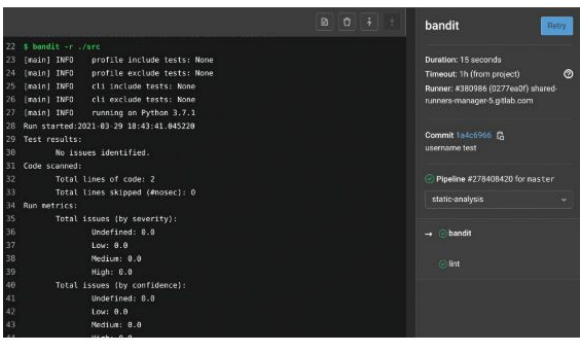
essentially any good linting tool includes style/formatting conformity, undeclared variables, unused variables, unreachable code, syntax errors and uncaught exceptions. Now you might be thinking, "Okay but my IDE does all that", and you'd be right. Practically every IDE available today has a linter built-in. However, there are a few key problems with relying exclusively on the one in your IDE:

Visibility – if it's in the Continuous Integration/Continuous Deployment (CI/CD) pipeline then your reviewer just needs to glance to see that you've not introduced new problems.

Commonality – is the entire team using your IDE? Do all of the IDEs in use the same linter and rule-set?

Extensibility – if it's in the CI/CD then your team can extend it with new rules that you all agree on.

In the case of Python, we have an ever-growing hub of style and formatting standards written by some of the most seasoned developers in the world by the way of PEPs (Python Enhancement Proposals), perhaps the most famous of which being PEP8. Tools like *PyLint* are kept up-to-date with all the accepted standards and are ready to help you keep your code easy to read and maintain. *PyLint*, however, only scratches the surface compared to a tool like *Prospector*, which wraps not only *PyLint*, but also tools like *McCabe*, *Dodgy* and many more into a singular utility that presents all of the findings in a common structure for easy parsing and tracking. It provides a singular tool that covers the complete linting landscape.



Bandit is a great Static Analysis Security Testing utility, identifying hardcoded passwords, common injection vulnerabilities and much more.

Dynamic analysis

So if static analysis is so good, what is dynamic analysis and why do you need it? Dynamic analysis is the general term used to group any testing that requires your software to run in order to do the tests. These include system tests, vulnerability tests, memory error tests, performance, concurrency tests and pen-tests.

Even if, in theory, static analysis could identify 100 per cent of all security issues and bugs, this would only be able to cover the code and libraries. There are always ways to write bad code that obeys all the rules and passes all the tests, and there are many more ways to deploy good code poorly. Dynamic analysis helps us detect a significant proportion of these issues. It can help us find bugs like Cookie-slack (where pages that should be hidden behind a login can be accessed even if there's no cookie, or it's expired) and Slowloris (a type of attack that with very little effort or traffic can effectively deny access to your app) vulnerabilities, along with configuration issues such as supporting insecure SSL/TLS versions. Dynamic analysis is, of course, wider than just these DAST (dynamic analysis security testing) focused issues. Some of the most notable dynamic analysis solutions that are security adjacent are also important from a wider application stability standpoint, including memory-error detection, load-testing and system testing.

The use of memory-error detection systems like Valgrind in C/C++ is fairly common (in part down to the ease of making memory leaks in these languages) yet in the communities of modern languages like Python, Rust and Go, this kind of tooling is largely ignored as heavy reliance on the language to "just handle all of that" is prevalent. This frequently leads to inefficient constructs and overlooked memory leaks that are only apparent under heavy load, which is the last moment you want to discover that you have a memory leak.

The loss of profit due to a crash during CyberMonday is a great way to lose sales revenue and increase costs. So, how is that security adjacent? Well, if a malicious actor discovers the ability to crash your systems by opening a barely noticeable amount of traffic over an extended period, then they could either increase your hosting costs, or take down your site during a big sales event, or worse (both at the same time).

However, the sad reality is that while the industry as a whole is slowly getting better at incorporating static analysis in their CI/CD pipelines, this shift-left of quality assurance doesn't seem to extend to the realm of dynamic analysis; where the farthest most have gotten on dynamic analysis seems to be their semi-annual pen-test. This is what security experts refer to as testing in production. The unfortunately common belief that "there are more interesting targets than us" leads to a false sense of security and over-reliance on the old fashioned semi-annual pen-test that leads to glaring

» MANAGE EXPECTATIONS

Building security analysis and code analysis into your CI/CD is one of the key means by which we can monitor the quality of your code and any risk exposure. It gives you insight into (and frequently quick wins against) some of the most common vulnerabilities and bugs.

The good news is that by adding a couple of small stages to the pipeline you can easily (and in some cases automatically) mitigate entire classes of attack, while also making cleaner code that's easier to read.

The even better news is that most languages and frameworks are already tooled-up and good to go, so you just need to grab the relevant Docker images and very quickly you can be up and running. Having said that, do remember that it's better to over-estimate time when learning new systems so that you have a buffer if something comes up. If it takes less time than you've requested then you might get more flexibility in the future.

If you come prepared with a demo from an open-source project that has implemented this well, then static analysis is a pretty easy sell to get things rolling.

security holes going unnoticed for months. In many cases, this leads to these pen-tests consisting of little more than a consultant running an automated vulnerability scanner like *Zed Attack Proxy* (*ZAP*), finding a large number of issues and not really testing further, because at that point you've already got an eight-page report to deal with.

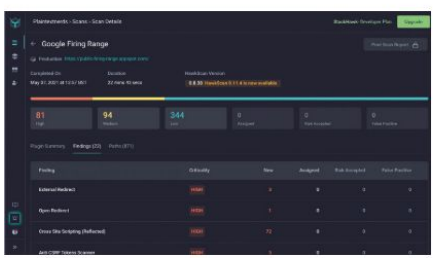
Scanning for vulnerabilities

Needless to say, testing for common vulnerabilities and exploits on the production estate is important. However, by the time your deployment reaches production, you should already be reasonably sure that it has no known vulnerabilities. So how do you go about that?

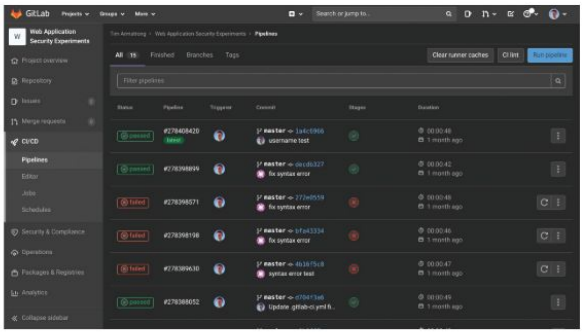
While there's a growing number of tools like *ZAP* that can be used to run a vulnerability analysis against your web app, there's very few that appear to be CI/CD ready out of the box. However, we're starting to see some entrants to this space, most notably *StackHawk*, which is free for open-source and individual developers and built around a tuned *ZAP* based solution.

One of the notable things about this particular option is that the Founder of *ZAP*, Simon Bennetts, joined the team at *StackHawk* in 2020 due to its dedication to Open Source and continued support of *ZAP* development. Now, it's possible to build your own solution based around *ZAP* if you wanted to keep your pipeline free of commercial tooling, but in the long run a self-build solution will probably end up costing you more. Of course, there are alternative commercial solutions, such as *Probely*, but integrating these platforms into your CI/CD commonly entails exposing public endpoints to your staging environments, which is its own bundle of potential issues that are best summarised as undesirable. With both *StackHawk* and *ZAP*, this exposure isn't necessary as both scanners are capable of being run be run in a local environment or from a container.

Integrating vulnerability scanning into your CI/CD means you'll be able to check your web app for a lot of the same problems that hackers are looking to exploit, which results in automated drive-by attacks having a significantly harder time finding holes in your defences. To put it simply, automated vulnerability scanning is to Dynamic analysis what linting is to Static analysis. It's the bare minimum that you should be doing to test your web app. However, unlike linting where it might be acceptable to let your app go live with some minor style conformity issues (like a missing docstring on a function), there isn't really an excuse for a well-designed web app to be failing a vulnerability test, because this means that you've got a known issue that could expose either your servers or your customers to malicious activity or data theft.



The StackHawk user interface saves a lot of time when discussing priorities and triage.



Seeing the change over time as you work on your code can be helpful when planning sprints, because you can evaluate how many iterations it takes to get your code passing.

Building secure software

Building secure software starts with understanding the nature of the attacks you're likely to face. If you have PII (personally identifiable information) or PCI (payment card information) then data is often the primary target of an attack. However, if you host a blog, or other (semi-) static content, then it's likely that an attacker would want to inject code into your site with the hopes of compromising, defrauding your readers, or compromising your hosting.

Doing a threat-modelling exercise as part of feature planning can reduce the time to market, by folding in the necessary protections to the design from the start. To draw parallels to the electronics industry, adding shielding at the end of a production line to protect your device from electromagnetic interference is less effective and more costly than simply designing your board layout to include sufficient signal trace spacing and physical distance between "noisy" components (such as a switching PSU) and your sensitive components (such as a sensor network). By including threat modelling in the design phase we can catch the cases that haven't yet reached the level of intuition.

Building both Static Analysis and Dynamic Analysis into your CI/CD pipelines will give your product managers the numbers and metrics with which they can evaluate the trade-off of new features vs refactoring. But there's also a direct benefit to you and your team in that you're able to recognise potential technical debt introduced by your Merge Request (Pull Request if you're in the Github eco-system). Over time both of these things will lead to improvements to the code base and enhance the project's quality, security and reliability.

Web app security isn't just a slide on a shareholder presentation or pitch deck – it's a culture that needs to be fostered in your organisation. The upcoming tutorials in this series aim to show that, for the most part, it's not something you even need permission to kick-off, as there are many big wins that you can make over a period as short as a lunch break if it comes to it. LXF

PHOTOREC

Recovering deleted files

Shashank Sharma knows from experience that recovering deleted files is far easier than growing hair over a bald patch.

OUR EXPERT

Shashank Sharma is a trial lawyer in Delhi and an avid Arch user. He's always on the hunt for geeky memorabilia..

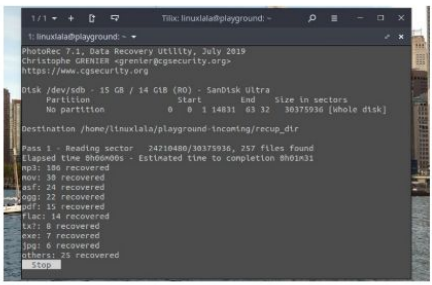
Hardware failure and a careless user feeling adventurous with powerful utilities such as `dd` and `fdisk` can lead to data loss. Not only that, sometimes spring cleaning a partition or directory can also lead to accidentally deleting some useful files. Should that happen, there's no reason to despair. With the *PhotoRec* utility, you can easily recover a variety of files, be it documents, images, music, archives and so on.

Developed by CGSecurity and released under the GPL, *PhotoRec* is distributed as a companion utility of *Testdisk*, which can be used to recover and restore partitions. You can use either of these tools to recover files, but each has a job that it's best suited for.

Testdisk is best suited for recovering lost partitions, Whether this is on account of you overwriting or deleting a partition, or a partition becoming unreadable for any reason, *Testdisk* can help you restore the partition, or at the very least, recover data from it.

But if all you're interested in is recovering deleted files from a partition, hard disk or even a USB drive, you can use *PhotoRec*. Although initially designed to only recover image files (hence the name), *PhotoRec* can be used to recover just about any manner of file. Even better, *PhotoRec* works by ignoring the underlying filesystem on the specified partition, disk or USB drive. Instead, it focuses on the unique signatures left by the different file types to identify them. This is why *PhotoRec* can work with FAT, NTFS, ext3, ext4 and other partition types. In contrast, *Testdisk* only supports a limited number of filesystems.

The greatest drawback of *PhotoRec* – if any tool that can seemingly pull deleted files out of the digital ether can have a drawback – is that it doesn't retain the original filenames. This means that recovered files all sport a gibberish alpha-numeric name. If this is a deal-breaker for you, consider using *Testdisk* first to recover your lost files.



You can stop a recovery operation at any time, if you want, and even exit PhotoRec. But when you next start PhotoRec, it will ask if you wish to resume the previous session.

Understanding the basics

PhotoRec isn't distributed individually, but as part of the *Testdisk* utility. You can install it on most distribution using the software management tool because it's part of the repositories of most popular distros such as Fedora, Mind, Debian and Ubuntu. The command `sudo apt install testdisk` will install it on Debian/Ubuntu and derivative distributions. If you're running Fedora, or its ilk, use the `sudo dnf install testdisk` command.

Before you start using *PhotoRec*, it's important to understand how a filesystem handles deleted files. When you delete a file, it isn't immediately zapped into oblivion. Instead, the file system merely marks the file as deleted and assigns the space the file occupied as

» OTHER RECOVERY SOLUTIONS

Testdisk and *Photorec* have long been part of almost all data recovery CDs and toolkits because of their effectiveness. But they aren't the only tools that can help you recover deleted files, or extract useful data out of formatted or corrupted disks or partitions.

Although it hasn't seen a new update in some time, *Extundelete* is quite useful in recovering files, especially from ext3 and ext4 partitions. It works best when you already know the exact path of the file you wish to recover, such as with the `sudo extundelete /dev/sda7 -restore-file ~/Documets/todo.txt` command. You can alternatively use the `sudo extundelete /dev/sda7 -restore-all` command to recover all files from the specified partition.

Scalpel is another file carver, just like *PhotoRec*, and so can work independent of the underlying filesystem. This means that you can use it to recover files from across different filesystems such as FAT, NTFS, ext3 or ext4. By default, *Scalpel* isn't configured to look for any file types, so the first step is to define the file formats you wish to look for in the **/etc/scalpel/scalpel.conf** directory. Just like *PhotoRec*, you can use *Scalpel* to recover different files such as PDF or images.

You'll find both these tools in the software repositories of most popular distributions such as Ubuntu and Fedora, so installation is as easy as using your favourite software management tool.

available for use. This means that until that space is taken up by another file, the original file is still there and can be retrieved using specialised data recovery tools.

It's for this reason why you should ideally stop using the system as soon as you realise that you've deleted an important file, because you'll minimise the risk of its space being occupied. Although both *PhotoRec* and *Testdisk* can recover files from deleted and then overwritten partitions, it decreases the chances of recovering all the files.

Get recoverin'

Before proceeding further, you should create a directory where you wish to store the files recovered by *PhotoRec*. This directory should not be in the same partition or device from which you are attempting to recover files.

Now open the terminal and fire up PhotoRec with the `sudo photorec` command. After you enter the password, you'll be presented with the welcome screen that lists all the partitions as well as connected disks and devices:

```
PhotoRec 7.1, Data Recovery Utility, July 2019
Christophe GRENIER <grenier@cgsecurity.org>
https://www.cgsecurity.org
PhotoRec is free software, and
comes with ABSOLUTELY NO WARRANTY.

Select a media (use Arrow keys, then press Enter):
>Disk /dev/sda - 500 GB / 465 GiB (RO) - WDC
WD5000LPVX-08V0TT5
 Disk /dev/sdb - 15 GB / 14 GiB (RO) - SanDisk Ultra
 Disk /dev/loop0 - 148 MB / 141 MiB (RO)
 Disk /dev/loop1 - 145 MB / 138 MiB (RO)
>[Previous]  [ Next ]  [ Proceed ]  [ Quit ]

Note:
Disk capacity must be correctly detected for a
successful recovery.
If a disk listed above has an incorrect size, check HD
jumper settings and BIOS detection, and install the
latest OS patches and disk drivers.
```

You can use the up/down arrow keys to select the media device from which you wish to recover the deleted files. Depending on your system configuration, you might have to scroll through a far larger list of devices than this example. After selecting the device, use the right/left arrow keys to select the Proceed button at the bottom of the screen, and hit Enter.

If your selected device comprises several partitions, *PhotoRec* will display all the partitions and enable you to choose the partition that was home to the deleted file.

You're then asked to specify the filesystem type where the file was stored. Select the ext2/ext3 option if the file as stored in an ext2, ext3 or ext4 filesystem. Alternatively, you can select the Other option if the partition was FAT, NTFS, HFS+ or ReiserFS, for example.

Next you have to decide whether the search for deleted files should be limited to only the free space on the partition or device or whole of it. The final step is to select the directory where you want *PhotoRec* to place the recovered files. After selecting the directory, press C, and *PhotoRec* will start hunting for deleted files.

Depending on the size of the partition or device, *PhotoRec* can take quite a while to perform the complete search for files. Use this time to make yourself a fresh brew.

If you don't feel the need for hot refreshments, you can help speed things up by restricting the search to specific file types, such as pdf, jpg or mp3. After selecting the device or partition from which you wish to recover deleted files, use the right/left arrow keys to select the File Opt button and press Enter.

By default, *PhotoRec* will search for all filetypes. Press S to deselect all the file types, and use the arrow keys to scroll through the list of file types. Press the spacebar to select all the file formats you wish to search for.

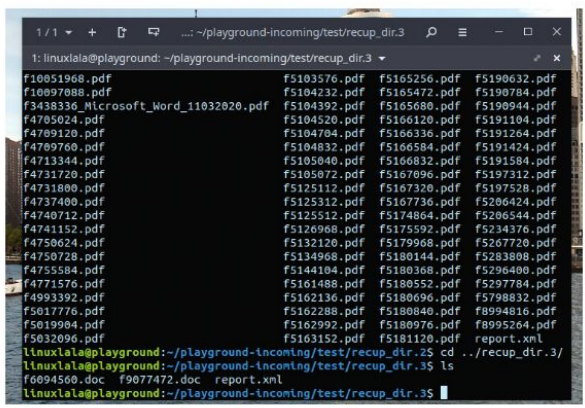
Making sense of recovered files

PhotoRec creates several directories such as **recup_dir.1** or **recup_dir.2** within the specified destination directory for recovered files. You'll find all the recovered files within these directories, but *PhotoRec* makes no attempt at sorting the different files based on their name or file formats.

You can, however, resort to some quick CLI-fu to organise the recovered files. First, create directories for different file formats, such as mp3 or jpg. Next, use the `mv` command to move files of a specific format to the relevant directory:

```
$ mv ~/miracles-happen/recup_dir.*/*.jpg ~/miracle-
happens/images/
```

You can similarly sort media and office files into separate directories.



PhotoRec will keep you informed about the recovered files. When done, you'll be informed of the total number of files that have been brought back from the brink.

QUICK TIP

There's no shortcut to return to the home or welcome screen, but you can press q on any screen inside PhotoRec to go back to the previous screen.

DESKREEN

Credit: www.deskreen.com

Use your mobile as a secondary display

Nick Peers reveals how to display your desktop on your phone, tablet or laptop's WebRTC-capable browser.

OUR EXPERT

Nick Peers loves smart solutions like this, which enable you to find additional uses for the technology you own – even those supposedly obsolete devices gathering dust in the corner.

Not everyone has the desk space – never mind the finances – to afford the luxury of a dual-monitor setup. Adding a second display can, however, revolutionise the way you use your PC.

Deskreen is a free application that provides a clever workaround to this problem, one that makes it possible for you to view multiple windows at once across additional displays without spending anything. Simply put, it enables you to convert other devices, including phones, tablets, laptops, and even smart TVs, into a secondary display for your computing tasks.

This display can easily mirror your entire desktop so that you can monitor your PC remotely, or you can use it to view a specific program's content alongside your main display. Armed with the right know-how (see the box, *below*), you can even use it to extend your desktop for even greater flexibility.

Deskreen makes it possible to view selected desktop applications (or your entire desktop) via your tablet or phone.

Deskreen transmits your display – both X and Wayland windowing systems are supported – to your mobile devices over your local (wired or wireless) network using the WebRTC protocol. Most of the heavy lifting is done by your PC, so even older phones and tablets should work with it.

The key requirement for your mobile is that it has a web browser that supports WebRTC – on Android, that means running *Chrome 28* or later, or *Firefox 22* or later. If you're hoping to use an old iPhone or iPad, it'll need to run iOS 11 or later.

You can download *Deskreen* as a .deb or AppImage. The latter, of course, means no installation is required – head over to **www.deskreen.com** and click your chosen download link. Download and double-click the .deb file to install it via Ubuntu Software, or right-click the downloaded AppImage file and choose Properties> Permissions tab to tick 'Allow executing file as a program' before running it for the first time.

On first launch, the main *Deskreen* window will pop up and you'll be prompted to confirm your language and

» EXTEND YOUR DISPLAY WITH DESKREEN

By default, when you come to share your entire screen, *Deskreen* will enable you to mirror it – handy, for example, if you want to keep an eye on several running tasks without being tied to your desk. If your PC's graphics has multiple ports, you can go further by plugging a Virtual Display Adaptor into a spare HDMI or DisplayPort socket. These 'dummy' plugs cost as little as £6 on eBay – make sure you purchase the correct type for your ports.

Once connected, your PC is tricked into thinking a second monitor has been plugged in, and *Deskreen* will give you the option of viewing this dummy display for real through your connected device. Adding a secondary display like this means you can extend your desktop, enabling you to drag windows from your monitor on to the second desktop to be viewed – and controlled via your mouse and keyboard – directly from your remote display.

The dummy plug option is the simplest way to extend your desktop, but if you're comfortable playing around with a bash script, try *Virtual Display Linux* (**https://github.com/dianariyanto/virtual-display-linux**). Nvidia users will need to try a different approach – follow the link at the bottom of the read-me file to discover a workaround using Nvidia's TwinView option.

Deskreen's developers are looking at adding Virtual Display support without the need for an adaptor.

choose between a light or dark theme. Make your choice and click Continue. The step-by-step guide (*below*) reveals how to make the initial connection between *Deskreen* and your remote browser.

Controlling the connection

Your browser will display the application window or desktop on your secondary display. If you're using it to display a single application window that's hidden on your current desktop then it's a 'dumb' terminal: you can only control it in the usual manner by bringing it to the fore on your primary display using your mouse or keyboard. If you've chosen to display your entire desktop, then it'll simply mirror what you see on-screen.

Above this are the *Deskreen* logo and several controls: freeze or suspend the connection using the pause button, click the settings button to flip the screen (for teleprompters) or adjust the display quality if your network bandwidth or device hardware is struggling.

Next to this is an option to view full-screen, which will remove all screen estate and maximise your remote display. While you can't interact directly with the screen you can pinch and zoom into and out of it to make things easier to read. Lastly, you'll also see a 'Default Video Player' option. This only works with your device's default browser – *Chrome* (Android) or *Safari* (iOS) – and is only required if you're using the flip-screen option.

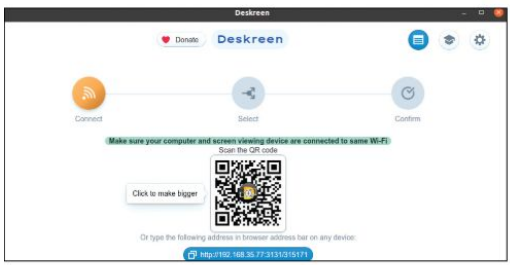
One of *Deskreen's* strengths is that you can use it with multiple remote displays. Once your first connection has been set up, just click 'Connect New Device' in the main *Deskreen* program to set up any subsequent connections. Remember, this will require more horsepower on the part of your PC, and you'll also be constrained by the speed of your network connection.

Once set up, use the Connected Devices button at the top to review which windows are being shared with which devices, and to end any (or all) connections. For most people, the best use of *Deskreen* with a smaller tablet or phone screen is as a dedicated display for specific utilities that require easy monitoring, but little actual interaction. For example, you could keep an eye on resource usage in System Monitor or monitor the progress of an encoding session in *Handbrake* while performing other tasks. LXF

QUICK TIP

Visit the Deskreen project's Github page (https://github.com/pavlobu/deskreen) for the latest releases and information, including links to its user forums for more help and advice if you need it.

CONNECT YOUR TABLET OR PHONE TO DESKREEN

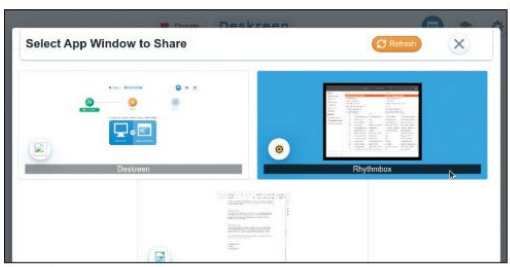


1 Scan in code or enter URL

Open *Deskreen*, which should display the Connect page, which will display a QR code and web address. You can either scan the QR code in using a QR code app on your tablet or mobile (in which case, it'll prompt you to click a link), or type the link directly into your mobile's browser. One of these options is significantly faster than the other.

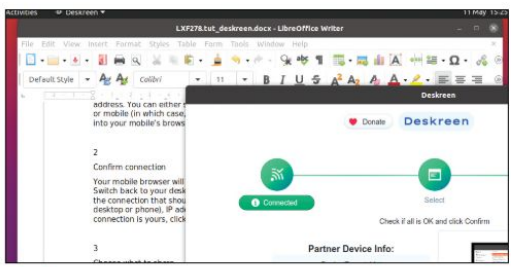
2 Confirm connection

Your mobile browser will tell you it's waiting for you to click 'Allow in Deskreen'. Switch back to your desktop, where you'll see it's displaying information about the connection that should tally with your device, namely its type (tablet, desktop or phone), IP address, browser and OS. Once you've verified the connection, click Allow.



3 Choose what to share

You'll be given the option of sharing your entire screen or a single application window. Choose the former and you'll be given the option of mirroring your display (see the box for details on using *Deskreen* to extend your desktop); otherwise select your chosen app window using the thumbnails as a guide.



4 Confirm and share

Review the partner device and screen or app being shared – click Confirm if you're happy, or 'No, I need to share other thing' if you chose the wrong option. After clicking Confirm, you'll see a confirmation message. Switch to your mobile where – after a short pause – the app or desktop should appear.

BACK ISSUES » MISSED ONE?

ISSUE 277
July 2021

Product code:
LXFDB0277

In the magazine
Discover what's new in the latest version of Ubuntu, grab a slice of network-attached storage, code a game in Scratch, emulate the Dragon 32 and set up your own streaming server with *Jellyfin*, Plus we look back at Prestel, the pre-internet data service!

DVD highlights
Ubuntu 21.04 (64-bit) and MX Linux 19.4 (32-bit).

ISSUE 276
June 2021

Product code:
LXFDB0276

In the magazine
We show you how to set up your own server for your photos, either at home or in the cloud. Learn how to emulate the Commodore PET, manage headless servers with *Cockpit* and render objects in *Blender*. Plus get coding and graph metrics in Python today!

DVD highlights
Manjaro 21 (64-bit) and Tails 4.17 (32-bit).

ISSUE 275
May 2021

Product code:
LXFDB0275

In the magazine
Get more from your Raspberry Pi with our great selection of projects. A parents' guide to coding, an introduction to 3D tool *Blender* and a tour of video conferencing tool *Jitsi Meet*. Plus code a first-person shooter, audio software reviewed and photo effects.

DVD highlights
Mageia 8 (64-bit) and AntiX 19.3 (32-bit).

ISSUE 274
April 2021

Product code:
LXFDB0274

In the magazine
Ramp up the security of your Linux system and turn it into a fortress. Have fun with a Raspberry Pi Pico, explore the best window managers, code a *Space Invaders*-style game in Python, emulate 486 PCs, set up a back-up system and more!

DVD highlights
Linux Mint 20.1 (64-bit) and Sparky 5.14 (32-bit).

ISSUE 273
March 2021

Product code:
LXFDB0273

In the magazine
Machines within machines are the order of the day as we show you how to better run a VM. Discover how to construct a mind map, emulate a BBC Micro and we explain Active Directory. But it's not all work: we review five game engines, connect a gamepad to a Pi and explore chess engines!

DVD highlights (64-bit only)
Manjaro 20.2 and Gecko Linux 152.

ISSUE 272
February 2021

Product code:
LXFDB0272

In the magazine
Enjoy a fresh FOSS start with the latest version of Ubuntu, plus encode video more efficiently, remake *Angry Birds* in Python, emulate the Classic Apple Mac, transform your command line life, and get into astronomy with our review of star-gazing software.

DVD highlights
Ubuntu 20.04 LTS (64-bit) and Bodhi Linux 5.1 (32- and 64-bit).

To order, visit www.magazinesdirect.com

Select Single Issues from the tab menu, then select Linux Format.

Or call the back issues hotline on **0330 333 1113**
or **+44 (0)330 333 1113** for overseas orders.

Quote the Product code shown above and have your credit or debit card details ready

BLENDER

Credit: www.blender.org

Part Four! Did you miss an earlier instalment? See page 66

Import and convert images to 3D models

Taking inspiration from images that started life outside of Blender, **Michael Reed** helps you turn them into shiny 3D objects.

OUR EXPERT

Michael Reed tells us that outside of playing with Linux,his life is so square it's practically a Blender default cube.

We're going to look at a couple of ways of taking externally sourced images and turning them into 3D objects inside *Blender*. First, we'll take a rough sketch as our starting point. Thanks to vector drawing application *Inkscape* (**https://inkscape.org**), we'll convert this into a set of curves and lines before importing it into *Blender*. From here, we can convert this flat shape into a solid 3D object. Then, most importantly, we can make it shiny! (*ooh, now I'm interested! – Ed*)

Secondly, we'll look at working from a reference photo to draw an outline of a real-world object. From here, we can use a lathe-like technique to turn this into a solid 3D shape to which we can add some thickness so that it has solid walls. From there, it's a cinch to add a texture to it, and give it a realistic-looking surface.

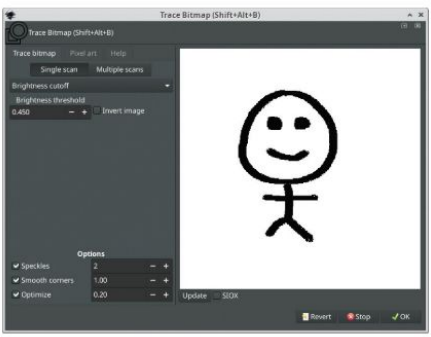
Sketch to logo

We're going to take a sketch and turn it into a metallic 3D object in *Blender*. For this, we'll assume that you've already got an image of a hand-drawn black and white 3D logo. Although this example starts with a sketch, there's no reason that you couldn't create something afresh in *Inkscape*, using its excellent drawing facilities. All you have to do to make this work is to generate a shape inside *Inkscape*, export it as an SVG file and then import it into *Blender*.

Inkscape is the ideal starting point for this type of work, and we'll use it to prepare the image ready for importing into *Blender*, and we'll assume you already have it installed. We're going to take advantage of its Scan Bitmap function to convert a bitmap image into a vector image, a collection of lines and curves, rather than an image made of pixels.

Select the JPG or PNG image, ideally a line drawing, that you want to turn into a 3D object, and drag it into *Inkscape*. Left-click on the image to select it. Open the Trace Bitmap dialog (Path>Trace Bitmap...). In most cases, the default options are sufficient, including the default method of Brightness Cutoff. Click the Update button to get an idea of what the trace will look like. Note that you can increase the size of the preview area by maximising the Trace Bitmap dialog itself.

Remember that we don't need an especially good-looking image. We just need a clean path with a smooth outline to work with. When you have everything how you want it, click OK and close the Trace Bitmap dialog window. As you look at the traced version of the image in the main *Inkscape* viewing area, note that the original bitmap image is still present. So, drag the traced version to one side and then delete the original bitmap (left-click to select and press Delete).



Tracing a bitmap image in Inkscape. Even if you don't understand all the options, try them all out to see what produces the best results.

Clean it up

Zoom in (Ctrl+mouse wheel) to examine the trace. If you spot any elements that you want to remove, use the 'Edit paths by nodes' tool (left-hand toolbar). The type of things that you'll want to deal with are unwanted gaps and jagged edges. Drag over the control points of unwanted elements and then press the Delete key.

To complete the cleanup, look at the status area at the bottom of the window, and you'll see an indication of the number of nodes that make up the image. It's always worth trying the Simplify option (Path>Simplify) to see if you can reduce the overall number of nodes without greatly sacrificing the detail of the drawing. There's no hard and fast rule, but for this sort of work it's usually better to have a smaller number of nodes, both to reduce complexity and to give smoother curves on the edges.

In the case of the example image that we used, the Simplify tool took things down from 933 nodes to a

QUICK TIP

Rely on Undo (Ctrl+Z) to explore techniques, but remember that there are a limited number of undo steps. You can increase the number of available steps (Edit> Preferences... >Undo Steps), but consider doing a save (File>Save As...) before trying things out.

more suitable 75. When you have everything how you want it, save it as an .SVG file (File>Save as...).

Take it to Blender

Back in *Blender*, delete the default cube (left-click and press Delete), and import the SVG file that you've just created (File>Import>Scalable Vector Graphics). *Blender* has a tendency to interpret these images as being rather small. The easiest way to rectify this is to stay at the default zoom level, click the imported object, press S and move the mouse to scale it to an appropriate size on screen. Orientate your view using the zoom (mousewheel) and move (move the mouse with middle mouse button+Shift) functions. Ideally, you want a view of the whole object, slightly tilted back.

At the moment, we have a flat version of the drawing, so we need to give it some depth. *Blender* classes the object as a set of curves, and there's a specialised panel for altering the parameters of such objects, such as the depth. Search the Properties Browser (bottom right-hand side) for the Object Data Properties panel (not Object Properties). In here, open the Geometry subsection. Increase the Extrude parameter, and this should increase the depth of the object, making it truly 3D. Note that this can be a hair-trigger when dragged with the mouse. To get around this, either type smaller values in directly by clicking once on the parameter box or drag over that box with Shift held down.

To make the object look more like a solid, real-world object, bevel it by scrolling down to the Bevel subsection and increasing the depth parameter. As with the Extrude parameter, smaller values tend to work better here.

It's possible that some flaws from our clean-up operation in *Inkscape* will have become apparent by now. If there are odd-looking areas in the 3D object that we now have, consider using Edit mode (press Tab). Having done this, by default, you're immediately in Vertex select mode. For best results, press 7 on the numeric keypad to place the view into the Orthographic (no perspective) Overhead view.

The control points for the curves are represented by the little dots that you should see dotted around the object. If you spot an obnoxious obtrusion that you wish to smooth out, click a control point that's close to it and move it around (press G for grab mode). As often as not, the solution is to move the point nearest the problem area inwards, away from the edge of the object. If that doesn't work, try deleting the point (press X) If you've moved the wrong point or made a mistake, don't worry, it's easy enough to undo it (Ctrl+Z) and try again.

Putting some flesh on the little chap's bones with the Extrude parameter for Curves. Unfortunately, some flaws became apparent, but they're easy to correct in Edit mode.

Add a material

By now we should have a credible-looking 3D shape. However, by default the material properties are set to a shiny black. Let's change it to gold. Make sure you're in Object mode rather than Edit mode (press Tab to switch back). The first thing to do when working with materials is to change to Material View (the third of the four Viewport Shading icons in the top right-hand side of the interface).

Going back to our old friend, the Properties Browser, locate the Material Properties panel. By the way, make sure that the object is still selected. If it's not, left click to select it again. We say this because *Blender* will let you make changes here even if the object isn't selected, without giving you any obvious feedback as to why you're not actually changing anything.

By default, the object should have a material with a name like SVGMat.001. Deleting this existing material and its material slot (press the – button), and having done this, create a new material using the New button. This should give a plain white look to the object.

To make the surface metallic, scroll down the shader parameters and increase the Metallic parameter to its maximum level (1.000) and the Roughness parameter to around 0.030. Change the Base Color parameter to a yellow shade to approximate a gold appearance. Note

QUICK TIP

If you need to use a menu function that doesn't have a keyboard shortcut assigned to it, you can right-click the menu entry and add it from there. Keys F5–F8 are kept free for the user. In addition, you can use the same method to an oft-used menu entry (press G for fast access).

» CREATING A REFERENCE PHOTOGRAPH

If you're taking a photograph to use as a reference image then there are a few useful guidelines we can give. Aesthetics aren't important, and we want something clear, with as little perspective distortion as possible.

As ever with photography, as much light as possible will produce the clearest shots. If you're using a camera, step away from the subject, physically, and zoom in (50mm equivalent or more) to compress perspective. If you're using your phone's camera, step back and then crop the image afterwards. If you get too close, with the wide angle lens of a phone camera then you'll exaggerate the effect of perspective, and for our uses we want to flatten things as much as possible. For the same reason, make sure that the camera or phone is on the same level as the object and as level as you can make it.

Take a few shots, so that you've got some choices once they're transferred onto the computer. Speaking of which, consider editing the image (*GIMP* is a good choice) to crop the image, and use the Rotate Image tool to get things as straight as possible. Because we won't be using the image other than as a reference, consider reducing it to monochrome or increasing the contrast, to make the edges more obvious.

QUICK TIP

Once you get to know Blender, you may find that you keep making the same changes to the settings, such as the rendering setup. In that case, make the changes in a blank, new file, and then go to: Defaults> Save Startup File. That will be your starting point from then on.

that an effect like that one will appear different, depending on the exact shading mode that *Blender* is in. So, for example, metallic materials will look their most realistic using the Cycles rendering engine (selected in Render Properties, in the properties browser, followed by F12 to render).

To make the most of metallic materials, add other objects around them. For example, a gold object floating in space with nothing around it will look mustard coloured and not particularly metallic. At minimum, consider adding a plane (Add>Mesh>Plane) to use as a floor. Scale the floor (Scale tool) to make it stretch into the distance and then place it below the metallic object using the Move tool. That should bring your metallic object to life.

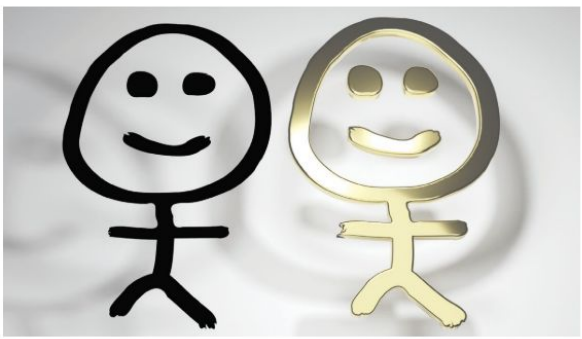
Reference images

Sometimes, when creating a model based on a real-world object, it helps to use a reference image, and *Blender* makes this easy to do. See the box (*page 69*) for tips on taking suitable photographs. In this example, we'll take an image (a mug), trace around the outline and 'lathe' it to produce a 3D object. Because we're going to use *Blender's* modifiers, the entire object will be fully editable at all times. So, for this reason don't worry about making mistakes; it's easy to try something out and then alter it later in the process.

We'll start by setting up our view so that the reference image isn't imported at a skewed angle. Do this by switching to orthographic front-view by pressing 1 on the numeric keypad.

Import the image (Add>Image>Reference). The image is imported as what *Blender* calls an 'empty'. Empties are only visible in the Editor viewport and won't be included in a rendered output image. Although the empty is lined up on the centre of the scene, *Blender* has no way of determining what's in the image, so it's important to visually line it up by hand. Staying in the front view, click the reference image and move it (press G to grab it) so that the actual real-world object is on the centre-line of your view. If you don't do that, you'll end up with an object that's either too wide or too narrow.

We need a mesh to start with; so, if you've removed the default cube, add another one (Add>Mesh>Cube), and left-click it to select it. Go into Edit mode (press Tab), and make sure we're still in front view (1 on the numeric keypad again, if you've changed the view). In actual fact, it's not that important what the mesh is – we just need something to reduce to a single vertex, so do that by selecting Mesh>Merge>At Center.

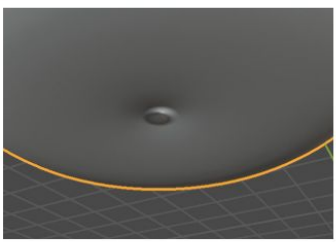


Rendering of the basic, imported SVG file that we traced using Inkscape, transforming the same object with increased thickness, bevelling and a gold-looking material. This is all 'live' using Blender's modifiers; so you can tweak the results to your heart's content.

Trace an outline

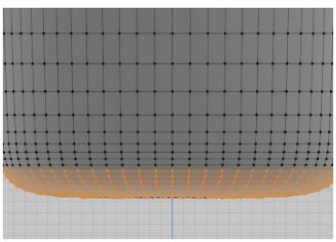
Now we can start drawing the outline of one half of the reference object. Don't forget that you can zoom at any time using the mouse scroll wheel to obtain a more detailed look at what you're doing and scroll around by dragging with the middle mouse button. Move the vertex that we created to one lip of the cup above the right-hand edge (press G to grab it) . Press E (Extrude), which enables you to place a second, connected vertex.

CLEAN UP YOUR IMPORTS



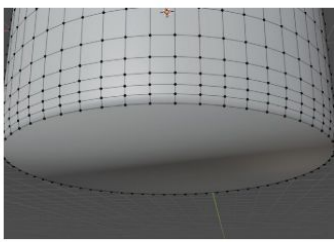
1 Identify the problem

Sometimes when you create a mesh by importing elements, you end up with an anomalous feature that you want to remove, and that's where Edit mode comes in. In the case of the cup that we made from a reference photograph, the bottom appears to be messed up. Select the object, and click Apply on each modifier. Press Tab to go into Edit mode.



2 Remove problem vertices

Press 1 on the numeric keypad to go into Front Orthographic view, and zoom in using the mousewheel. We need to be in Wireframe rendering mode so that we don't merely select vertices on the visible side. Drag over the bottom of the cup, and remove the selected vertices in the problem region (press X and choose Vertices on the pop-up menu).



3 Add new faces

You should see that the cup is missing two bottoms. To add a flat bottom to the object, access Edge select mode (press 2) and select (Alt-click) the loop of edges – the rim – that we need to fill in and press F to add a face. Do this for each missing bottom. You might have to adjust the height of the faces/loops (use the Move tool for this).

Manually creating a path around the outline of the reference photograph in Blender's Edit mode.

Keep doing this until you've drawn an outline of the outside of one half of the mug.

At the bottom of the mug, go slightly beyond the middle. For the job that we're doing here, aim for about a dozen vertices in total, with closer spacing on the more complex curved areas.

Don't worry about making mistakes as you work your way down, because they're easily corrected. If you've misplaced a vertex, click it and press G to move it. To delete a vertex, select it, press X and select Vertices from the menu that pops up. To add an extra vertex, switch to Edge select mode (use the icon or press 2), select an edge (a line between two vertices), then right-click and select Subdivide on the context menu. Then switch back to Vertex select mode (use the icon or press 1). Having made amendments, to go back to adding new vertices, click the final one in the chain and press E again.

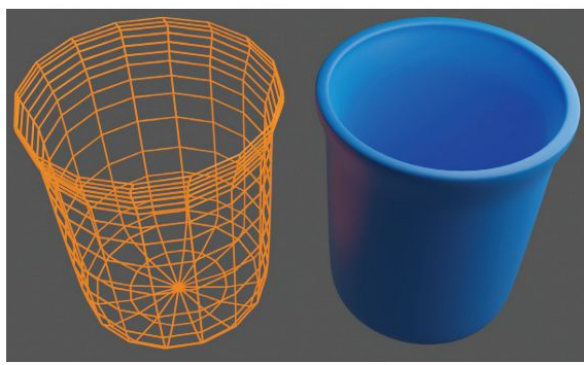
Flesh it out

Once you have a completed outline of one half of the shape, go back to Object mode by pressing Tab. From here, we can start making a 3D shape that resembles the body of the mug. Click the reference image and hide it by pressing H. To unhide, if you need it again, press Alt+H after selecting the empty within the Outline Browser (upper right column).

We'll turn the outline into a three-dimensional object by employing a well-known 3D modelling technique known as solid of revolution. In this case, we'll use *Blender's* Screw modifier to do this. With the outline we created selected, locate the Modifier Properties panel in the Properties Browser, click Add Modifier and select Screw. This should give you the shell of the body of a cup. If something goes wrong then try selecting a different axis in the modifier. Note that the outline is rotated around the origin of the object, which should be in the centre of the screen. If things still don't look right, make sure it's in the middle of the screen (Object>Set Origin>Origin To 3D Cursor with the object selected).

There are a few ways of adding width to the sides and the bottom of the cup, but for speed we'll add a second modifier for this purpose. Add a Solidify, and in its panel increase the thickness to something that gives a credible appearance to the cup. The object we have now has substance to it, but it's still rather angular looking. To fix this, add a third modifier to the object, the Subdivision Surface modifier. Once this is added, increase the Levels setting to 2.

Hopefully, you've now got a nice-looking cup object. Remember that because we've used modifiers without applying them, you can press Tab to go back into Edit mode at any time. From here you can move vertices and see the results of the edits in real time until you've tweaked everything the way you want it. It might be a good idea to use the methods from the previous section to add a material to the object, and from there, adding a texture. If you want to edit the finished object, apply each of the modifiers to turn it into a mesh. LXF



A rotational solid created from a manually traced side-profile: using the Wireframe modifier (left); and using the Solidify modifier and applying a material (right).

» APPLY TEXTURE TO YOUR OBJECT

There are plenty of websites that supply textures for free download such as **https://ambientcg.com** or **www.blendernation.com**. Find something suitable in PNG or JPG format and then download it. Locate the Material Properties panel and add a new material as described in the main tutorial and add a new material.

Click the yellow dot next to the Base Color field, and from the pop-up menu, choose Image Texture. Then press the Open button and select the texture that you downloaded, and this will apply the texture to the object.

By default, this is done in a crude manner, and dealing with problems is a complicated subject in itself. However, there's a quick fix that works in most situations. At the top of the window, click the UV Editing tab. In here, press A to select all. You should see that selected portions of the object (right window) are mapped to the texture (left window). In the UV menu (right window) select the option UV Smart Project. In this dialog, select Scale to Bounds. Finally, click OK and return to the Layout tab.

EMULATION

Emulating the classic Altair 8800

Launched in 1975, the Altair 8800 is considered to be the first computer aimed at enthusiasts. **Mike Bedford** shows you how to experience it.

OUR EXPERT

Mike Bedford can't claim to have used an Altair 8800, but some time later he did own a not-too-dissimilar computer that he had to solder together himself.

Having delved into various emulators of long-gone computers in recent issues, we've become accustomed to how primitive some of these early machines really were. Even so, we suspect that the specification of the computer we're looking at here will still be quite an eye-opener. That machine is the MITS Altair 8800 from Micro Instrumentation and Telemetry Systems (MITS), and its claim to fame is that it was the first personal computer to be aimed at enthusiasts, as opposed to business users. The first most people had heard of it was when it featured on the front cover of *Popular Electronics* magazine in January 1975. Compared to the home computers of the early 80s, like the ZX80 or the BBC Micro, you didn't get a lot of computer for your money.

The processor was an eight-bit Intel 8080 that was clocked at 2MHz, and the memory capacity was just 256 bytes. What's more, that RAM wasn't augmented by a non-volatile ROM containing a primitive operating system or the BASIC language, it didn't have a keyboard, there was no way of attaching a screen, not even a low-resolution TV, and there was no storage such as a cassette recorder, let alone a floppy or hard disk. Yet it cost $439 as a kit, or $621 ready built – prices being today's equivalents of $2,200 and $3,100 respectively.

It would be understandable, when presented with a list of everything the Altair 8800 didn't have, that you might wonder how it was possible to do anything at all. As we're about to see, though, it did enable those who bought it to enter and execute their own programs and learn a lot more about the inner workings of computers than would be gleamed by using a modern PC.

It might look primitive today, but appearances are deceptive: in 1975 the Altair 8800 was truly ground-breaking.

CREDIT: Ed Uthman, CC BY-SA 2.0, https://commons.wikimedia.org/wiki/File:Altair_8800_Smithsonian_Museum_(white_background).jpg

Assembler code only

To see what it was like to operate we're going to use the online simulator at **https://s2js.com/altair/sim.html**. You'll notice that it looks like the Altair 8800's front panel which, since it couldn't be connected to a keyboard or display screen, provided the sole user interface. In particular, user input was provided via the switches and output was through the various LEDs.

Start by turning on the simulator by clicking the on/off switch. This is a locking toggle switch so it'll move down to the on position and remain in that position, various status LEDs will illuminate, and you'll even hear the fan start up. Because there's no non-volatile memory, and therefore there's no software to run when you first turn on the Altair 8800, the first job at power-on will always be to enter some code into memory. This is done using the switches and LEDs which, because of the lack of any software at power-on, are connected pretty much directly to various pins on the processor.

To relive the experience of an Altair 8800 user of 46 years ago, we'll enter a simple program that was provided in the user manual and, in so doing, learn how to use the front panel controls. That program adds the contents of two memory locations and stores the result in a third location. If we were to write the program in 8080 assembler language it would be as follows:

```
VAL1    EQU     80H
VAL2    EQU     81H
SUM     EQU     82H
        ORG     00H
START:  LDA     VAL1
        MOV     B,A
        LDA     VAL2
        ADD     B
        STA     SUM
        JMP     START
```

The first four lines are assembler directives, as opposed to processor instructions. The first three define symbols that represent memory addresses, and their use enables those instructions that access memory to refer to meaningful names instead of just numbers. And the ORG indicates that the following code is intended to be loaded into memory, starting at the specified address which, in this case, is 00 hexadecimal.

QUICK TIP

You might recall the 3.5-inch floppy disks (which had a hard outer-casing so only the disk inside was actually floppy) that could be read by PCs until a few years ago. Before that were five-inch floppy disks, which were genuinely floppy, but the Altair 8800 used the even earlier eight-inch floppy disk.

Next, we have the actual 8080 instructions, the first of which has the label "START". The LDA instruction loads the content of the specified memory location (specifically "VAL1" which was defined as representing the address 80 hexadecimal) into the accumulator. The MOV instruction moves the value in the accumulator to the B register.

The next instruction is another LDA instruction, the purpose of which should be clear. Then, the ADD instruction adds the content of the B register to the contents of the accumulator, and the STA instruction writes the contents of the accumulator into the memory location "SUM", which equates to 83 hexadecimal. This completes the actual program, but if we left it there it would continue by executing the contents of the following memory locations, as if they were instructions, with unpredictable results. For this reason, we've added a JMP instruction which causes execution to return to the first instruction, so it'll loop forever.

Bare bones computing

Because the Altair 8800 came with no software, and because the front panel only accepted binary numbers, it would have been necessary for users to hand-assemble our sample code by reference to the manual for the Intel 8080, which you'll easily find online. It probably goes without saying that it would have been time consuming, and all too easy to make mistakes.

Today, we can carry out the process automatically using an assembler, for example at **www.asm80.com/onepage/asm8080.html**, so you might like to cheat by entering the above code and clicking Assemble. This generates a machine code program, that's a list of op codes – the binary numbers that represent the processor instructions. The table (*overleaf*) shows the op codes at each address, and these are provided in hexadecimal, binary and octal. The binary data has a 1-to-1 correspondence with the front panel switches and LEDs. The octal data is also useful for entering or inspecting data via the front panel because the switches and LEDs are arranged in groups of threes, and the hexadecimal data is useful because this is how numbers are represented in the simulator's register and memory dump feature. You'll notice that some of the op codes occupy more than a single location. For example, the first instruction, LDA VAL1, occupies three locations.

Now it's time to enter this code into the Altair 8800 simulator. First, reset the machine by clicking the top part of the RESET/CLR switch. From now on, though, when we need to use a switch like this with two functions, we won't specifically refer to the top or bottom part, but we'll just specify the appropriate half of the label so, in this case, we'd have referred to the RESET switch. Resetting the machine selects address `0x0000` (we're now using the prefix `0x` for hexadecimal, `0b` for binary, `0o` for octal and nothing for decimal) for any subsequent reads or writes to memory. Strictly speaking it isn't necessary if the Altair 8800 has only just been switched on, but it's good to get into the habit.

Unlike the real Altair 8800, this simulator will display a full memory dump and register contents, but if you choose not to use that feature then it provides the same experience as the original.

The first instruction is entered by writing `0b00011010` to address `0x0000`, so set that binary code on the Data/Address switches for 7 to 0, which are the same switches that are used for A7 to A0 for read operations. Switches should be up for 1 and down for 0, and each time you click one of these switches it toggles between the two. It's easier to enter binary digits in groups of three, which is why the switches are grouped in threes, and this is also why we provide the octal equivalent of each instruction.

Click the DEPOSIT switch to "deposit" (i.e. write) the number on the switches to the selected address. Having clicked that switch, the value just written will appear on the data LEDs 7 to 0 and the address it's been written to will be shown on the address LEDs A15 to A0, but since that address is zero on this occasion, the address LEDs will all remain extinguished. The next value of `0b10000000`, which needs to be written to address `0x0001`, is written in a slightly different way. As before,

QUICK TIP

Altair 8800 expansion cards were designed to plug into 100-way sockets on the backplane. Those sockets hadn't been designed for use in computers, yet MITS' assignment of the signals to the pins eventually became a standard known as the S-100 computer bus.

» TRY A REAL ALTAIR 8800

Want to try a real Altair 8800 instead of a simulation? The bad news is that it won't be cheap because these vintage machines are in considerable demand by collectors. However, you can come close to the real Altair experience thanks to the availability of several modern clones. Some still cost more than a cheap laptop that would be phenomenally more powerful, but in real terms they cost much less than it would have cost a 1975 enthusiast. What's more, if your first experience of computing was back in the 1970s, getting your hands on one of these clones will offer you that proverbial trip down memory lane.

One of the best known Altair 8800 clones, ready built in a case that's a reasonably good replica of the original machine, can be found at **https://altairclone.com**. It's interesting to note that it's been priced exactly the same as the very first Altair 8800, at $621. However, because of inflation over the course of the intervening years, it's much cheaper in real terms. In reality, though, it's cheaper still because several things that you had to buy as extras in 1975, such as more than 256 bytes of RAM, a floppy disk controller and drives, serial I/O and an EPROM card, are provided as standard. Although this clone takes the form of a real physical computer, though, it's still an emulation because its processor is a Microchip PIC24FJ128 microcontroller, not an Intel 8080.

start by setting the binary code on the same switches, but this time click the DEPOSIT NEXT switch part to write it to the next address. Continue in this way until you've entered all the code.

Let's enter the two numbers to be added together. It makes sense to separate data from code in memory, so it would be possible to extend the code without overwriting the data, and this is the approach we've used here. Specifically, the two numbers to be added are at addresses `0x0080` and `0x0081`, and the result will be written to address `0x0082`. However, this gap means that we can't write the values to memory by continuing to use the DEPOSIT NEXT switch.

Set the address `0x0080` (`0b0000000010000000`, `0o000200`) on the switches, which are now used for specifying an address instead of data. Next, click the EXAMINE switch to "examine" (i.e. read) the value at the address specified by the switches. The value will appear on the LEDs 7 to 0, but it also defines the address for subsequent writes. For this reason, you can now set the data on the switches for 7 to 0, click the DEPOSIT switch and then write the other value to address `0x0081` using the DEPOSIT NEXT switch. Ensure the sum of the two values doesn't exceed `0xFF`, for example 256.

Address			Op Code		
Binary	Octal	Hex	Binary	Octal	Hex
0000000000000000	000000	0000	00111010	072	3A
0000000000000001	000001	0001	10000000	200	80
0000000000000010	000002	0002	00000000	000	00
0000000000000011	000003	0003	01000111	107	47
0000000000000100	000004	0004	00111010	072	3A
0000000000000101	000005	0005	10000001	201	81
0000000000000110	000006	0006	00000000	000	00
0000000000000111	000007	0007	10000000	200	80
0000000000001000	000010	0008	00110010	062	32
0000000000001001	000011	0009	10000010	202	82
0000000000001010	000012	000A	00000000	000	00
0000000000001011	000013	000B	11000011	303	C3
0000000000001100	000014	000C	00000000	000	00
0000000000001101	000015	000D	00000000	000	00

Here's the sample code that we use on the simulator of the unexpanded Altair 8800.

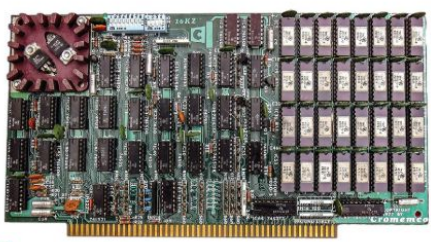
At this point you might like to use a facility of the simulator that wasn't available to users of the real Altair 8800, so click the tab at the bottom of the image of the machine and a panel will appear showing the contents of the processor's registers and memory. We suggest you use this to check that you've correctly entered the code and data before continuing. Once you're confident that the program and data are correct it's time to run the program, so click RESET and then click RUN. You'll notice various LEDs flashing and, within just a fraction of a second, it'll have repeatedly added the two numbers, so you can click STOP. Now, examine address `0x0082` and hopefully you'll find the expected answer.

If you've only ever programmed in high-level languages, you might be surprised to learn that it required five 8080 instructions to add two numbers together. After all, this could have been done in a single instruction with a high-level language. What's more, given the clunky way in which instructions were written to memory, this was much more of an issue than it would be in entering machine code on a modern PC.

We have to wonder how common it was to users of an unexpanded Altair 8800 to write much more than simple programs. Certainly programs couldn't have exceeded 256 bytes in length because of the limited memory capacity, but it would have been a major undertaking to enter even a 256-byte program. If it took 15 second to write to each memory location, the whole job would have taken about an hour. And there was no way of saving the code. The next time you wanted to use the program, and you'd turned the machine off in the meantime, you'd have had to enter it all again.

Expanding the Altair 8800

Given the primitive way of using the basic Altair 8800, it would have come as a relief to users that it had been designed to be expandable, albeit at a not inconsiderable cost. Indeed, even in its entry-level configuration, it had two boards that were connected together via a passive backplane. These two boards were the CPU board, which contained little more than the processor, some interfacing circuitry and a 1KB memory board, which was populated with sufficient RAM chips to provide 256 bytes of memory.



The Altair 8800 was expanded via S100 boards, like this 16k memory card designed for a slightly later Cromemco computer.

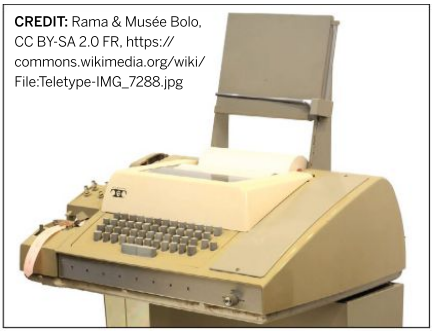
CREDIT: Cromemco, CC BY-SA 3.0, https://commons.wikimedia.org/wiki/File:Cromemco_16KZ_S-100_Board.jpg

» SPOTLIGHT ON THE 8080 CHIP

The 8080 was only Intel's third ever microprocessor, following on from the four-bit 4004 which appeared in 1971, and its first eight-bit chip, the 8008 which appeared soon after. Its instruction set comprised just 78 instructions, and our simple machine code programming example served to illustrate one of its shortcomings.

In particular, the five instructions needed to add two values from memory and write the answer to another location is a couple more than many people would expect, if they've not previously used the 8080. Specifically, on processors that came out just a couple of years later, just three instruction would have been needed. First, load a value from memory into the accumulator; second, add a value from memory to the value in the accumulator; and third, write the value in the accumulator to memory.

The snag with the 8088, which resulted in those two extra instructions, is that the add instruction could only add a value from an internal register to the value in the accumulator. Actually, that's a slight oversimplification, because the specified register can be a pair of special registers that point to a memory location, but to use that the values in those two registers would first have to be initialised. This state of affairs applied to several instructions, not just the add instruction, so more instructions were often needed to do a particular job on the 8080 than on slightly later chips, such as the 6502.



CREDIT: Rama & Musée Bolo, CC BY-SA 2.0 FR, https://commons.wikimedia.org/wiki/File:Teletype-IMG_7288.jpg

Altair 8800s were sometimes used with Teletype ASR-33s, which provided a keyboard, printer and paper tape reader/punch, but at a considerable cost.

To see just how different a more fully configured Altair 8800 was, and how remarkably sophisticated this ground-breaking machine could be back in 1975, we're going to turn our attention to a different simulator called *SimH*. This open source software is actually a family of simulators of historic machines, and you can find the Altair 8800 simulator at **https://schorn.ch/altair.html**, together with a quick-start guide. Before getting some hands-on experience, though, let's see just how the machine simulated here differs from the entry-level computer.

Perhaps most importantly, this expanded Altair 8800 has a serial interface card ($119 kit or $138 assembled). This enabled it to be connected to an external terminal, thereby providing a more sophisticated user interface than the front panel switches and LEDs. One option for that external device would have been a CRT terminal which provided an alphanumeric display and a keyboard, which connected to a computer via a serial RS232 interface. Although even the cheapest third-party CRT terminals cost almost $1,000, MITS advertised one for $129/$169.

Just adding an interface board wouldn't have been very useful, because the Altair 8800 didn't have the necessary software to talk to it. Adding a non-volatile ROM expansion into where the required start-up code – rather like a BIOS – could be stored would be necessary. Such a card cost $76/$128. With a sensible user interface, it became practical to use a high-level language such as BASIC. In fact an implementation of BASIC for the Altair 8800 was written by Bill Gates as Microsoft's first commercial enterprise. This was supplied on audio tape or paper tape. Accessing this required another expansion card. Alternatively, paper tape could be read using a paper tape reader.

BASIC required more memory than 256 bytes, and even populating the standard memory card to its full capacity of 1KB wasn't sufficient. You needed either one or two 4KB memory boards ($264/$338 each) depending on the version of BASIC was required. While not essential the Altair 8800 configuration in the SimH simulator runs the CP/M operating system from floppy disk, which required a floppy disk drive and an interface card, which were supplied together for $1,480/$1,980. We'll leave you to add all that up (and don't forget $150 for the disk file-enabled version of BASIC, and an unspecified amount for CP/M), and multiply the total by 4.9 to take account of inflation, but it's probably fair to say that only the very well-healed enthusiast would have aspired to such a system.

Now you understand more about the machine being simulated, it's time to fire up the *SimH* simulator and try your hand at BASIC. As described at **https://schorn.ch/altair.html**, you'll start *SimH* in a terminal window, which is where CP/M will start up, thereby presenting the same sort of experience of running the Altair 8800 on a CRT terminal or teletype. You'll notice that the name of the executable is *AltairZ80*, which seems surprising. However, later expansion cards offered the possibility of migrating to the more powerful Zilog Z80 processor.

When *SimH* starts up, you'll see some introductory text. If you want to prove that you're actually running a disk operating system, just type `DIR` and you'll see a list of files on the A disk. One of those files is MBASIC.COM, so type `MBASIC` and you'll be transported to Microsoft Altair BASIC. If you want to try a ready-written program, type `LOAD "ELIZA"` and, when it's loaded, type `RUN`.

This version of BASIC is much closer to its roots than those that appeared with home computers in the 1980s, which means that it's less sophisticated in many ways, although it does go beyond most of the 1980s BASIC versions in providing support for reading to and writing from disk. To learn more about it, take a look at the Microsoft BASIC 80 manual at **https://altairclone.com/downloads/manuals/**. When you're done, type `SYSTEM` to return to CP/M. LXF

```
mike@Mike-PC: ~/Downloads
64K CP/M Version 2.2 (SIMH ALTAIR 8800, BIOS V1.27, 2 HD, 02-May-2009)

A>DIR *.COM
A: ASM      COM : COPY     COM : CPU      COM : CREF80   COM
A: DDT      COM : DDTZ     COM : DO       COM : DUMP     COM
A: ED       COM : EX8080   COM : EXZ80ALL COM : EXZ80DOC COM
A: FORMAT   COM : GO       COM : HALT     COM : L80      COM
A: LADDER   COM : LIB80    COM : LOAD     COM : LS       COM
A: LU       COM : M80      COM : MBASIC   COM : OTHELLO  COM
A: PIP      COM : PRELIM   COM : RSETSIMH COM : SID      COM
A: HDIR     COM : STAT     COM : SUBMIT   COM : SURVEY   COM
A: W        COM : XFORMAT  COM : TIMER    COM : UNCR     COM
A: UNERA    COM : USQ      COM : BOOT     COM : WM       COM
A: WORM     COM : DIF      COM : XSUB     COM : ZAP      COM
A: ZSID     COM : ZTRAN4   COM : R        COM : SHOWSEC  COM
A: SPEED    COM : SYSCOPY  COM : URL      COM : BOOTGEN  COM
A>MBASIC
BASIC-80 Rev. 5.21
[CP/M Version]
Copyright 1977-1981 (C) by Microsoft
Created: 28-Jul-81
32824 Bytes free
Ok
LOAD "ELIZA"
Ok
LIST
5 PRINT TAB(16);"************************"
10 PRINT TAB(26);"ELIZA"
20 PRINT TAB(20);"CREATIVE COMPUTING"
30 PRINT TAB(18);"MORRISTOWN, NEW JERSEY":PRINT
40 PRINT TAB(19);"ADAPTED FOR IBM PC BY"
50 PRINT TAB(20);"PATRICIA DANIELSON AND PAUL HASHFIELD"
```

In its expanded guise, as recreated here in the SimH simulator, using the Altair 8800 really wasn't too different from using a command line interface today.

QUICK TIP

Paper tape readers weren't fast at either 110 or 300bps. So, loading BASIC from a paper tape, as you'd have done if you weren't lucky enough to have a floppy disk drive on your Altair 8800, would have taken several minutes.

POKY LINUX

Credit: www.yoctoproject.org

Mastering the smallest Linux desktop distro

Poky Linux is a great DIY Linux distro builder for embedded systems. Find out how **Alexander Tolstoy** uses it to prep an image for a desktop setup.

OUR EXPERT

Alexander Tolstoy is a system administrator and DevOps engineer who enjoys exploring open source software.

The general sequence of baking a standard Poky Linux image is pretty straightforward – see Tam Hanna's tutorial in **LXF251**. In short, you start by preparing your rhost system for building by installing development packages such as *gawk*, *texinfo* and *diffstat*. Then grab the Poky code from the Yoctoproject Git repository, initialise the build environment with the bundled script, and start baking (`$ bitbake <some image>`).

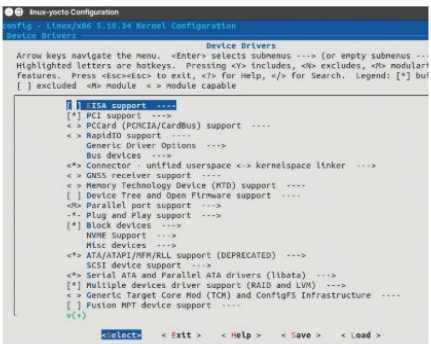
This last part is very resource intensive. You'll need to set aside several hours, during which time your host system will be bogged down by *Bitbake*, and you'll also need to make sure you have at least 70GB of free disk space, otherwise there won't be enough room for *Bitbake* to do its job. All the extra software parts added on top of the minimal image also require gigabytes of free disk space, so don't cut this corner.

These are just the minimum requirements for getting a tiny bootable Linux distribution up and running, which occupies just tens of megabytes on its own. On the other hand, most processes in Poky are well automated and therefore introduce far fewer difficulties than, say, the Linux From Scratch (**www.linuxfromscratch.org**) build routine.

Yocto on your desktop

While Yocto is primarily aimed at embedded systems, we'll be looking at desktop usage. The idea is to access the benefits of Poky and bring it closer to a standard desktop workstation. As such, we can ignore everything related to board support packages (BSP), which are designed for device-specific builds (such as those for RaspberryPi), and concentrate on the generic x86-64 build. It's this one that we can run on a regular PC.

The difference between a tiny standard image suggested by *Bitbake* (for example, **core-image-minimal**) and our goal is that the latter needs a lot of extra software pieces to be manually added in order to produce something viable. To be more specific, these additions include extra drivers in kernel for better real-world hardware support, middleware insertions for better experience (*OpenSSH*, *NetworkManager* and *PulseAudio*, for example), a basic graphical environment together with any extra applications you need. We'll start our journey by obtaining the essential part of the Poky codebase.



Running the build on a bare-metal system makes you'll need to tweak the Linux kernel configuration. This handy menu will help.

Getting the basics

We kick things off by obtaining the code from the Yocto servers. Before we proceed with Git, let's find out the codename of the latest Poky release by looking at **https://wiki.yoctoproject.org/wiki/Releases**. At the time of writing it is Hardknott. Once we have the correct name we're ready to fetch the Poky source code:

```
$ git clone -b hardknott https://git.yoctoproject.org/git/poky
```

Enter the downloaded directory (usually it's **poky**) and source the **oe-init-build-env** file to initialise the build environment:

```
$ ./oe-init-build-env
```

This will create the local build configuration and enable you to start baking your own Poky build. By default our target is called **qemux86-64**, which is perfect for testing. We can leave it intact, or change it to **genericx86-64** to prepare more general-purpose builds. The setting is stored in the **conf/local.conf** file. For example, to change the target machine type from Qemu to Generic X86_64, uncomment the following line:

```
MACHINE ?= "genericx86-64"
```

QUICK TIP

Delete files in the deploy/images directory with care. You need to know which package generates the deploy target, because an image may depend on the output files generated by another image. Keep your test sandbox safe!

In addition, comment out the line with the `qemux86-64` value. You can leave it intact to make *Bitbake* build two images, but this will take up extra time and disk space. Right now there's no need to change other defaults, and therefore we may safely proceed with building the image:

```
$ bitbake core-image-x11
```

The build process means compiling the whole Linux system and all its components from their source code. It'll take several hours even on a powerful host, but after all tasks have been completed we'll have a ready-to-use system stored under **build/tmp/deploy/images**.

Note that you can build any number of images using the same build tree. For a different image *Bitbake* will just add the missing parts and repack the previously built recurring packages into it. Essentially, this means you don't need to wait as long for subsequent images to be created.

Extending the build

It's time to enhance the default code tree with auxiliary component known as the OpenEmbedded project. This approach will add more layers and plenty of extra software. In particular, systems with those extra layers will instantly be able to build the **core-image-minimal-xfce**, which is a perfect desktop starting-point.

To add layers provided by the OpenEmbedded project, use Git to download within a main Poky tree:

```
$ git clone git://git.openembedded.org/meta-openembedded -b hardknott
```

It's not enough to just get hold of the code, because Bitbake hasn't been instructed to use it. We now need to add proper lines into the **bblayers.conf** file. It's possible to add this either by editing that file, or using the *bitbake-layers* utility with the following syntax:

```
$ bitbake-layers add-layer /path/to/layer
```

Here is the example part of **bblayers.conf** featuring layers provided by the Openembedded project:

```
BBLAYERS += " \
  /path/to/poky/meta-openembedded/meta-oe \
  /path/to/poky/meta-openembedded/meta-multimedia \
  /path/to/poky/meta-openembedded/meta-networking \
  /path/to/poky/meta-openembedded/meta-gnome \
  /path/to/poky/meta-openembedded/meta-python \
  /path/to/poky/meta-openembedded/meta-xfce \
    "
```

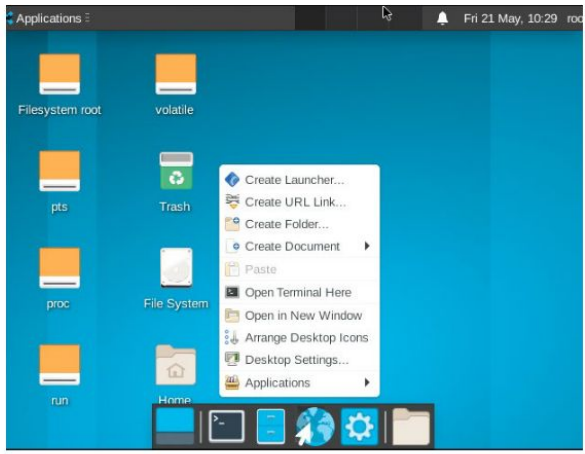
Create a practical desktop

Let's build the image that would include the lightweight Xfce desktop. The **meta-xfce** layer already bundles the ready-to-use recipe under **meta-openembedded/meta-xfce/recipes-core/images/** and therefore we can immediately start the image-generating process with the following command:

```
$ bitbake core-image-minimal-xfce
```

The resulting image will be a modern Wayland-enabled system bundled with lots of programs that we otherwise would need to add by hand. There's a great collection of desktop accessories and frequently used tools that normally ship with the Xfce desktop.

Mastering a perfect Poky Linux image assumes constant test boots, for which the most convenient tool is the *Qemu* emulator. It's perfectly safe to stay in the *Qemu* sandbox during the debugging stage and switch the build target to **genericx86-64** in the end. Before that, test your build using the Yocto-provided *runqemu* utility. It defaults to using as little as 256MB of RAM, so use this code to expand this value – say, up to 1GB:



It doesn't take much effort to generate a basic desktop image with the nifty Xfce environment.

```
$ runqemu qemux86-64 qemuparams="-m 1024"
```

By default, the target system has only one user (root) with a blank password. In order to create a user, add the following line to the recipe file:

```
EXTRA_USERS_PARAMS = "useradd -P <password> <username>;"
```

In our case, it would be **meta-openembedded/meta-xfce/recipes-core/images/core-image-minimal-xfce.bb**.

Layer cake

The simplest way to add extra packages to your build is to append them right into the **local.conf** file. For instance, let's add *OpenSSH* in order to have remote access to our build. Add the following line:

```
CORE_IMAGE_EXTRA_INSTALL += "openssh"
```

» CLEANING UP THE BUILD

You'll soon notice that whatever you do with your project, the size of the **build** directory only grows. At some point, *Bitbake* takes up all of your disk space leaving you unsure how to free it up. When you build a system image with *Bitbake*, the dependencies required to build it are recursively discovered and built. That's not the case for the reversed action, when you use the built-in clean command:

```
$ bitbake core-image-minimal -c clean -f
```

This command will only clean the target image files, not its dependencies. However, you can manually delete either unused or unnecesary parts of the build tree, such as:

```
$ rm -rf tmp/work # source trees of individual packages
$ rm -rf tmp/sysroots # various host-side utilities
$ rm -rf tmp/deploy/images # accumulated build targets
```

It's also possible to clean individual packages if you know their exact names using this template: `$ bitbake <package> -c clean -f` . This is sometimes required if the previous build process was interrupted. As a last resort, use the following command to clean up all build data from a package:

```
$ bitbake <package> -c cleanall -f
```

QUICK TIP

If you build more than one image, it may become unclear what exactly you're going to run with QEMU during your tests. You can instruct the runqemu utility what you want it to run. For example, $ runqemu qemux86-64 core-image-minimal ext4.

It's possible to specify several packages using space as a delimiter. This technique is mostly suitable for adding a few extra packages to the stock setup.

If there are going to be a lot of extra packages, it'll make more sense to set up a separate layer where all our additions will reside. To do so, issue the following commands from Poky's build directory:

```
$ bitbake-layers create-layer --priority 7 ../meta-custom
$ bitbake-layers add-layer ../meta-custom
```

The last command will add the appropriate line into the **bblayers.conf** file for you, which enables you to engage the new custom layer. At the moment it's empty, but we can move extra packages appends to **meta-custom/layer.conf** from the global **local.conf**. If later on there's a need to populate the build with custom recipes, we'll already have a layer for them.

Modern niceties

Historically, adding a new software to Poky meant writing a recipe for building it. A recipe is an instruction that's similar to PKGBUILD in Arch or SPEC in an RPM-based Linux distro. However, the Poky Linux ecosystem, with its rich collection of third-party layers and recipes, contains everything an average desktop user needs.

It's possible to include a fancy Yocto boot logo in your image and help it look even more professional.

» DON'T GET STUCK AT LOG-IN TIME!

If you want to access the target system as root without a password, uncomment the `EXTRA_IMAGE_FEATURES ?= "debug-tweaks"` line in the **local.conf** file. If you don't do that and at the same time deflect from explicitly adding a user in one of your recipes (`EXTRA_USERS_PARAMS`), you won't be able to log in the target system at all. Use the following command to quickly find out if you have enabled root access without a password or not:

```
$ bitbake -e | grep "^EXTRA_IMAGE_FEATURES"
```

This is deliberately maintained as a part of Poky's security policy. As you may know, one of the most common vulnerabilities of routers, Pi devices and other boards that Poky is designed to counter is a weak or missing root password. The current approach to network security assumes that once you have Bitbake up and running, you need to explicitly set up users and their passwords. Bitbake supports various manipulations with users and groups, and there's a special dummy recipe for that at **meta-skeleton/recipes-skeleton/useradd/useradd-example.bb**. Customise it by entering real user names and passwords and include that recipe to the build. The `EXTRA_USERS_PARAMS` parameter supports several commands, such as `useradd` , `usermod` and `groupadd` .

The first place to search for a specific program that's not available in Poky right out of the box is **https://layers.openembedded.org**. It lists various third-party layers that resemble PPAs in Ubuntu or COPR repos in Fedora. A layer may depend on some other layers, similar to how a repository may require another repository. Luckily, when we move down to the recipe level, there are no signs of a dependency hell that Linux users used to suffer from years ago. However, some manual effort may be required to arrange layers properly. As an example, here's the sequence for adding the *Chromium* web browser to the build:

```
$ git clone -b master git://github.com/OSSystems/meta-browser.git
$ git clone -b hardknott git://git.openembedded.org/meta-python2
$ git clone -b hardknott https://github.com/kraj/meta-clang.git
$ bitbake-layers add-layer meta-browser meta-python2 meta-clang
```

Then we add these two lines into the **layer.conf** file of the **meta-custom** layer:

```
IMAGE_INSTALL_append = " chromium"
LICENSE_FLAGS_WHITELIST += "commercial_libav commercial_x264"
```

The second line is required because of licencing limitations of some codecs used by *Chromium*.

If necessary, use the `IMAGE_INSTALL_append` line to add extra software to your Poky build, such as *NetworkManager* or *Gimp*. However, some system-wide software needs to be enabled in the high-priority configuration file. For example, if you want to enable Systemd, don't use your own recipe. Instead, place the following lines directly into the main **local.conf** file:

```
DISTRO_FEATURES_append = " systemd"
DISTRO_FEATURES_BACKFILL_CONSIDERED += "sysvinit"
VIRTUAL-RUNTIME_init_manager = "systemd"
VIRTUAL-RUNTIME_initscripts = "systemd-compat-units"
```

Bitbake builds any extra packages and resolves runtime dependencies automatically. It's just a case of waiting patiently and getting ready to test the image when everything's completed.

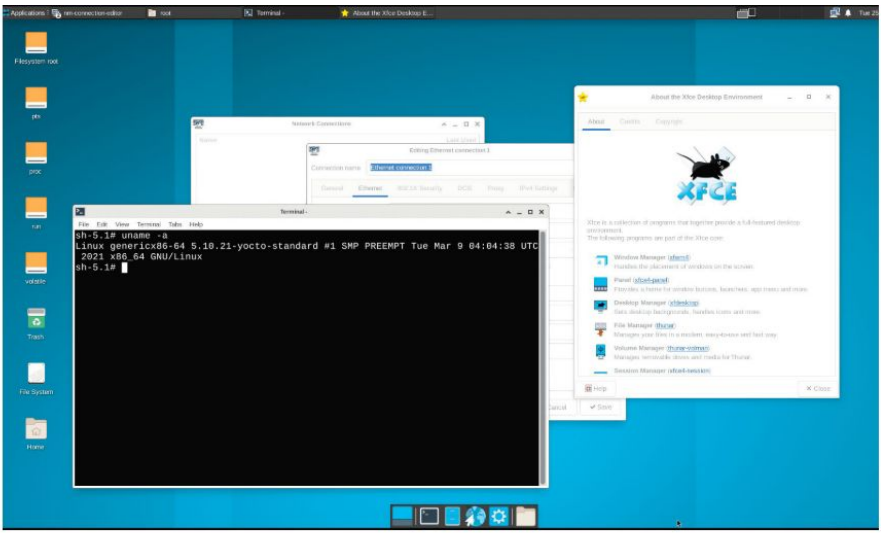
Extra kernel things

You'll never avoid altering the kernel, unless Poky defaults to a limited set of supported hardware, which won't be of much use to desktop systems. We need to change the kernel configuration to add more device drivers. This can be done by entering the standard menu configuration interface:

```
$ bitbake -c menuconfig virtual/kernel
```

After a while, you'll see a text UI with a tree-like menu of the Linux kernel options. At this stage it's important to locate and enable all drivers that you'll need on the real hardware where your build will run. Each kernel driver can be compiled either as a standalone module (**.ko** under **/lib/modules**), or as a part of the initial RAM disk (initrd). The difference mainly comes down to the fact that a driver should be placed inside Initrd if it's required for booting the system (if it's a file system driver, say). Once you've finished configuring the kernel, save your changes and exit the menu for the next step:

```
$ bitbake -c savedefconfig virtual/kernel
```



The Xfce recipe provides a ready-to-use, configurable desktop that runs on top of the recent Linux kernel.

However, as long as we have our own 'meta-custom' layer, let's add the kernel configuration append file (**.bbappend**):

```
$ mkdir -p meta-custom/recipes-kernel/linux
$ touch meta-custom/recipes-kernel/linux/linux-yocto_5.10.bbappend
```

There are lots of things we can do with the Linux kernel here, including adding patches, pushing extra files and so on, but at this stage enabling certain kernel features will suffice nicely. In the recipe directory of your layer create another subdirectory and name it **files** (so that it reads **meta-custom/recipes-kernel/linux/files**). Create an empty text file under it with the name **defconfig** and populate it with lines like `CONFIG_SOMETHING=y`, where `y` means baking the feature right into the Initrd image. Alternatively, replace `y` with `m` to make it a standalone module. To refer to the **defconfig** file in the append file, place the following two lines inside the latter:

```
FILESEXTRAPATHS_prepend := "${THISDIR}/files:"
SRC_URI += "file://defconfig"
```

The only problem left is how to determine the list of modules that you'll actually need. If you plan to run the Poky build directly on the bare-metal system such as your current host, it makes sense to collect some data on it. For instance, you may want to run the `$ make localmodconfig` against the kernel source tree on your host system in order to generate the **.config** file with all the modules that are currently in use. Then simply transfer the file's contents to your layer's **defconfig** file to create a similar kernel setup for the target Poky build. Re-run the *Bitbake* image building command in order to reconstruct the kernel.

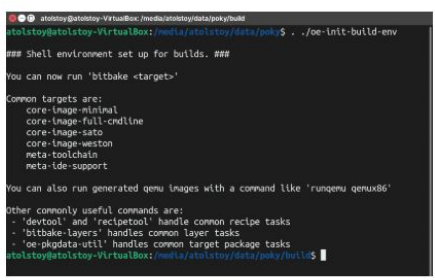
I like to boot it, boot it

Again, the *Bitbake* defaults produce a number of standalone files that you're meant to manually copy to your target device. However, we get little use of *Bzimage* and *Initrd* if we're unable to boot the system outside of *Qemu*. To fix it, let's tell *Bitbake* to produce a bootable ISO image. Add the following line into the **local.conf** file:

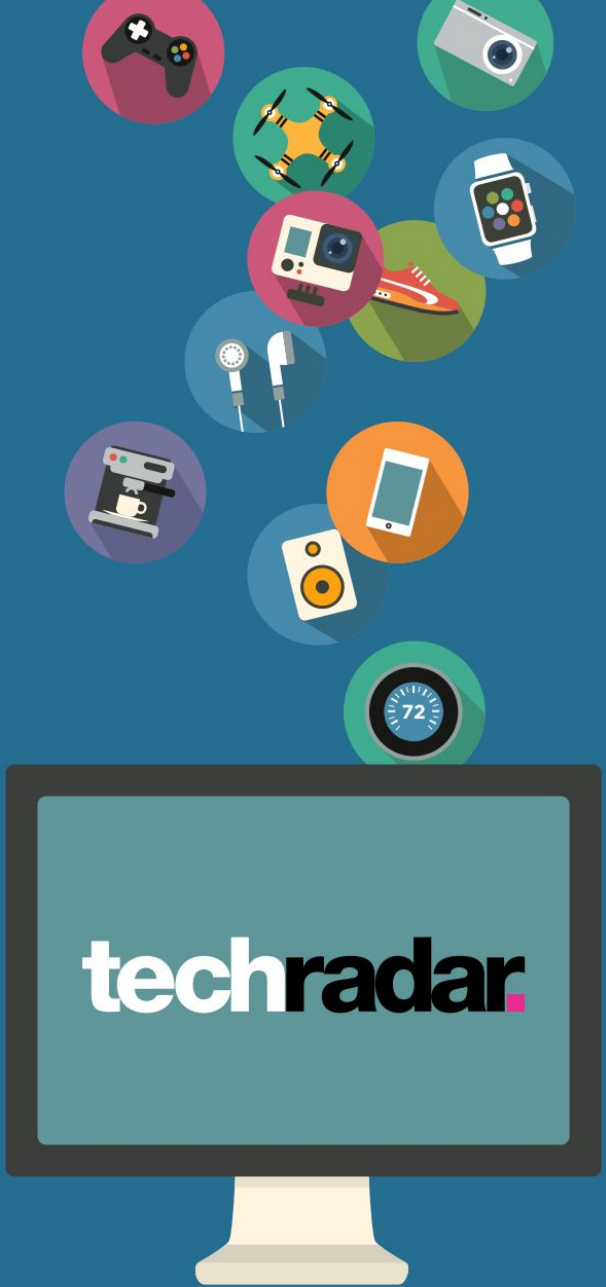
```
IMAGE_FSTYPES = "iso"
```

Rebuilding the **core-image-minimal-xfce** target will then produce the ISO file available under **build/tmp/deploy/images/genericx86-64**. You can burn this ISO file onto a hard drive or other medium to get a bootable and ready-to-use Linux system. Other image types are available as well, including ext4 and squashfs. The official Yocto documentation provides details on how to customise the root file system size and choose the right image type depending on your needs.

As for updating the installed/flashed system, there are also several paths. Although you can update the whole system at once later on, there's the Yocto-supported update tool *swupd*, which is capable of updating individual packages. It doesn't come with either the minimal image by default or the Xfce minimal image that we described above. To use *Swupd* on the target, you should attach the **meta-swupd** layer and append the **swupd-client** package to your image configuration. LXF



Poky comes with the oe-init-build-env script. Source it and gain access to the wonderful world of Bitbake build targets.



The home of technology

techradar.com

HotPicks

KDFM » Oil » Mousemic » Diskgraph » Mergerfs » Fedy » Autokey » Eggvance » URL Snake » Pacstall » Xbrzscale

Alexander Tolstoy
is here with a refreshing set of hot and spicy open sauces. No one mention juicy Tux burgers...

FILE MANAGER

KDFM

Version: GIT **Web:** https://sourceforge.net/projects/kdfm

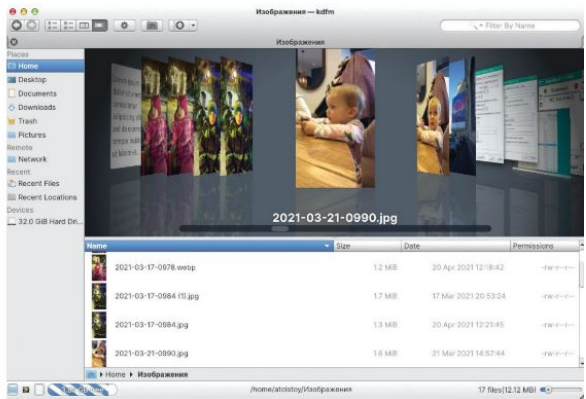
Many open source browsers advertise themselves with a 'browse with style' motto, meaning that the user benefits from the best experience when exploring the wonders of the web. But why should this be limited to online activities? Your local files and folders deserve better browsing treatment, too! With that in mind, we took *KDFM* for a test ride to see how it handles file management.

KDFM is a KDE-centric file manager designed to be an alternative to *Dolphin*. It doesn't have as many features as the latter, but it's simpler to use, cleaner and more customisable. *KDFM* is the successor of *DFM* and a side-project created by the author of the DSP (aka Styleproject) style for Qt5. It imitates Apple's Finder far better than anything else in Linux, and is the only open source software that incorporates the Flow view mode.

Whether Flow is just eye-candy or a useful tool is up for debate, but it definitely looks amazing. Browse to a folder with images or videos and press the Flow button on *KDFM's* tool bar. This mode enables you to browse items by rolling a smooth carousel, just like in Finder's Cover Flow mode. There are also Icons, Details and Columns views to choose from, which aren't available in other file managers. *KDFM* can draw thumbnails for images and videos (otherwise, Flow wouldn't make any sense), keep records of recent files and locations, access network shares, manage tabs and more.

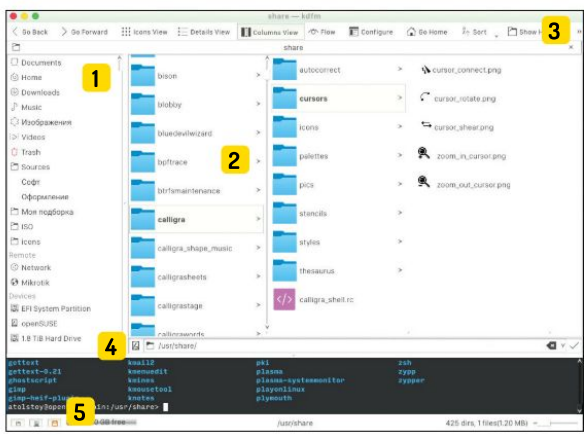
Notice the original bottom panel that's packed with extra features. The first three buttons toggle visibility for the side panel and the built-in terminal, and manage the lock mode for the Places section. The usual system widgets such as the free space bar and the scale tool are also present.

What we liked most (besides Flow) was how easy it is to rearrange parts of the *KDFM* interface, add or remove panel buttons and get the file manager looking exactly how we wanted. Choose from a minimalist window, a mission control-like station with many features available with one click, or anything in between.



KDFM is another Qt-based file manager that rocks, thanks in part to its Flow view mode.

EXPLORING THE KDFM INTERFACE...



1 Left panel
Similar to many other file managers, this area displays a list of frequently accessed places under the user's home directory, together with custom favourites.

2 Files and directories view area
Browse your files using four different view types including those unavailable in *Dolphin* or *Nautilus*. Thumbnails are supported in KDFM as well.

3 Main tool bar
Switch between views, including the outstanding Cover Flow-like view. In addition, sort files, display hidden files and access KDFM's settings from here.

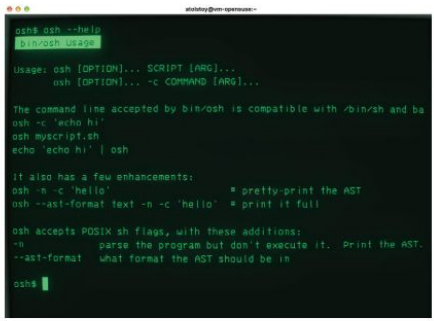
4 Location bar
Don't get lost inside the file system tree! Use the location bar below the main area to keep track on where you currently are.

5 Important KDFM switchers
These three tiny buttons show and hide the terminal panel, the left-most side bar, and also unlock the widgets in KDFM. Make the program look how you want it to!

COMMAND-LINE SHELL

Oil

Version: 0.8.9
Web: www.oilshell.org



Upgrade your terminal shell and access advanced programming features with Oil.

Everyone knows *Bash*, the classic and accessible open source shell used in Linux and elsewhere. There are other shells too, but perhaps none are as ambitious as the *Oil* shell. It's a one-person project that implements another POSIX-compatible shell that you can use in your Linux system.

Building *Oil* is a piece of cake. For those who see a command line interface as a text-based UI and a means of viewing the output of a running program, *Oil* doesn't deviate far from this definition as long as it's fully compatible with your existing *Bash* scripts. Run your commands and scripts as before, and that's it.

However, this is just one half of the *Oil* software which is known as *Oil Shell* (**bin/osh**). The other half is the Oil programming language (**bin/oil**), which is designed to supersede *Bash* in scripting. The idea is to get rid of the huge *Bash* legacy and introduce a cleaner, simpler language that would be even more feature-rich than *Bash*. To be precise, Oil implements Python-like expressions over typed data, Ruby-like blocks that support declarative configuration, extra built-in commands and a more easily understood code syntax. The bundle of OSH+OIL means that there's a way of gradually transitioning from the *Bash* syntax to the Oil syntax while not breaking existing scripts.

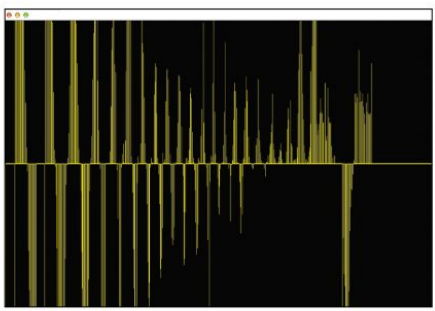
The creator of *Oil* cites large open source projects (such as Kubernetes) that include tens of thousands of *Bash* code, and that they could benefit from moving to *Oil*. The language is a work in progress, but it's considered stable and mature enough for daily use. The documentation is superb and detailed when it comes to low-level tasks, abstractions and logical differences.

Aside from the programming part, OSH is a pleasing and robust interactive shell that can already replace *Bash* in most scenarios. As an example, OSH works flawlessly with numerous *Bash* expandable commands, such as completions.

MOUSE TOOL

Mousemic

Version: GIT **Web:** https://github.com/ortegaalfredo/mousemic



Your mouse can reveal a lot. It's just a matter of finding a way to hear it...

Some things are hard to believe until you experience them in person. Are we talking about para-normal events related to Linux? Not quite, but there's a Github project that suggests using your mouse as a microphone. Interesting...

After taking a closer look at *Mousemic* we found it to be a fully functional but still slightly weird project. The idea behind the program is to detect and amplify mouse movements and output a proper spectrogram. Down and right movements draw spikes above the horizontal axis, while up and left do the same below it. It's an original and fun idea,

However, *Mousemic* clearly shows that even basic cheap mice are very sensitive. In case of an advanced mouse with higher resolution and sensitivity, *Mousemic* becomes able to detect even tiny micro-movements including vibrations. For instance, leave the mouse next to the powerful speaker and turn the volume up to feel the bass. *Mousemic* will faithfully visualise the vibrations.

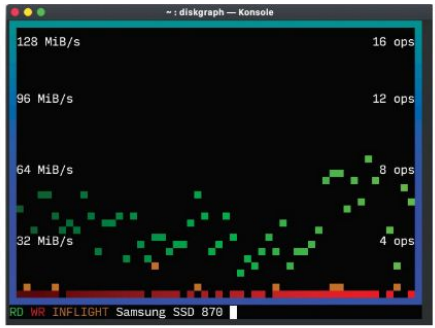
It doesn't take much imagination to think of possible useful *Mousemic* applications, including forensics and data capturing. In other words, even if the computer doesn't have a proper mic, it doesn't mean that it can't capture sounds. Not exactly real sounds (there's no tonal information), but characteristics related to sound, such as loudness and shaking should be detected. Setting X11 forwarding and accessing a remote host via SSH enables you to run *Mousemic* and find out if another user is moving the mouse.

Right now, it's not possible to restore the sound from the graph, mainly due to the fact that an average gaming mouse has a polling rate of 1,000Hz, which means we have only 1KHz sampling when capturing vibrations as audio. Regular mics have 44KHz. It's still a nice toy open source project that's fun to play with. Make sure you have the SDL2 development package in order to build *Mousemic* from its sources.

DISK MONITOR

Diskgraph

Version: GIT **Web:** https://github.com/stolk/diskgraph



Any visual representation of disk I/O always looks better than mere figures.

Monitoring hard drive performance is an important part of system admin, regardless of the actual hardware format (rotational or solid state). There are plenty of open source tools that are up to the job, but at the same time it's hard to give an example of a small, lightweight and non-obscure utility that can deliver good monitoring results without hogging system resources.

Diskgraph is one such tool. It compiles in just a few seconds into a tiny 30KB executable, which is then ready to go. Its operation is so simple that you can even omit the full path of the target drive you want to monitor and simply throw in only the last part, just like this:

```
$ diskgraph nvme0n1
```

As expected, *Diskgraph* draws a graph and puts coloured dots on it to represent read and write operations based on **/sys/block/nvme0n1/stat** data. The horizontal axis is for time, while the vertical axis is for both read/write speeds (left) and number of operations (right). The lowest part of the window displays the legend and the drive model. The latter is perhaps the most important and helpful design decision because it reminds you of the device that you're currently monitoring.

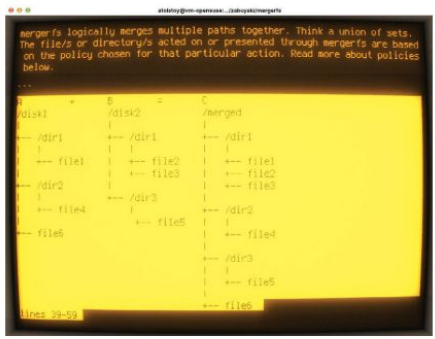
The green dots are for reading, and the red dots are for writing. Launch *Diskgraph* against one of your disks and load it with something. For instance, updating the system via its package manager spawns a wave of write operations, whereas searching for a file or running tools such as *Disk Usage* or *Filelight* are read-intensive.

Diskgraph isn't just a toy tool, of course. Use it to monitor remote servers to find out if their disk I/O is the source of slowdowns or other issues. When used over a network, *Diskgraph* consumes a lot of its bandwidth due to constant graph re-draws, although you can make the terminal window smaller to partially offset this effect.

FILE SYSTEM

Mergerfs

Version: 2.32.4 **Web:** https://github.com/trapexit/mergerfs



Mergerfs enables you to merge several mount point into one without causing any damage.

Organising data storage can be a challenge, especially if you need to bring several storage locations together and create a single-entry point for everything. There are so many ways to do that, and depending on exactly what kind of data sources you have, it may involve running a database, merging hard drive partitions into an LVM module, and more.

Step forward *Mergerfs*. This may appear to be another filesystem, but it's actually an overlay for other existing filesystems. The program unites several mounted partitions into a single mount point and merges file structures into a single tree.

There are many ways to get the most from *Mergerfs*. For instance, we can merge a read-only and a writable location in such way that the united mount point remains writable, or merge two locations and write new data to the one with more free space. *Mergerfs* has a long list of mounting options that affect new files placement, their attributes, or balancing read and write threads. Furthermore, the original mount points remain functional; *Mergerfs* merely provides a logical overlay.

Here's a real-world example of the mount command that unites three **/diskX** mount points into **/media**:

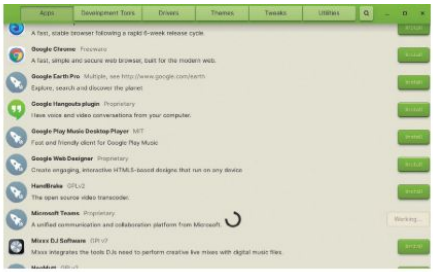
```
$ sudo mergerfs -o defaults,allow_other,use_ino,category.create=epmfs,minfreespace=1000G,moveonenospc=true,ignorepponrename=true /disk1:/disk2:/disk3 /media
```

`Allow_other` gives non-root users access and write to the united mount point, `use_ino` helps linked files share the same inode value, the `category.create` policy is set to `ExistingPathMinFreeSpace`, which tries to keep data aggregated on the same part of the union. Then we have the minimal free disk space barrier, which ensures *Mergerfs* will render the united pool full if it has as little as 1GB of free space. The last option relocates the file if the disk runs out of space or disk quota.

POST-INSTALL TOOL

Fedy

Version: 5.0.12 **Web:** https://github.com/rpmfusion-infra/fedy



Discover great third-party applications for your Fedora workstation with the help of Fedy.

This program offers customisation and post-installation tweaks for Fedora. That's right, *Fedy*'s target audience are users of this popular Linux distribution. If you carry out clean Fedora installations that require further tuning, then *Fedy* might be the perfect utility to help you in your work.

Fedy is produced and supported by the Rpmfusion project, which is well-known thanks to the same-named extra Fedora repositories. *Fedy* automates many post-installation routines for Fedora users, including extra drivers set up (wireless and Nvidia, for example), codecs, fonts, Java runtime and more.

To be frank, the golden age of *Fedy* is perhaps over these days since recent Fedora releases already include many of these missing elements (or offer the extras as Flatpaks). Still, *Fedy* hasn't entirely lost its lustre yet and it keeps bringing extra software within easy reach.

Begin by installing *Fedy* using the available Copr repo, as follows:

```
$ sudo dnf copr enable kwizart/fedy -y && sudo dnf in fedy -y
```

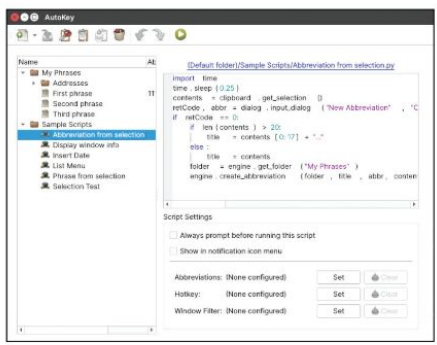
Fedy provides a great-looking GTK3-based interface with six tabs for additional programs, dev tools, drivers, tweaks, themes and utilities. You can install any item with a single click of the respective button.

It's truly amazing how many advanced software packages are readily available within *Fedy*. Looking to settle down with a development workstation? Get *VS Code*, *IntelliJ Idea* or *Sublime Text* editor within minutes (mostly spent waiting while things are being downloaded). The same approach works for any other use case. It's possible to click many Install buttons for various items and make *Fedy* put those tasks in a queue. Perhaps you may need to provide the root password in the beginning, but the rest should proceed via the 'unattended' way. Get hold of extra eye-candy, re-install *Grub2* with updated settings, clean up temporary files and enjoy a super-smooth experience with this stunning *Fedora* helper tool!

AUTOMATION TOOL

Autokey

Version: 0.95.10 **Web:** https://github.com/autokey/autokey



Autokey is a sophisticated auto-replacement text tool that will save you plenty of time in the long run.

For anyone who types a lot in Linux there's no better companion than *Autokey*. This small utility boosts typing performance by replacing user-defined abbreviations with more detailed phrases. For example, if you're working on a technical or legal document that has repeated occurrences of lengthy phrases, *Autokey* can expand your own abbreviations into those phrases. If you're coding, you can configure *Autokey* to expand your shortcuts into something more lengthy to make writing a code or a script super-fast! Let's see how it works.

Autokey is a Python3-based applications with an easy-to-use GUI. Go to the **lib** folder and issue the following command to start it up:

```
$ python3 -m autokey.gtkui
```

Replace `gtkui` with `qtui` to use the alternative interface and get ready to install some missing Python3 modules as suggested by the *Autokey* output.

The configuration window is a two-panel editor for phrases and scripts. The left panel provides some ready to use samples that help you get used to *Autokey*. To trigger a phrase expansion or a script, one should define a shortcut: choose from setting an abbreviation, assigning a hot key or defining a window properties rule. The first method leads to the auto-replace-like behaviour: short keywords are expanded automatically. The second method inserts a phrase after pressing a keyboard sequence, while the third one makes replacements inside a designated window.

The samples catalogue includes a script that can turn any sequence of characters into a phrase abbreviation without invoking *Autokey* itself. Scripts show you what Python coding looks like. There are many other possible scripting ideas that may strike your fancy, such as keyboard layout switching or launching a third-party program.

GAME CONSOLE EMULATOR

Eggvance

Version: 1.0 **Web:** https://github.com/jsmolka/eggvance

Linux is a very friendly platform for retro gaming thanks to the abundance of high-quality emulators. This time we're rediscovering the world of Gameboy Advance emulators that will enable you to play your favourite (legal) ROMs without too much effort.

Back in **LXF245** we featured *Mednaffe*, a GUI to the powerful *Mednafen* emulator that supports a range of consoles in one bundle. This issue's *Hotpick* is *Eggvance*, which is another console emulator. It's a small project that focuses solely on Nintendo's Gameboy Advance. It's robust, compact and straightforward to set up. Furthermore, it's claimed to be the most accurate emulator in that it passes more CPU tests than its competitors. While some of those can be 'edge cases', they deliver more genuine Gameboy Advance emulation than you might expect.

The project is easy to compile from source using a modest set of dependencies, mainly SDL2 and GTK3. No preconfiguration is required because *Eggvance* already comes with a built-in console BIOS stub.

Relive and enjoy pure gaming nostalgia thanks to this Gameboy Advance emulator.

Therefore, it all comes down to finding a game ROM to put *Eggvance* through its paces.

Once you fire up *Eggvance* you'll notice the thin top menu bar within its window. It's here where you can load a game (.GBA) and access various settings: set a game's save path, change keys, adjust audio/video quality and so on. It doesn't take much to scale up old games graphics to make them better playable on high-resolution screens, mute sounds or selectively disable certain audio channels.

Eggvance has recently reached the 1.0 milestone with dozens of reported bugs successfully fixed and closed. We found that the emulation quality was excellent. All of our GBAs worked flawlessly with all their sounds, intro animations and graphics. Want a genuine-level Gameboy Advance emulation on Linux? Then get started with *Eggvance* today!

ARCADE GAME

URL Snake

Version: GIT **Web:** https://github.com/epidemian/snake

We've seen lots of unusual arcade, casual and action games, but *URL Snake* definitely beats them all in terms of eccentricity. It's a browser-based game that's yet another clone of the *Snake* game. It's dead simple, classic and familiar to anyone who's used a simple push-button feature phone.

The ground-breaking difference with *URL Snake* is in the fact that you play it in the browser, but not inside its window. Instead, *URL Snake* implements the game inside a browser's URL address bar. It's a clever Javascript-powered hack that turns that tiny little space inside the URL area into a playing field. It's obviously not a square, but rather an outstretched rectangle limited by its longer side to be the same regardless of the client's resolution or browser window size. It's just the 20x4 field made of dots-like Braille symbols.

URL Snake behaves differently depending on your web browser and its default approach to showing a full address in the URL area (without shortening or truncating it). If *URL Snake* doesn't behave correctly for you, press the '?' link below to emulate the URL bar

Those dots were actually too small to see without zooming in...

inside the main page area. After that, use arrow keys or WASD keys to control the snake and help it grow.

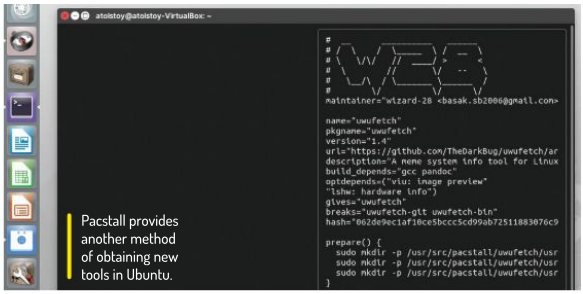
The game wants you to collect extra spots in order to make the snake longer. Don't hit the walls or your own tail, otherwise you'll lose. If you do, *URL Snake* will auto-restart without notice. Your high score is recorded and displayed next to the playing field, on the right. If you feel like you have a score that's worth shouting about, feel free to hit the Share button and let other people know.

There's a strong temptation to enjoy the game's public instance at **https://demian.ferrei.ro/snake**, but you can easily host the game on your own web server. *URL Snake* is a fun way to pass the time and a perfect alternative to Google's *Dino Run* or similar browser Easter-egg-like games designed to entertain you if your internet connection stops working.

SOFTWARE MANAGER

Pacstall

Version: 1.2 **Web:** https://github.com/Henryws/pacstall



Pacstall provides another method of obtaining new tools in Ubuntu.

Pacstall is an AUR clone for Ubuntu users. That said, if you acknowledge the benefits of the AUR build system used in Arch Linux and its derivatives but don't want to leave the beloved Ubuntu installation, take a closer look at *Pacstall*. This is an attempt to bring the AUR package management approach to Ubuntu. Of course, right now the list of available packages is limited compared with the huge number available within the AUR abyss, but it's already worth exploring. *Pacstall* stores package recipes in . pacscript files that are similar to PKGBUILDs, although the two are not directly compatible.

As a brief reminder, the AUR approach means that each open source software title, for which the repository has the description and the building recipe, is fetched and compiled at runtime. Ideally, you don't need to browse individual Github or Gitlab repositories, because AUR already has everything it needs to build that software for you. Currently, the above applies to such titles as *Alacrity*, *Neofetch*, *Wine*, *Discord*, Gnome-tweaks, a bunch of GTK themes and more. There are around 40 programs, and we do expect the catalogue to grow. Install *Pacstall* with:

```
$ sudo bash -c "$(wget -q https://git.io/JfHDM -O -)"
```

If you don't trust unfamiliar *sudo* commands *Pacstall* also has the DEB package linked in the **Readme.md**. The syntax of frequently used *Pacstall* commands resembles *Pacman* commands used in Arch Linux. For instance, search for themes with

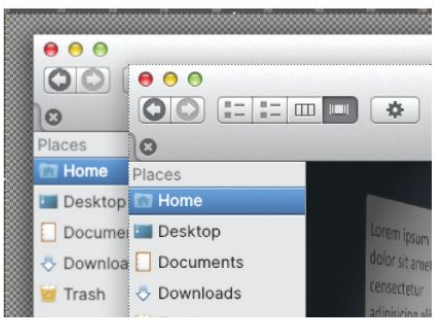
```
$ pacstall –S theme
```

Pacstall will then show search results and their . pacscript recipes in a separate terminal panel. Choose something and hit Enter, and *Pacstall* does the rest. By default, it'll install build dependencies, fetch and compile the build target, and install it. It's also possible to purge those build dependencies to keep the system uncluttered.

SCALE ALGORITHM

Xbrzscale

Version: GIT **Web:** https://github.com/atheros/xbrzscale



Xbrzscale is a better way of improving an image quality with very little effort.

Scale up a low-resolution image and it'll become either blurred or pixellated, which is unsurprising because you can't restore missing details out of nowhere... or can you? Using a well-trained neural network known as Generative Adversarial Nets (GAN) and the glorious *Waifu2x* upscaling software featured in **LXF243** you could, but this is beyond the scope of *Hotpicks*!

Instead, we'll lower the upscaling science barrier at the cost of slightly simpler results. *Xbrzscale* is a high-quality software that can help you achieve substantially better results when trying to upscale drawn artwork, anime or pixel art, screenshots and icons. The tool implements the xBR algorithm that sports pattern recognition and smart interpolation rules. The enlarged image will feature well-defined curves and contrast edges, which isn't possible with classic or bicubic upscaling. xBR is available as a third-party plug-in for *Gimp* (**https://github.com/bbbbbr/gimp-plugin-pixel-art-scalers**), but *Xbrzscale* enables you to do the job without having *Gimp* at hand.

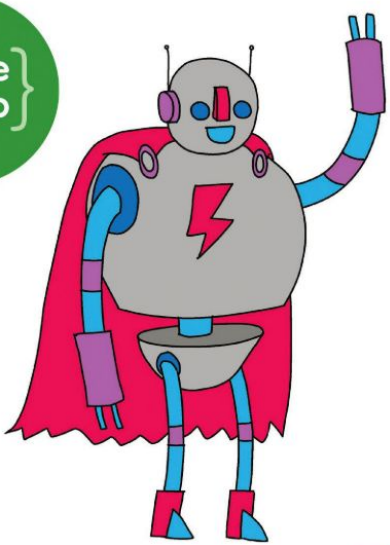
The program is easy to compile and run, provided you have the development packages of SDL2 and SDL2-image installed. To process an image, use the following syntax:

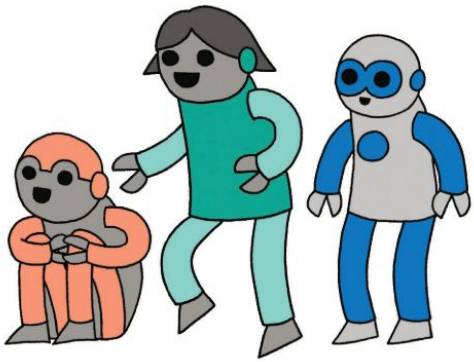
```
$ xbrzscale X input.image output.image
```

Replace X with the desired scale factor from 2 to 6. The amount of time required for completing all calculations depends on the size and type of the source image. Obviously, smaller images are processed faster, but PNG icons and screen grabs are also faster to upscale than photographic JPG images.

Xbrzscale supports a variety of bitmap images, just as the SDL2-image library does. The results are very good, especially if you're upscaling pixel-based art. *Xbrzscale* is a lightweight CLI utility that's easy to use in scripts, remote servers and more. LXF

{code club}





Can you help inspire the next generation of coders?

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

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

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

There are loads of ways to get involved!

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

RABBITMQ

Exchange messages between tasks

The code
Get it from linuxformat.com/archives

Sending messages and rabbits are **Mihalis Tsoukalos** favourite things, so he's in Python heaven explaining how to use both.

OUR EXPERT

Mihalis Tsoukalos is a systems engineer and a technical writer. He's also the author of Go Systems Programming and Mastering Go.

RabbitMQ is an open source message broker. It's ideal for exchanging information asynchronously, when you need a place where messages can be stored safely until they're read. *RabbitMQ* enables you to exchange information, which is the main reason that you don't need to use it directly unless you have to perform administrative tasks. As a result, this tutorial will show you how to develop command line utilities in Go and Python that interact with *RabbitMQ*.

You can install *RabbitMQ* using your favourite Linux package manager. However, this tutorial is going to use *RabbitMQ* with the help of a Docker image, which is the recommended method of running *RabbitMQ* because it's portable and easy to configure.

You can obtain a *RabbitMQ* Docker image by executing `docker pull rabbitmq:3.8-management`. Although *RabbitMQ* doesn't come with a graphical interface (because you don't usually interact with it directly), this particular Docker image comes with the *RabbitMQ* management plugin (**www.rabbitmq.com/management.html**) installed, which gives you extra management capabilities through an informative and easy-to-use user interface. We'll cover the management plugin in the next section.

Docking complete

The best way to use the *RabbitMQ* Docker image is by creating a **docker-compose.yml** file and setting it up appropriately. The following output shows the *RabbitMQ* configuration found in **docker-compose.yml** that's used in this tutorial.

```
rabbitmq:
  image: 'rabbitmq:3.8-management'
  container_name: rabbit
  ports:
   - '5672:5672'
   - '15672:15672'
  environment:
   AMQP_URL: 'amqp://rabbit?connection_
attempts=5&retry_delay=5'
   RABBITMQ_DEFAULT_USER: "guest"
   RABBITMQ_DEFAULT_PASS: "guest"
  networks:
   - rabbit
```

This shows a screen from the user interface of the RabbitMQ Management plugin. This give you a decent overview of your RabbitMQ installation and enables you to perform administrative tasks.

The previous configuration sets both the default *RabbitMQ* username and password to guest and "sets" the AMQP URL to **amqp://rabbit?connection_attempts=5&retry_delay=5**. By default, *RabbitMQ* listens to port number 5672. Additionally, the management plugin listens to port number 15672, which means that you can access it at **http://localhost:15672/** from the machine that you executed the `docker-compose up` command.

You can now run `docker-compose up` to run the configuration described in **docker-compose.yml**. After that, you can connect to the running *RabbitMQ* container by running `docker exec -it rabbit bash`.

The screenshot (*above*) shows the UI of the management plugin that comes with the **rabbitmq:3.8-management** Docker image. Feel free to play around with the various settings.

The use of the *RabbitMQ* REST API can also give you similar information. As an example, you can find the names of the available queues with the help of *curl* by running `curl -s -i -u guest:guest http://localhost:15672/api/queues` – the previous command generates JSON output, which is the case with most REST APIs.

Next, we're going to interact with *RabbitMQ* using a small Go utility (**hwRabbitMQ.go**) that just connects to

the *RabbitMQ* Docker container – this is a proof of concept that everything works well with the *RabbitMQ* installation and that external producers and consumers can connect to the *RabbitMQ* instance. If you experience problems with your *RabbitMQ* installation, either when using Docker or a regular installation, start from this code to find out what's happening.

The **github.com/streadway/amqp** Go module does the job of connecting to *RabbitMQ* using the AMQP protocol, provided that you supply the correct connection details. Because we're working with a recent Go version (1.16.4, which is the latest Go version at the time of writing), we need to put the source code under **~/go/src**, which in this case is **~/go/src/github.com/mactsouk/rabbitmq** and its subdirectories, and enable Go modules before being able to run it. This is also a requirement for all Go code mentioned in this tutorial.

Enabling Go modules and running **hwRabbitMQ.go** produces the following output:

```
$ cd ~/go/src/github.com/mactsouk/rabbitmq/hw
$ go mod init # This is where we define that we want to use Go modules
$ go mod tidy # This is where the amqp module is being downloaded
$ go run hwRabbitMQ.go amqp://guest:guest@localhost:5672/
Using default connection string
Successfully Connected to our RabbitMQ Instance
```

The connection string (`amqp://guest:guest@localhost:5672`) used as a command line parameter can be split into three parts. The first part (`amqp://`) is the protocol that's used, the second part contains the user credentials and the third part is the server name and port number (`localhost:5672`).

The important part of **hwRabbitMQ.go** is the `conn, err := amqp.Dial(connectString)` statement. So, given a connection string with the details of the *RabbitMQ* server, `amqp.Dial()` tries to connect to the server – this statement is going to be used again in this tutorial. If the *RabbitMQ* server either can't be found at the provided address or the port number value is erroneous, then **hwRabbitMQ.go** will generate an error message. It'll look something like `dial tcp 127.0.0.1:52: connect: connection refused`.

Command line tools

As mentioned earlier, *RabbitMQ* comes with a number of command line utilities that enable you to configure and administer *RabbitMQ*. The list of utilities includes: *rabbitmqctl*, which is used for administrative tasks; *rabbitmq-plugins*, which is used for administrating plugins; *rabbitmq-diagnostics*, which is used for finding information about the status of a RabbitMQ server; *rabbitmq-queues*, which is used for working with queues; and *rabbitmq-upgrade*, which can be used for tasks related to upgrades.

The screenshot (*above*) shows the *rabbitmq-plugins* and *rabbitmq-diagnostics* command line utilities in action. If you manually install RabbitMQ, you can expect to have the same utilities installed on your local machine instead of the Docker container.

Now that we know the basics of *RabbitMQ* and how to use some of the command line utilities that come with utility, the next step is to begin developing utilities that write and read from *RabbitMQ* queues.

This shows the rabbitmq-plugins and rabbitmq-diagnostics command line utilities in action.

We begin by developing a simple *RabbitMQ* producer in Go named **producer.go.** Provided that we already know the use of `amqp.Dial()`, as discussed earlier, then the important part of **producer.go** are the following statements:

```
ch, err := conn.Channel()
q, err := ch.QueueDeclare("LXFQueue", false, false, false, false, nil)
err = ch.Publish("", "LXFQueue", false, false,
        amqp.Publishing{ContentType: "text/plain",
Body: []byte(message)},
)
```

First of all we define a channel to the *RabbitMQ* instance using the connection that we've already established. AMQP connections are multiplexed with channels – this means that lots of channels share the same AMQP TCP connection for efficiency.

Next, we define and create the queue we want, which in our case is called **LXFQueue**, to publish messages using `ch.QueueDeclare()`. The `QueueDeclare()` method accepts lots of parameters – in our case the important parameter is the first one that defines the name of the *RabbitMQ* queue we want to use.

Additionally, we define the format of the data we send in the amqp.Publishing{} structure, which in this case is plain text. Finally, we actually send the data to

QUICK TIP

A distributed event streaming platform such as Kafka allows you to send messages between processes, applications and servers. Applications connect to Kafka to send or get data. You can learn more about Kafka by visiting https://kafka.apache.org and www.confluent.io.

» AN INTRODUCTION TO RABBITMQ

RabbitMQ is a Message Broker, which means that it can safely hold messages generated by applications and make them available to other applications. The main advantages are reliability, support for clustering and high-availability queues, tracing capabilities for troubleshooting tricky issues, support for protocols other than AMQP and the availability of tools and client libraries in many programming languages. *RabbitMQ* has no tlimit on the payload size, which is important when you have to exchange large messages. The two main disadvantages of *RabbitMQ* include the fact that it's written in Erlang, which might discourage some people from extending it, and that it doesn't support message batching.

If you have to choose between *RabbitMQ* and *Kafka* (see **LXF252**), which can both handle message exchanges, you should begin by considering their differences. First, *Kafka* is faster than *RabbitMQ*. Second, *RabbitMQ* works with the push model whereas *Kafka* works with a pull-based approach. Third, *Kafka* offers support for batching whereas *RabbitMQ* doesn't. Finally, *RabbitMQ* doesn't have a limit on the payload size whereas *Kafka* has some restrictions on the payload size. If speed is your main concern, then *Kafka* might be a better choice whereas if payload size and simplicity are your main concerns, then *RabbitMQ* is the more sensible choice.

the desired *RabbitMQ* queue by calling `ch.Publish()` – the data is kept in the **message** variable, which is converted into a Go byte slice, and is passed to the Body field of the amqp.Publishing{} structure.

Note that the job of **producer.go** is to send a single message, which in this case is "Linux is 30 years old!", to a *RabbitMQ* queue and nothing more. All messages are going to remain in the *RabbitMQ* queue until they're read (consumed) by a consumer. The next section explains the development of a *RabbitMQ* consumer in Go.

Consumers, Go!

With the producer in place we can move development on to a *RabbitMQ* consumer, again in Go. Both the name of the *RabbitMQ* queue to read messages from and the *RabbitMQ* server address are hardcoded. The important part of **consumer.go** is the following:

```
msgs, err := ch.Consume("LXFQueue", "", true, false,
false, false, nil)
go func() {
        for d := range msgs {
                fmt.Printf("Received: %s\n", d.Body)
        }
}()
```

There are two important things happening here. First, we declare the *RabbitMQ* queue we want to read from, and second, we create a goroutine that keeps waiting for messages from a Go channel that's returned by the Consume() method until that channel is closed for some reason. Goroutines and channels are the fundamental elements of the Go Concurrency model.

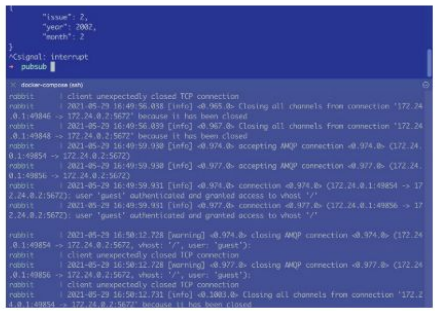
The screenshot (*below*) shows some interactions between **consumer.go** and **producer.go** using two terminal windows as well as the way to enable Go modules. Although in our case both the producer and the consumer reside on the same Linux machine, in real-world applications they can reside anywhere you want. That's the beauty of *RabbitMQ* – it doesn't care about the details of the producer or the consumer. It just stores and delivers its messages reliably.

Prosumer?

Let's go crazy and create a Go program that's both a producer and a consumer. The trick behind that code is the use of Go routines that work independently from each other. Additionally, this version of the program communicates using the JSON data format, which is a popular and widely used format for data exchange. This

```
→  produce ll
total 4.0K
-rw-r--r-- 1 mtsouk users 746 May 29 18:41 producer.go
→  produce go mod init
go: creating new go.mod: module github.com/mactsouk/rabbitmq/produce
go: to add module requirements and sums:
        go mod tidy
→  produce go mod tidy
go: finding module for package github.com/streadway/amqp
go: found github.com/streadway/amqp in github.com/streadway/amqp v1.0.0
→  produce go run producer.go
Linux Format RabbitMQ Tutorial
Queue: {LXFQueue 0 0}
Message published to Queue!
→  produce go run producer.go
Linux Format RabbitMQ Tutorial
Queue: {LXFQueue 1 0}
Message published to Queue!
→  produce go run producer.go
Linux Format RabbitMQ Tutorial
Queue: {LXFQueue 0 1}
Message published to Queue!
→  produce go run producer.go
```

An interaction between producer.go, which writes messages to LXFQueue, and consumer.go, which reads messages from the LXFQueue queue.



Here's the pubsub.go utility in action, which implements both a RabbitMQ producer and consumer. The bottom half of the screen shows the log messages generated by the RabbitMQ container.

is a utility that can be used either for testing the operation of a *RabbitMQ* server, or highlight any communication problems between the current machine and the *RabbitMQ* server machine you want to use.

When working with JSON data in Go, you need to serialise (known as 'marshalling' in Go terminology) the JSON data before sending it and deserialise ('unmarshalling' in Go terminology) it after receiving it. In more detail, marshalling is the process of converting a Go structure into a JSON record. You usually want that for transferring JSON data via computer networks or for saving it on disk. Unmarshalling is the process of converting a JSON record given in plain text format as a Go byte slice into a Go structure. You usually want that when receiving JSON data via computer networks or when loading JSON data from disk files.

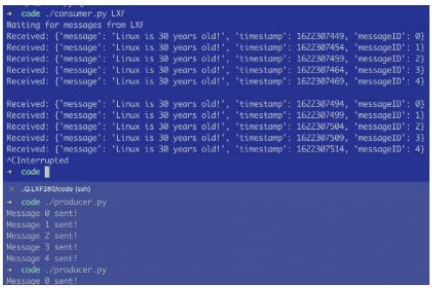
Both processes require the use of a Go structure that contains extra JSON tags and is used for holding JSON records. This is the way Go works with JSON data. The definition of the Go structure, which should reflect the format of the JSON record used, is as follows:

```
type LXF struct {
        ID    int `json:"issue"`
        Year  int `json:"year"`
        Month int `json:"month"`
}
```

The JSON tags specify the names of the fields used in the JSON record. But you might ask, why not have the same field names for both the Go structure and the JSON record? The answer to this is simple: Go follows a simple rule which states that functions, variables, data types, structure field names and so forth that begin with an uppercase letter are public, whereas functions, variables, types and so on that begin with a lowercase letter are private.

This approach might have caused problems, especially when a Go structure is defined in a different package than the one used. As a result, Go uses JSON tags for solving this issue. Have in mind that JSON tags can do many more things than just translating a Go structure field name to the desired JSON field name and vice versa. The Go functions used for marshalling and unmarshalling JSON data are **json.Marshal()** and **json.Unmarshal()**, respectively. As you'll soon see, the same is true for Python.

The important code of **pubsub.go** that implements the producer part is the following:



This program shows consumer.py and producer.py in action when exchanging JSON data. Notice that nothing stops you from using consumer.py with producer.go or consumer.go with producer.py.

```
for {
        t := LXF{ID: i, Year: year, Month: i % 12}
        body, err := json.Marshal(t)
        if err != nil {
                fmt.Println("Error in Marshalling:",
err)
                continue
        }

 err = ch.Publish("", "LXFQueue", false, false,
                amqp.Publishing{ContentType:
"application/json", Body: []byte(body)})
...
}
```

The previous code shows how a new structure is constructed before it's serialised with `json.Marshal()`. Additionally, the value of `ContentType` is now `application/json` to indicate that we're working with JSON data. Similarly, the important code that implements the consumer part is the following:

```
msgs, err := ch.Consume("LXFQueue", "", true, false,
false, false, nil)
for d := range msgs {
        data := LXF{}
        err = json.Unmarshal(d.Body, &data)
        _ = PrettyPrint(data)
}
```

The consumer part needs to read the data from the *RabbitMQ* queue, which as before takes place in the for loop with the use of the Go channel that's returned by the `Consume()` method, and process it with `json.Unmarshal()`.

The screenshot (*top of the previous page*) shows the output generated by **pubsub.go** when interacting with it as well as the log entries generated by the Docker image. The `PrettyPrint()` helper function is responsible for the printing of the JSON records.

So far, we've learned how to work with *RabbitMQ* in Go. The list of programming languages supported by RabbitMQ includes, but isn't limited to, Python, Java, Ruby, Elixir, PHP and Swift. The next section shows how to work with RabbitMQ in Python.

Pythons love Rabbits

Let's look at developing a *RabbitMQ* consumer and producer in Python. Because JSON is so popular, the presented code is working with the JSON data format. As it happened with the Go version, we need to serialise the data before sending it and deserialise it upon receiving it. But first, we need to install the pika Python module (**https://pika.readthedocs.io/en/stable**), which offers an implementation of the AMQP 0-9-1 protocol, by running `pip3 install pika` with administrative privileges. The important Python code of the consumer, named **consumer.py**, is the following:

```
connection = pika.BlockingConnection(pika.Connectio
nParameters(host='localhost'))
channel = connection.channel()
channel.queue_declare(queue=queue)
```

This code connects to *RabbitMQ*. The code assumes the server runs on **localhost**, creating a channel and declaring the queue that's going to be used.

The important code of the producer, which is named **producer.py**, is the following:

```
channel.queue_declare(queue='LXF')
channel.basic_publish(exchange='', routing_key='LXF',
   body=json.dumps(message), properties=pika.
BasicProperties(delivery_mode = 2,))
```

The first statement defines the name of the queue that's going to be used, whereas the second statement sends the desired message to *RabbitMQ* in JSON format after serialising it using `json.dumps()`.

The final screenshot (*at the top of this page*) shows an interaction between **consumer.py** and **producer.py**. The producer process ends after sending five messages to the queue whereas the consumer process keeps waiting for more messages.

RabbitMQ has more capabilities than the ones presented here and is a rational choice when you need a message broker to work with. Although it offers advanced capabilities such as clustering, it can also work in simple scenarios. As usual, you need to spend some time learning and understanding *RabbitMQ* before you use it in production. Happy learning! LXF

QUICK TIP

You can learn more about RabbitMQ at www.rabbitmq.com and at https://github.com/rabbitmq. The command line utilities of RabbitMQ are presented in more detail in www.rabbitmq.com/cli.html. And, you can find the RabbitMQ documentation site at www.rabbitmq.com/documentation.html and lots of practical RabbitMQ tutorials at www.rabbitmq.com/getstarted.html.

» RABBITMQ TERMINOLOGY

A message broker such as *RabbitMQ* acts as a middleman for software to exchange information using one or more queues for storing the messages. Usually, we don't interact directly with *RabbitMQ*. A client sends data to *RabbitMQ*, which in *RabbitMQ* terminology is called a producer, and another program reads these messages, which in *RabbitMQ* terminology is called a Consumer. So, a producer is any application that sends messages whereas a consumer is any application that receives messages from the queue.

A queue is a FIFO (first in, first out) structure that supports two operations: adding elements and getting elements.

The AMQP protocol, which stands for advanced message queuing protocol, is a open protocol for message-oriented middleware. The characteristic features of AMQP are message orientation, queuing, routing, reliability and security. AMQP works with binary data and transmits data in frames. There are nine types of frames depending on the task (open, close, transfer data, etc.) that you can perform. You can find more information about AMQP at **www.amqp.org** and at **https://en.wikipedia.org/wiki/Advanced_Message_Queuing_Protocol**.

ASYNCIO

Create asynchronous code with Python

Prepare to multitask in ways you've never done before, as **Mihalis Tsoukalos** explains the uses of the asyncio Python module.

OUR EXPERT

Mihalis Tsoukalos is a systems engineer and a technical writer. He's also the author of Go Systems Programming and Mastering Go.

Say that you have a web page that needs to load data from different servers. Now imagine that some of these servers aren't responding immediately, which delays the loading of the entire web page. Not good, right? Now, imagine that you're writing a Python script that needs to interact with multiple machines/servers. Do you want your entire script to wait for a specific server to become available, instead of continuing with its operation while keeping an eye on that server? Generally speaking, it's always better to do something than waiting for something else to become available, because it's how you take a better advantage of your available resources.

On a web page, you can solve that issue by writing asynchronous JavaScript code. In Python, the answer is asynchronous programming. This tutorial shows how to write asynchronous code in Python 3 with the asyncio module, a library for writing concurrent code that follows the async/await paradigm. Asyncio has been part of the Python standard library since Python version 3.4; however, this tutorial assumes that you're using Python version 3.8 or newer.

Because asyncio contains some advanced concepts and features, we'll cover some simple working examples. Once you understand them, you can modify them and use them for solving your own problems.

QUICK TIP

The official documentation site of asyncio can be found at https://docs.python.org/3/library/asyncio.html. You can also learn more information about asyncio at https://docs.python.org/3/library/asyncio-dev.html and at https://realpython.com/async-io-python.

Introducing asyncio

It's important to understand the purpose of the asyncio library, which is to have only one thread run an event loop. In this loop, which is the core of every asyncio application, tasks are switched when the programmer writes the `await` keyword. So, it's the developer who decides when to switch tasks instead of the operating system scheduler, based on the `await` signals.

As a result, asyncio isn't efficient for CPU intensive tasks because it doesn't run multiple threads in parallel. It's better suited for asynchronous input and output operations such as working with network sockets, because of how sockets are implemented in Python (they're written in C and use different resources than Python). In this tutorial we simulate delays with calls that put a function to sleep.

```
Sequentially:
Task 1 done
Task 2 done
Task 3 done
Task 4 done
Task 5 done
Task 6 done
Task 7 done
Task 8 done
Task 9 done
Non sequentially:
Task 1 done
Task 2 done
Task 3 done
Task 6 done
Task 4 done
Task 5 done
Task 9 done
Task 7 done
Task 8 done
```

```
Sequentially:
Task 1 done
Task 2 done
Task 3 done
Task 4 done
Task 5 done
Task 6 done
Task 7 done
Task 8 done
Task 9 done
Non sequentially:
Task 4 done
Task 5 done
Task 6 done
Task 1 done
Task 3 done
Task 7 done
Task 8 done
Task 2 done
Task 9 done
```

This is the output of order.py that illustrates sequential and asynchronous execution of functions and coroutines, respectively. Coroutines are executed in a random order.

Because asyncio is part of the Python 3 library, there's no need to install additional libraries to use it. Just include a `import asyncio` statement in your Python 3 code and you're ready to go.

Before continuing with the use of asyncio, let's cover some common coding topics. Asynchronous operations return immediately with a promise that they're going to deliver their results when they're ready. An object can be called awaitable if it can be used in an `await` expression. The three main types of awaitable objects are Coroutines, Tasks and Futures.

Coroutines are a more generic form of subroutines. Although subroutines are entered at one point and exited at another point, coroutines can be entered, exited and continued at multiple points and are implemented with `async def` statements.

You can consider the following code, which is saved as **asyncIOhw.py**, as the Hello World version of asyncio:

```
import asyncio
async def main():
    print('Hello ...')
    await asyncio.sleep(1)
    print('... World!')
asyncio.run(main())
```

Bear in mind that asyncio has its own special version of `sleep()`, which is called as `asyncio.sleep()`, which is

non-blocking. In practice, this means that `asyncio.sleep()` suspends the current task, the one that called `asyncio.sleep()`, but enables all other tasks to operate. Put simply, `asyncio.sleep()` doesn't block the entire Python script.

Our Hello World code prints "Hello ...", waits one second because of the `asyncio.sleep(1)` call and then print "... World!". How does this happen? We create the `main()` function that we run by calling `asyncio.run()`.

Order of execution

The code here is clear that you should not make any assumptions about the order of execution of the coroutines when using asyncio. This is also a principle of concurrent and parallel programming. This characteristic is presented in **order.py** – the most important code in **order.py** is the following:

```
async def task_coroutine(pid):
    await asyncio.sleep(random.randint(0, 2))
    print('Task %s done' % pid)
async def concurrently():
    tasks = [task_coroutine(i) for i in range(1, 10)]
    await asyncio.gather(*tasks)
asyncio.run(concurrently())
```

We have three important things happening here. First, we create a coroutine named `task_coroutine()` that sleeps for a random amount of time. Second, we have the definition of a second coroutine named `concurrently()` that generates ten `task_coroutine()` coroutines. Their results are collected using `asyncio.gather()`. Finally, we have the execution of `concurrently()` using `asyncio.run()`. Any function that is defined as asynchronous should be executed using the functionality of the asyncio library.

The screenshot (*see left, page 92*) shows the output of **order.py** when it's executed twice. The sequential execution generates the same output every time because functions are executed in the same (sequential) order, whereas the asynchronous execution generates a different output each time because coroutines are executed in a random order.

Processing multiple values

Let's run an asyncio example where we create multiple coroutines and collect their return values – the name of the script is **results.py**. The screenshot (*above*) shows the Python code of **results.py**.

So what's happening here? The single most important thing is the use of an event loop for processing, which is created using a call to `asyncio.get_event_loop()`. Second, the use of `run_until_complete()` in order to execute the required amount of runners that are going to perform the desired jobs. The parameter of `runner()` is a list of integers. Then we go to the `runner()` coroutine, which is the one that executes the required number of `fibo()` coroutines and collects the results. The list of coroutines is created with the help of `asyncio.create_task()`. The parameter of `asyncio.wait()` is the list of coroutines that are about to get executed – the return values of `asyncio.wait()` are two futures (futures are presented in the next section). Once the processing is done, the results are collected using the `results = [res.result() for res in done]` statement.

The output of **results.py** is as follows:

```
$ ./results.py
```

```
async def fibo(value):
    if value < 0:
        raise "Negative number"

    a, b = 0, 1
    for i in range(2, value+1):
        a, b = b, a+b

    return b

async def runner(values: List[int]):
    requests = [asyncio.create_task(fibo(number)) for number in values]
    done, pending = await asyncio.wait(requests)

    results = [res.result() for res in done]
    print('Total results: {}'.format(len(results)))
    print('Pending results: {}'.format(len(pending)))
    print(sorted(results))

if __name__ == '__main__':
    loop = asyncio.get_event_loop()
    loop.run_until_complete(runner([1, 10, 2, 3, 4, 15, 6]))
    loop.close()
```

This is the code of results.py where we execute and collect the results from multiple coroutines. Each coroutine calculates a number of the Fibonacci sequence, but you can make do anything you want.

```
Total results: 7
Pending results: 0
[1, 1, 2, 3, 8, 55, 610]
```

The last line shows the sorted output (because the order of execution can't be determined) whereas the first line of output prints the total number of results that have been collected. The second line of output shows that we have 0 pending results.

Tasks and futures

A task is a way of scheduling a coroutine for execution. On the other hard, a future is an indirect reference to a forthcoming result. The good thing is that you can ask a future to send back its response when it's ready. Tasks are illustrated in **tasks.py**, which can be seen in the screenshot (*page 94*). Futures are illustrated in **futures.py**. The important code of **futures.py** is the following:

```
async def main():
    future = asyncio.Future()
    await asyncio.ensure_future(fibo(future, 10))
```

» CONCURRENCY VS PARALLELISM

It's a common misconception that concurrency is the same thing as parallelism. Parallelism is the simultaneous execution of multiple entities of some kind, whereas concurrency is a way of structuring your components so that they can be executed independently.

It's only when you build software components concurrently that you can safely execute them in parallel, when and if your operating system and your hardware permit it. The Erlang programming language (**www.erlang.org**) did this a long time ago – long before CPUs had multiple cores and computers had lots of RAM.

In a valid concurrent design, adding concurrent entities makes the whole system run faster because more things can be executed in parallel. So, the desired parallelism comes from a better concurrent expression and implementation of the problem. The developer is responsible for taking concurrency into account during the design phase of a system and will benefit from a potential parallel execution of the components of the system. So, the developer shouldn't think about parallelism, but rather about breaking things into independent components that solve the initial problem when combined.

Even if you can't run your functions in parallel on a UNIX machine, a valid concurrent design still improves the design and the maintainability of your programs. In other words, concurrency is better than parallelism!

QUICK TIP

A program's development process should take into account the use of asyncio in order to properly design the application. The extra care and extra code that you write is the price you pay for faster execution time. Bear in mind that not everything can be executed concurrently

```
print(future.result())
```

There are three things happening here. First, we define a future object. Then, the `asyncio.ensure_future()` call converts the `fibo()` coroutine into a future object. Last, we print the result of the future using `print(future.result())`. Bear in mind that `ensure_future()` starts running the coroutine, which is passed as a parameter, as soon as the event loop is resumed.

What's not seen here is that `fibo()` returns the desired value using a `future.set_result()` call – this is the best way to return values when using a future. Please see the full code of **futures.py** for more details.

You can see the output from both **tasks.py** and **futures.py** as well as the code of **tasks.py** in the screenshot (*below*). The **futures.py** script returns a single result whereas **tasks.py** executes multiple tasks using `asyncio.create_task()`. Once again, the result of a task can be received using the `result()` method.

Fibonacci time!

We return to a classic, calculating numbers of the Fibonacci sequence, but concurrently using `asyncio.gather()`, which is responsible for collecting all results. `asyncio.gather()` waits until all the awaitable objects are finished and then retrieves the results. The important code of **fibo.py** is the implementation of `main()` that contains the `asyncio.gather()` call:

```
async def main():
  result = await asyncio.gather(
    fibo("A", -1),
    fibo("B", 3),
    fibo("C", 4),
    fibo("D", 5),
    fibo("E", 6),
    return_exceptions=True
  )
  print(result)
```

Here we're scheduling five function calls to be executed concurrently. The `asyncio.gather()` function expects awaitables to be passed as positional arguments. Additionally, the use of `return_exceptions=True` in the `asyncio.gather()` enables us to collect the exceptions that might happen in the process – in this case the exception is going to be raised with the `fibo("A", -1)` call because we pass a negative value to the `fibo() function`. Finally, despite the exception, the other functions continue to work without any issues.

You can see the output of **fibo.py** in the screenshot (*facing page, top*). The result lines contain the `fibonacci(x)` text only whereas the lines containing `Compute` are showing the computations' progress. All calculations need to compute `fibonacci(2)` to find the desired number of the Fibonacci sequence, but only the last one needs to compute `fibonacci(6)`. Last, the output shows that the available CPU time is shared among the available coroutines.

```
→  code cat tasks.py
#!/usr/bin/env python3

import asyncio
import random

async def this_is_a_task():
    print("Task execution")
    temp = random.randint(0,100)
    return temp

async def main():
    for i in range(0, 5):
        task = asyncio.create_task(this_is_a_task())
        await task
        print(task.result())

asyncio.run(main())
→  code ./tasks.py
Task execution
95
```

Figure 3: This Figure shows the code of tasks.py as well as the output of tasks.py and futures.py that illustrate the use of tasks and futures in asyncio, respectively.

Forgetting to schedule a coroutine

If you forget to schedule a coroutine, Python will detect such situations and generate warning messages on screen. This is illustrated in the code of **never.py**. The problem lies in the next code exempt from **never.py**:

```
async def main():
  test()
```

So, we're calling `test()`, which is implemented elsewhere, without being scheduled. The output of **never.py** is going to be as follows:

```
$ ./never.py
./never.py:10: RuntimeWarning: coroutine 'test' was
never awaited
  test()
RuntimeWarning: Enable tracemalloc to get the object
allocation traceback
main() is about to return.
```

The output shows the root of the problem, which is that the `test()` coroutine was never awaited and therefore never scheduled for execution. We have two solutions, which are illustrated in the next code exempt:

```
asyncio.create_task(test())
await test()
```

You can use any one of the two ways you want in order to properly schedule `test()` for execution.

Echo server

Let's develop an Echo server in Python that uses asyncio. The screenshot (*facing page, bottom*) shows the Python code of the Echo server. The crucial code of **echo.py** is the following:

```
server = await asyncio.start_server(handle, '', PORT)
addr = server.sockets[0].getsockname()
```

Most of the magic happens by the `asyncio.start_server()` function that starts a socket server using the specified parameters. Because its second parameter is `''`, this means that the server is going to listen to all available network interfaces, which can be a security risk when you're using that on a machine that's exposed on the Internet. Replace `''` with `'127.0.0.1'` to accept connections from the local machine only.

The first parameter is the function that will be executed each time a new client is connected to the server, to handle the connection. That handler function has two parameters, which are the reader and the writer, respectively and are used for interacting with clients.

The handle(reader, writer) coroutines that are created to interact with clients until the client sends the STOP keyword, which is defined in the code, and closes its connection. However, the remaining connections continue to work. Additionally, even if a client has a slow interaction with the server, the remaining clients work without any delays. When running and interacting with clients, **echo.py** generates the following output:

```
$ ./echo.py
Serving on ('::', 1234, 0, 0)
```

```
[TypeError('exceptions must derive from BaseException'), None, None, None, None
→ code ./fibo.py
Task B: Compute fibonacci(2)...
Task C: Compute fibonacci(2)...
Task D: Compute fibonacci(2)...
Task E: Compute fibonacci(2)...
Task B: Compute fibonacci(3)...
Task C: Compute fibonacci(3)...
Task D: Compute fibonacci(3)...
Task E: Compute fibonacci(3)...
Task B: fibonacci(3) = 2
Task C: Compute fibonacci(4)...
Task D: Compute fibonacci(4)...
Task E: Compute fibonacci(4)...
Task C: fibonacci(4) = 3
Task D: Compute fibonacci(5)...
Task E: Compute fibonacci(5)...
Task D: fibonacci(5) = 5
Task E: Compute fibonacci(6)...
Task E: fibonacci(6) = 8
[TypeError('exceptions must derive from BaseException'), None, None, None, None
→ code
```

The asyncio.gather() function enables you to collect the results from all coroutines, including the exceptions that might be raised.

```
Machine ('192.168.1.7', 42750) just connected.
Machine ('127.0.0.1', 51288) just connected.
('127.0.0.1', 51288) closed the connection.
Machine ('127.0.0.1', 51291) just connected.
Machine ('127.0.0.1', 51292) just connected.
('127.0.0.1', 51292) closed the connection.
('127.0.0.1', 51291) closed the connection.
('192.168.1.7', 42750) closed the connection.
```

The length of **echo.py** as well as its simplicity and readability make asyncio ideal for creating TCP/IP utilities. Especially when you consider that TCP/IP servers have frequent delays due to network issues or a slow client interaction. Learn more about creating streaming servers with asyncio at **https://docs.python.org/3/library/asyncio-stream.html.**

The next section examines whether asyncio improves the execution time of a program or not.

Is it really faster?

Let's calculate the numbers of the Fibonacci sequence with and without the use of asyncio, to verify that asyncio speeds up things when we have delays. The code of **multiFutures.py** presents the asyncio version whereas **serialFibo.py** presents the sequential version. The logic of **multiFutures.py** can be found here:

```
async def main():
    tasks = []
    for i in range(15):
        tasks.append(asyncio.ensure_future(random_
delay(i)))
    await asyncio.gather(*tasks)
```

The `main()` function defines the desired number of coroutines using `asyncio.ensure_future()` and puts them into a list named tasks. After that, `asyncio.gather(*tasks)` executes that list of coroutines and waits for them to finish. See the code of **multiFutures.py** for more details.

The results of these two Python scripts reveal that the asynchronous version is always faster than the sequential version mainly because the sleep (delay) times of the sequential version are added whereas the delays of the asynchronous versions are shared. Second, the code of **serialFibo.py** is easier to read, especially if you aren't familiar with asyncio. Finally, the asynchronous version uses futures and `asyncio.gather()`, which is a common when using asyncio.

» GLOBAL INTERPRETER LOCK (GIL)

Because Global Interpreter Lock (GIL) is a part of Python and the way Python works, it's good to learn more about it. GIL is a mutex (mutual exclusion variable) that permits only a single thread to hold the control of the Python interpreter. Put simply, this means that at every single moment only a single thread can be executed in Python. In a world with multiple CPUs and cores, this is not good.

So you might ask why do we need GIL. The main reason is to prevent deadlocks, which is a serious issue in multi-threaded programming. The second reason is that back in the first days of Python, computers didn't have multiple cores, so using GIL wasn't an issue back then. The main reason that GIL hasn't been removed from the language is that this would create a lot of backward incompatibility issues, which in a language as popular as Python would be a huge issue.

If you want to use threads in Python, there are three modules that can be used: the low level _thread (**https://docs.python.org/3/library/_thread.html**), threading (**https://docs.python.org/3/library/threading.html**) and multiprocessing (**https://docs.python.org/3/library/multiprocessing.html**). Discussing more about them is beyond the scope of this tutorial.

Due to GIL and the way it operates and restricts Python code, using multiple threads doesn't make your code faster most of the time. So it's worth trying asynchronous programming using asyncio.

Although it's not presented here, the same technique can be used when querying a database server and waiting for an answer. Because a query answer might take a while, depending on the nature of the query and the load of the database server, using asyncio in such cases looks like a rational choice.

The best way to learn asyncio and its idiosyncrasies is to begin building programs with it. Begin with small and easy ones before continuing with more complex ones. Once you get used to the way asyncio works, apply these principles when solving real world problems. At the end of the day, it's up to the developer to decide whether the extra effort of using asyncio and its features is worth it or not and write the extra code for supporting asynchronous operations. LXF

```
11    while True:
12        data = await reader.read(100)
13        message = data.decode().strip()
14        writer.write(data)
15        await writer.drain()
16        if message == "STOP":
17            writer.write(b'Bye!\n')
18            message = f"{addr!r} closed the connection."
19            print(message)
20            break
21
22    writer.close()
23
24 async def main():
25    # Listen on all interfaces and port number PORT
26    server = await asyncio.start_server(handle, '', PORT)
27    addr = server.sockets[0].getsockname()
28
29    print(f'Serving on {addr}')
30    async with server:
31        await server.serve_forever()
```

This is the implementation of an Echo server using asyncio as found in echo.py. Most of the work is done by asyncio.start_server() as well as the function that handles socket connections, which in this case is called handle().

On the disc

Discover the highlights from this month's packed DVD!

DOWNLOAD YOUR DVD
Get code and DVD images at:
www.linuxformat.com/archives

START HERE

USING THE LXFDVD

Using Linux for the first time can be very confusing. It'll most likely be unlike anything that you've operated before, especially if you're used to Microsoft Windows or Apple macOS.

Generally our DVDs are designed to be run directly, which is to say that when you first power on your PC (or Mac, *see below*) it should 'boot' from the DVD – so before Windows or macOS even starts to load – with Linux running directly from the DVD. This trick is known as a Live Disc. It enables you to try out the various versions of Linux without having to install or change anything on your PC. Just remove the DVD, restart your PC and it'll be exactly as you left it.

While many systems will boot from a DVD when it finds one, many will not. See below for the standard process for enabling booting from a DVD on various desktops and laptop PCs.

The alternative option is to locate the ISO file on the DVD and write this to your own USB thumb drive and attempt to run that. We recommend using *Etcher* from **https://balena.io/etcher** that's available for Windows, macOS and Linux. Good luck!

BOOT THE DISC

Many PCs should boot automatically if they're turned on with a disc in the drive. If not, many offer an early Boot Menu accessed by tapping a key while powering up from cold: F9 (HP), F12 (Dell, Lenovo), F8 (Amibios) or F11 (Award BIOS). Alternatively, use the BIOS/UEFI to adjust the boot order to start with the optical drive. Again, this is accessed by tapping a key during power up, usually Del but sometimes F1 or F2.

Some new UEFI PCs require access via Windows: holding Shift select its Restart option. If you're still having problems using the DVD then visit **www.linuxformat.com/dvdsupport**

Mac owners: Hold the C key while powering on your system to boot from the disc.

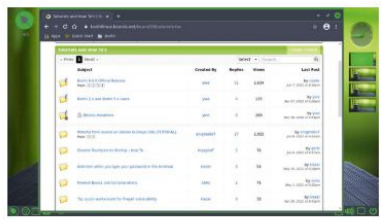
DESKTOP SATORI
MIN SPECS: 512MB RAM, 5GB SPACE

Bodhi 6.0 HWE

64-bit

We've always been fans of stories about humans sitting under bodhi trees listening to Nirvana or however it goes, and we're also big fans of Bodhi Linux. The lightweight distro sees some big changes with this release, one of which is the lack of a 32-bit edition. Bodhi 5.1 is still covered by the Ubuntu 18.04 support period, so you can still use that on aging hardware (even from the pre-PAE age) until 2023. We were going to put it on the disc too, but storage densities and ISO9660 overheads are harsh mistresses. We've gone for the hardware-enablement (HWE) release this time, since the standard release ships with a relatively old kernel. The HWE release uses a 5.8 series kernel, so covers some newer hardware from around the turn of the decade.

This release is based on Ubuntu 20.04.2, and has been in the works for some time. There have been some changes at camp Bodhi since the 5.0 release back in 2018. Long-time maintainer Jeff Hoogland stepped aside in June 2019 and installed a new lead developer: Robert 'ylee' Wiley. Since then ylee and the team have crafted a new website, wiki and version of the Moksha desktop. Moksha (apparently a term for that profound but rare sense of liberation contributors feel when they meet deadlines) is a fork of the Enlightenment (E17) desktop. We've always said it has a steampunk feel to it, but don't let that put you off.



There's a small but dedicated community behind Bodhi, and they'll be only to happy to help you (if you ask nicely).

We can't get enough of the visual bell effect featured here in Terminology. We're simple creatures on Linux Format.

Moksha is stylish, customisable and one of the lightest desktop environments known to Linux. Lately it's got a new theme, Arc-Green, a new splash screen and better language support. It also uses PNG icons, which load faster on older machines. The *PcManFM* file manager has been dropped in favour of Bodhi's own take on *Thunar* (the file manager from the Xfce desktop). Like Linux Mint, Bodhi by default disables Snap packages and the means by which to install them. But there are ways and means if you really want them.

Users of previous versions of Bodhi might have noticed that the *Forecast* weather applet stopped working some time ago. This was the fault of a change in the Yahoo API and would be bad news (now that it's the British summer and we need to know when the next shower will arrive), but this module has been refactored to use the **wttr.in** API instead. And it seems the next shower is, oh, now.

IMPORTANT NOTICE!

DEFECTIVE DISCS: For basic help on running the disc or in the unlikely event of your *Linux Format* coverdisc being in any way defective, please visit our support site at www.linuxformat.com/dvdsupport. Unfortunately, we're unable to offer advice on using the applications, your hardware or the operating system itself.

LIGHT BUT LUXURIOUS
MIN SPECS: N/A

Lubuntu 21.04

64-bit

Now that Ubuntu isn't doing 32-bit releases, its lighter editions are, to some extent, targeting newer hardware. The old (but feather-light) LXDE desktop that used to power Lubuntu, and was based on ancient and arcane GTK2 technology, was sunsetted a while ago. And now there's not much difference in terms of resource usage between Lubuntu, Xubuntu and Ubuntu MATE. So we flipped our distro triskellion and it came up showing the Lubuntu logo.

The new generation Lubuntu uses the LXQt desktop. One of the exciting things that happened when KDE 5.0 was released was the decoupling of the window manager, underlying libraries (including the Qt 5.0 toolkit) and the Plasma desktop itself. This enabled the LXQt project to make a modern-looking, Qt 5-powered desktop, operated by the lightweight *openbox* window manager. If you want to make it look more modern, you can switch it for KDE's Kwin and use any of its many beautiful themes.

Lubuntu uses the *Calamares* installer and KDE's Discover 'app store', which makes installing and augmenting the distro simple. If regular Ubuntu works, albeit slowly, on your system, then this should work a little better.

The first time you run Lubuntu, it will offer to configure power management. Good for laptops, not so good for VMs.

"PROUDLY ANTI-FASCIST"
MIN SPECS: 256MB RAM, 4GB DISK

AntiX 19.4

32-bit

The ripples from Ubuntu's decision three years back to wind down 32-bit support are in some ways still being felt. It's good then that those distros wanting to continue serving 32-bit architectures can at least rebase things on Debian 10 (which will support this machinery until 2024). This isn't a problem for antiX though, or its sister distribution MX Linux which was on last month's DVD, which have been based on Debian for ages.

If you don't like Systemd then good news, antiX uses its own custom init scripts instead. In fact, you'll note right off the bat how customisable the whole thing is, from the pleathora of options available at boot time to the number of different Conky configurations you can, with a couple of clicks, emblazon 'pon your desktop. The default desktop is a combination of *IceWM* and *Rox*, a file manager imbued with the strange ability to display desktop icons.

For those feeling the need for newer software, you can always enable the backports repository, or if you're feeling brave switch to the Testing or Unstable Debian branches. That's a one-way trip though: there's no going back. So why not use antiX's USB installer to have a portable install that you can mess with? And tell us how you got on – we're still stuck choosing Conky themes. LXF

Antix's control centre will help you prod whatever settings need prodded – no pointy stick required.

» NEW TO LINUX? START HERE...

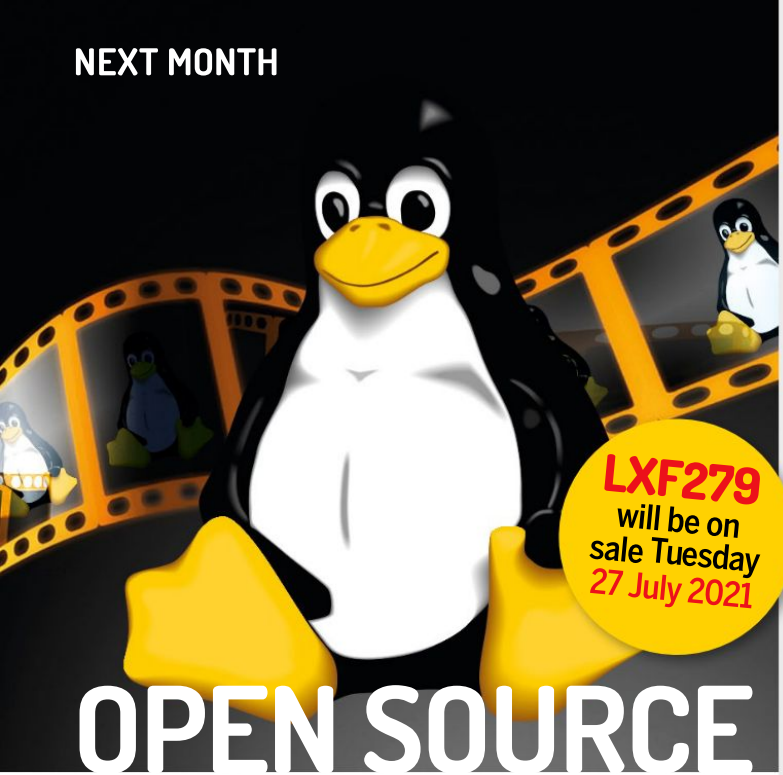
Never used a Linux before? Here are some handy resources:
● Read our quick-install guide **http://bit.ly/LXFinstall** ● Looking for an answer? **https://askubuntu.com** ● Want to delve more deeply? **https://linuxjourney.com**

» AND MORE!

THE LXF LIBRARY

- **Advanced Bash Scripting Guide**
  Go further with shell scripting.
- **Bash Guide for Beginners**
  Get to grips with the basics of Bash scripting.
- **Bourne Shell Scripting**
  First steps in shell scripting.
- **The Cathedral and the Bazaar**
  Eric S. Raymond's classic text explains the advantages of open development.
- **The Debian Book**
  Essential guide for sysadmins.
- **Dive Into Python**
  Everything you need to know.
- **Introduction to Linux**
  A handy guide full of pointers for new Linux users.
- **Linux Dictionary**
  The A-Z of everything to do with Linux.
- **Linux Kernel in a Nutshell**
  An introduction to the kernel written by master hacker Greg Kroah-Hartman.
- **The Linux System Administrator's Guide**
  Take control of your system.
- **Tools Summary**
  Overview of GNU tools.
- **GNU Emacs Manual**
  Six hundred pages of essential information!
- **Producing Open Source Software**
  Everything you need to know.
- **Programming from the Ground Up**
  Take your first steps.

NEXT MONTH



OPEN SOURCE STREAMING!

Become an internet star and stream your life to Twitch, YouTube or your own hosted solution.

Get a better VPN

Hide your tracks online with a better virtual private network. We explore the murky world to help you pick a better service.

VirtualBox networking

We kick off a two-part look at networking with *VirtualBox* by taking you through all the basic settings you need to know.

Office suites tested

We pitch *LibreOffice* against all comers to see if there's anyone in the FOSS world that can take on this productivity giant!

Easy image editing

Take a ride with *Nomacs*, the lightweight image viewer and editor that makes it easy to tweak your photos to perfection.

Contents of future issues subject to change – as we might never come back from the pub...

LINUX FORMAT
The #1 open source mag

Future Publishing Limited,
Quay House, The Ambury, Bath, BA1 1UA
Email linuxformat@futurenet.com

EDITORIAL
Editor Neil Mohr
CentOS FanBoy Jonni Bidwell
Art editor Efrain Hernandez-Mendoza
Operations editor Cliff Hope
Group editor in chief Graham Barlow
Senior art editor Jo Gulliver
Editorial contributors
Tim Armstrong, Mike Bedford, Neil Bothwick, Christian Cawley, Matthew Hanson, Jon Masters, Nick Peers, Les Pounder, Michael Reed, Calvin Robinson, Mayank Sharma, Shashank Sharma, Mihalis Tsoukalous, Alexander Tolstoy
Cover illustration magictorch.com

ADVERTISING
Commercial sales director Clare Dove
clare.dove@futurenet.com
Senior advertising manager Lara Jaggon
lara.jaggon@futurenet.com
Head of commercial – Technology Dave Randall
dave.randall@futurenet.com
Account director Andrew Tilbury
andrew.tilbury@futurenet.com

INTERNATIONAL LICENSING
Linux Format is available for licensing and syndication. To find our more contact us at *licensing@futurenet.com* or view our content at www.futurecontenthub.com.
Head of Print Licensing Rachel Shaw

NEW SUBSCRIPTIONS & BACK ISSUES
Web www.magazinesdirect.com
UK 0330 333 1113 **World** +44 (0) 330 333 1113

EXISTING SUBSCRIPTIONS
Web www.mymagazine.co.uk
UK 0330 333 4333 **World** +44 (0) 330 333 4333

CIRCULATION
Head of newstrade Tim Mathers

PRODUCTION AND DISTRIBUTION
Head of production UK & US Mark Constance
Production project manager Clare Scott
Senior ad production manager Jo Crosby
Digital editions controller Jason Hudson

THE MANAGEMENT
Chief audience and ecommerce officer Aaron Asadi
MD, tech specialist Keith Walker
Head of art & design Rodney Dive
Commercial finance director Dan Jotcham
Printed by Wyndeham Peterborough, Storey's Bar Road, Peterborough, Cambridgeshire, PE1 5YS
Distributed by Marketforce, 5 Churchill Place, Canary Wharf, London, E14 5HU www.marketforce.co.uk
Tel: 0203 787 9001

The Electronic Frontier Foundation is the leading nonprofit organization defending civil liberties in the digital world. Founded in 1990, EFF champions user privacy, free expression, and innovation through impact litigation, policy analysis, grassroots activism, and technology development. We work to ensure that rights and freedoms are enhanced and protected as our use of technology grows.

EFF.ORG

ELECTRONIC FRONTIER FOUNDATION

Protecting Rights and Promoting Freedom on the Electronic Frontier

PAPER POWER

60% of the energy used to produce paper and
paper packaging in Europe comes from renewable sources.

Discover the story of paper
www.lovepaper.org

LOVE PAPER

Source: Confederation of European Paper Industries (CEPI), 2018
CEPI represents 92% of European pulp and paper production