NEW
MARCH 2022

BDM's Complete
Manual Series

Linux

Coding & Programming

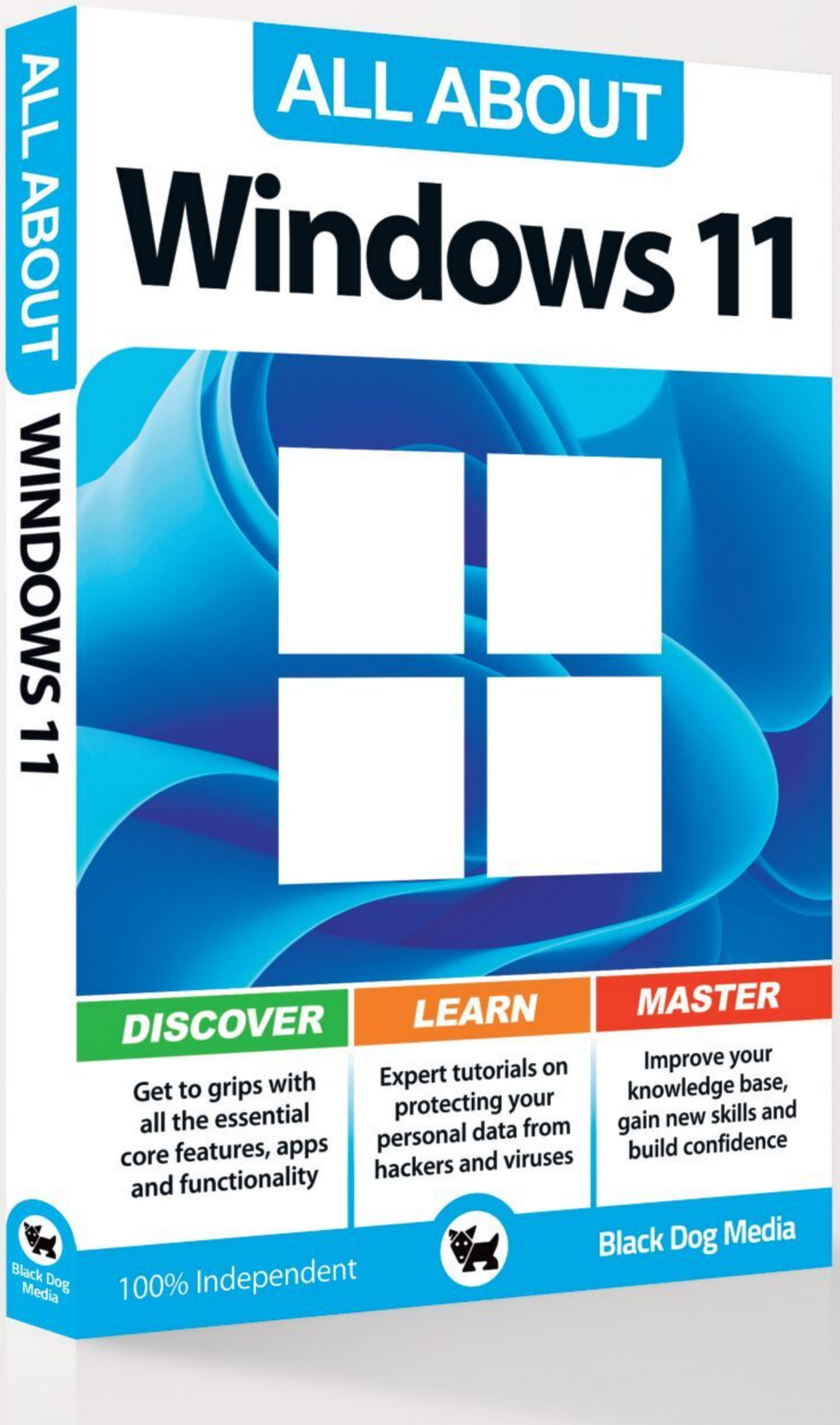
The essential step-by-step manual to improving your Linux coding skills

Discover Windows 11.
Learn New Features.
Master Your Desktop.

Black Dog Media

www.bdmpublications.com

NEW PRINT EDITION



Over 250 Pages of Expert Tutorials, Guides & Tips!

FEATURED INSIDE:

- Navigating the Start Menu
- Connecting to the Internet
- How to personalise Windows 11
- Web browsing with Edge
- Using OneDrive Cloud Storage
- Video chatting with Skype
- Email, Social Media & Messaging
- Improving Windows 11 security
- Wi-fi and Personal wi-fi hotspots
- Speeding up your computer
- Troubleshooting & User Advice and much more...

Click the link to BUY NOW from amazon

https://amzn.to/3EZFHKC

Available in all good bookshops

BDM's Complete Manual Series

Linux

Coding & Programming

Linux is everywhere. It powers the Internet as the main operating system behind the Web's servers, it powers spacecraft, it's the operating system for the fastest supercomputers in the world and it's used in smart TVs and mobile devices. Why? Because it's ultra stable, lightning fast and completely free of charge.

However, there's more to Linux than simply being a free to use operating system. Its unique configuration allows the user to customise and personalise the OS into any form they wish. A Linux user can change their OS look and feel from one day to the next, install thousands of freely available apps and programs and take back control of their computer.

Linux is about freedom. Freedom from the walled-garden approach of other restrictive operating systems, freedom to choose what you want on your computer, freedom to alter it and use it how you please. It's a worldwide community of likeminded users, all striving to get the best development from this incredible OS.

With this book, you too can become a part of the open community of Linux users. The tutorials within these pages will help you get to grips with Linux, show you how it works, what you can do with it and how you can code with it to take your Linux experience to even greater heights.

Discover Linux. Discover freedom.

Contents

BDM's Code Portal
60+ Python programs
21,500+ lines of code
Master Python with the help of our fantastic Code Portal, featuring code for games, tools and more.
Visit: https://bdmpublications.com/code-portal, and log in to get access!

6 Say Hello to Linux

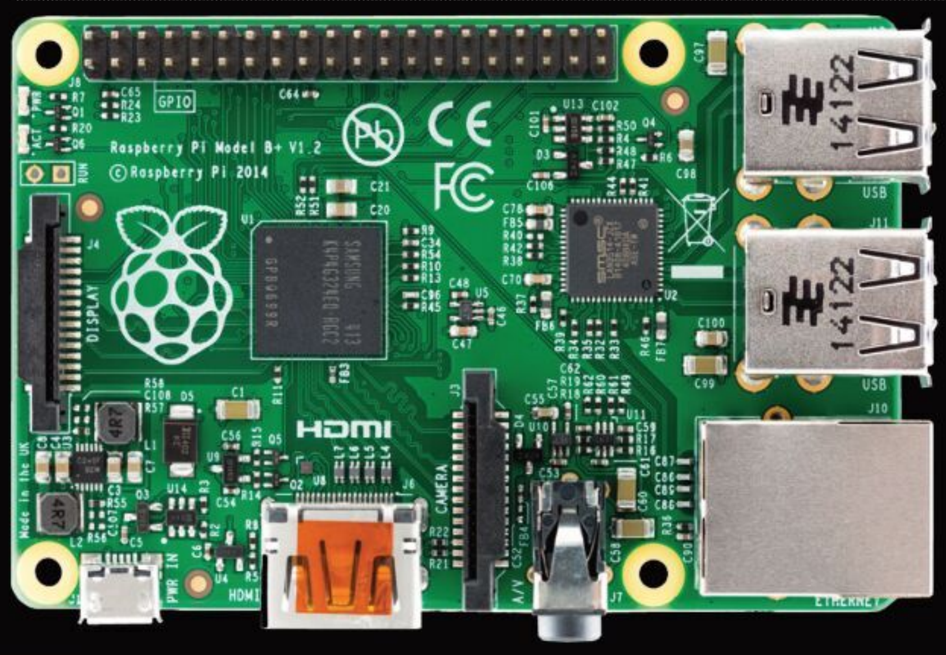
8 Why Linux?
10 The Best Linux Distributions
12 Equipment You Will Need
14 Desktop Environments
16 Which Distro?

18 Getting Started with Linux

20 Creating a Linux Installer on Windows
22 Installing Linux on a PC
24 Installing a Virtual Environment
26 Installing Linux in a Virtual Environment

28 Getting to Know Linux

30 Introduction to the Cinnamon Menu
32 Navigating the Cinnamon Desktop
34 10 Things to do After Installing Linux Mint
36 Did you Know…Apollo 11
38 Creating Users
40 Customising the Desktop
42 Becoming Anonymous Online



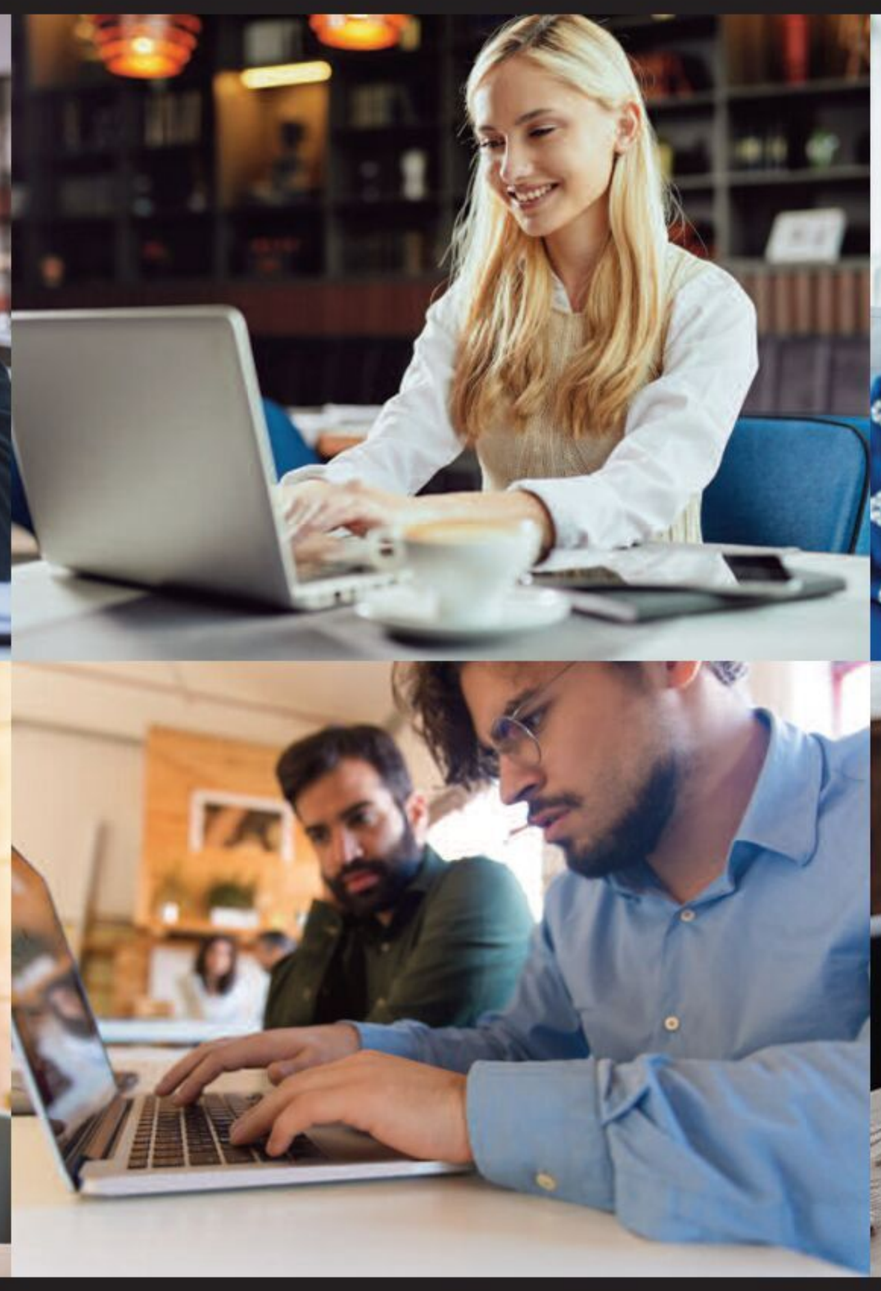
44 Using the Terminal

46 Basics of the Terminal
48 Update Mint via the Terminal
50 Install Apps via the Terminal – Part 1
52 Install Apps via the Terminal – Part 2
54 Did you Know…Linux Kernel 0.01
56 Creating a File Using the Terminal
58 Creating and Removing Directories
60 Fun Things to do in the Terminal
62 More Fun Things to do in the Terminal
64 Linux Tips and Tricks
66 Did you Know…Linux and the Big Bang
68 Creating Bash Scripts– Part 1
70 Creating Bash Scripts – Part 2
72 Creating Bash Scripts – Part 3
74 Creating Bash Scripts – Part 4
76 Creating Bash Scripts – Part 5
78 Pi x Linux = The Perfect Combination
80 Command Line Quick Reference
82 A-Z of Linux Commands
84 Did you Know…Good enough for NASA

86 Python on Linux

88 Why Python?
90 How to Set Up Python in Linux
92 Starting Python for the First Time
94 Your First Code
96 Saving and Executing Your Code
98 Executing Code from the Terminal
100 Did you Know…Space Invaders

Contents



102 Numbers and Expressions
104 Using Comments
106 Working with Variables
108 User Input
110 Creating Functions
112 Conditions and Loops
114 Python Modules
116 Did you Know…Debugging

118 C++ on Linux

120 Why C++?
122 Your First C++ Program
124 Structure of a C++ Program
126 Compile and Execute
128 Did you Know…Virus!
130 Using Comments
132 Variables
134 Data Types
136 Strings
138 C++ Maths
140 User Interaction
142 Did you Know…The Hobbit
144 Common Coding Mistakes

Black Dog i-Tech Series (Linux)
13th Edition | ISSN: 2044-4060

Published by: Black Dog Media Limited
Visit us at: **www.bdmpublications.com**
Managing Editor: James Gale
Production Director: Mark Ayshford
Editor: David Hayward
Production Manager: Karl Linstead
Design: Robin Drew, Lena Whitaker
Editorial: David Hayward
Digital distribution by: Readly, Pocketmags & Zinio

2022 © Copyright Black Dog Media Limited. All rights reserved. Notice: Before purchasing this publication please read and ensure that you fully understand the following guidelines, if you are in any doubt please don't buy. No part of this publication may be reproduced in any form, stored in a retrieval system or integrated into any other publication, database or commercial programs without the express written permission of the publisher. Under no circumstances should this publication and its contents be resold, lent, loaned out or used in any form by way of trade without the publisher's written permission. While we pride ourselves on the quality of the information we provide, Black Dog Media Limited reserves the right not to be held responsible for any mistakes or inaccuracies found within the text of this publication. Due to the nature of the software industry, the publisher cannot guarantee that all software and/or tutorials, tips, guides will work on every version of the required hardware. It remains the purchaser's sole responsibility to determine the suitability of this book and its content for whatever purpose. Any images reproduced on the front and back cover are solely for design purposes and are not representative of content. We advise all potential buyers to check listing prior to purchase for confirmation of actual content. All editorial opinion herein is that of the writer as an individual and is not representative of the publisher or any of its affiliates. Therefore the publisher holds no responsibility in regard to editorial opinion and content. Black Dog Media Limited reserves the right not to be held responsible for any mistakes or inaccuracies found within the text of this publication. The publisher, editor and their respective employees or affiliates will not accept responsibility for loss, damage, injury occasioned to any persons acting or refraining from action as a result of the content with this publication whether or not any such action is due to any error, negligent omission or act on the part of the publisher, editor and their respective employees or affiliates. Our articles are intended as a guide only. We are not advising you to change your device, and would actually advise against it if you have even the slightest doubts. There are potential risks to the hardware/software involved, and you must be aware of these before you decide to alter anything on your device. Read all of the information here carefully and then make up your own mind whether you want to follow our guides. We take no responsibility for damage to your computer or any other device used in the process. If you are unsure, please do not buy this publication.

This Black Dog Media Limited publication is fully independent and as such does not necessarily reflect the views or opinions of the manufacturers or hardware and software, applications or products contained within. This publication is not endorsed or associated in any way with The Linux Foundation, The Raspberry Pi Foundation, ARM Holding, Canonical Ltd, Python, Debian Project, Linux Mint, Microsoft, Lenovo, Dell, Hewlett-Packard, Apple and Samsung or any associate or affiliate company. All copyrights, trademarks and registered trademarks for the respective companies are acknowledged. Relevant graphic imagery reproduced with courtesy of Lenovo, Hewlett- Packard, Dell, Microsoft, Samsung, Linux Mint, NASA, and Apple.
All copyrights, trademarks and registered trademarks for the respective manufacturers, software and hardware companies are acknowledged.
All editorial content and design are copyright © Papercut Limited and reproduced under license to Black Dog Media Limited.
Additional images contained within this publication are reproduced under license from shutterstock.com.
All information was correct at time of print. Some content may have been previously published in other volumes or titles. We advise potential buyers to check the suitability of contents prior to purchase.

Black Dog Media Limited
Registered in England & Wales No: 05311511

ADVERTISING: Please contact: james@bdmpublications.com
INTERNATIONAL LICENSING: Black Dog Media Limited has many great publications and all are available for licensing worldwide.
For more information go to: www.bdmpublications.com
or email James Gale: james@bdmpublications.com

Say Hello to Linux

"How did you know so much about computers?"

"I didn't, it was the first one."

– Admiral Grace Hopper (pioneer programmer) when interviewed by David Letterman

Why Linux? What is it? Where do I get it? Why are there so many different versions of it? Most beginners ask these, and many more, questions when starting out. It's true that Linux is an incredibly versatile and powerful operating system but where do you start? Thankfully, you can find the answers in this section.

There is so much you can do with Linux but you need to know where to start; we're here to help you out. In this section you can learn what Linux is, what a distro is and what a desktop environment is. You can also begin to explore how Linux works and how it can work for you.

8	Why Linux?
10	The Best Linux Distributions
12	Equipment You Will Need
14	Desktop Environments
16	Which Distro?

Why Linux?

For many of its users, Linux means freedom. Freedom from the walled garden approach of other operating systems, freedom to change and use the OS as you please and freedom from any form of licensing or payment. There's a lot more to Linux than you may think though.

FREE AND OPEN

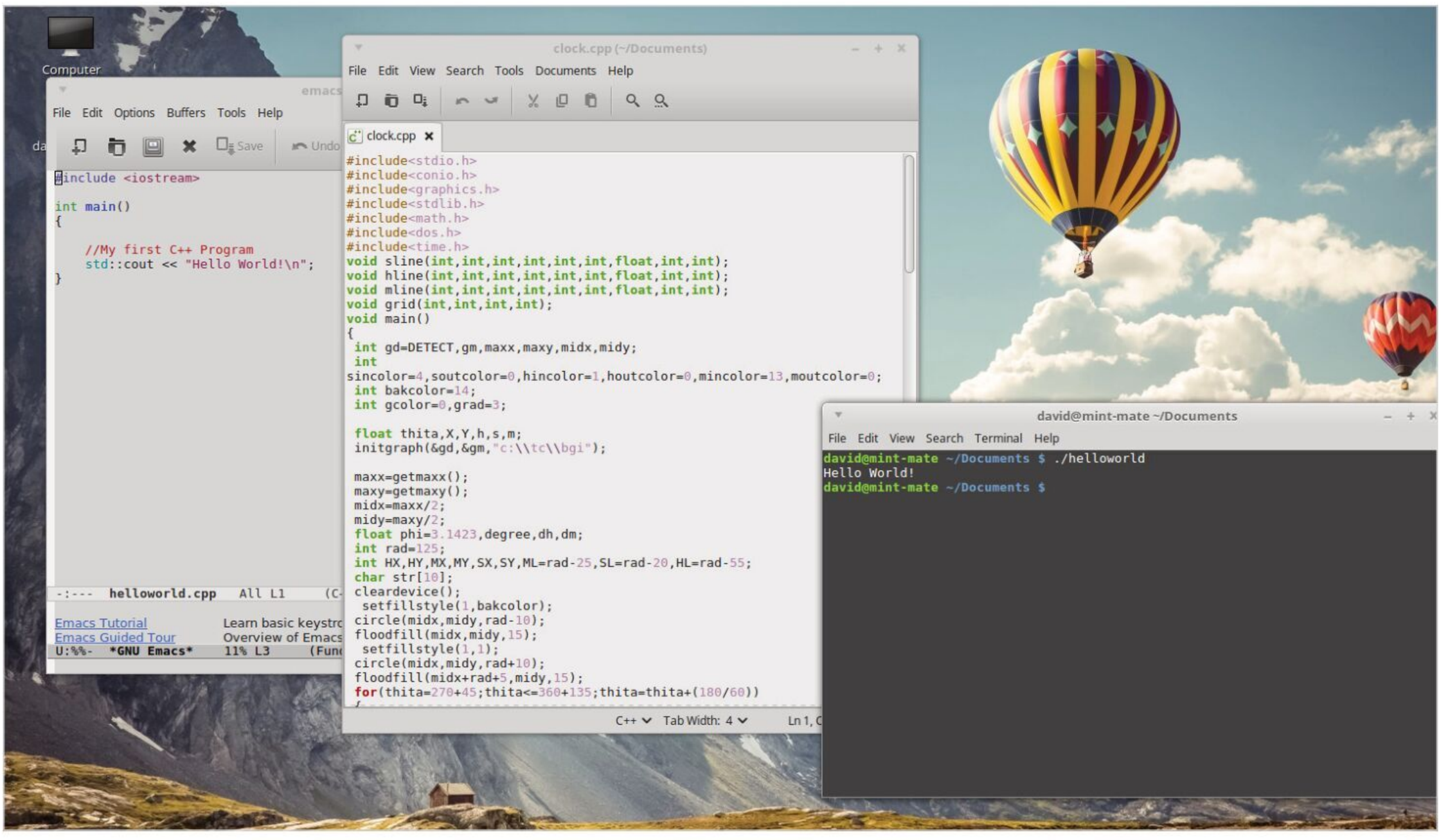
Linux is a fantastic fit for those who want something different. The efficiency of the system, the availability of applications and stability are just a few good reasons.

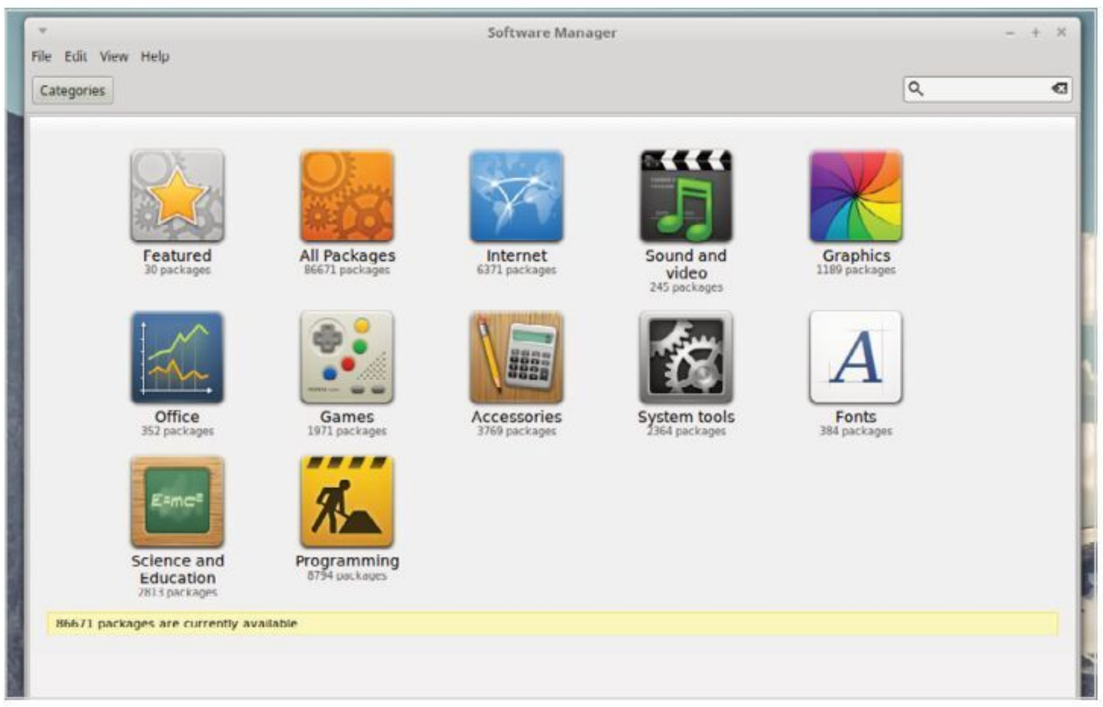
The first thing you need to know is that there is no such operating system called Linux. Linux is in fact the operating system kernel, the core component of an OS. When talking about Linux what we, and others, are referring to are one of the many distributions, or distros, that use the Linux kernel. No doubt you've heard of at least one of the current popular distros: Ubuntu, Linux Mint, Fedora, openSUSE, Debian, Raspbian, the list goes on. Each one of these distros offer something a little different for the user. While each has the Linux kernel at its core, they provide the user with a different looking desktop environment, different preloaded applications, different ways in which to update the system and get more apps installed and a slightly different look and feel throughout the entire system. However, at the centre lies Linux, which is why we say Linux.

Linux works considerably differently to Windows or macOS. It's free for a start: free to download, free to install on as many computers as you like, free to use for an unlimited amount of time and free to upgrade and extend with, equally, free programs and applications. This free to use element is one of the biggest draws for the developer. While a Windows license can cost up to £100, and a Mac considerably more, a user, be they a developer, gamer or someone who wants to put an older computer to use, can quickly download a distro and get to work in a matter of minutes.

Alongside the free to use aspect comes a level of freedom to customise and mould the system to your own uses. Each of the available distros available on the Internet have a certain 'spin',

Linux is a great operating system on which to start coding.





There are thousands of free packages available for programmers under Linux.

Each distro offers something unique to the user but all have Linux at the core.

in that some offer increased security, a fancy looking desktop, a gaming specific spin, or something directed toward students. This extensibility makes Linux a more desirable platform to use, as you can quickly mould the system into a development base, including many different kinds of IDEs for the likes of Python, web development, C++, Java and so on; or create a base for online anonymity, perhaps as a Minecraft server, media centre and much more.

Another remarkable advantage for those looking to learn how to code, is that Linux comes with most of the popular coding environments built in. Both Python and C++ are preinstalled in a high percentage of Linux distros available, which means you can start to program almost as soon as you install the system and boot it up for the first time.

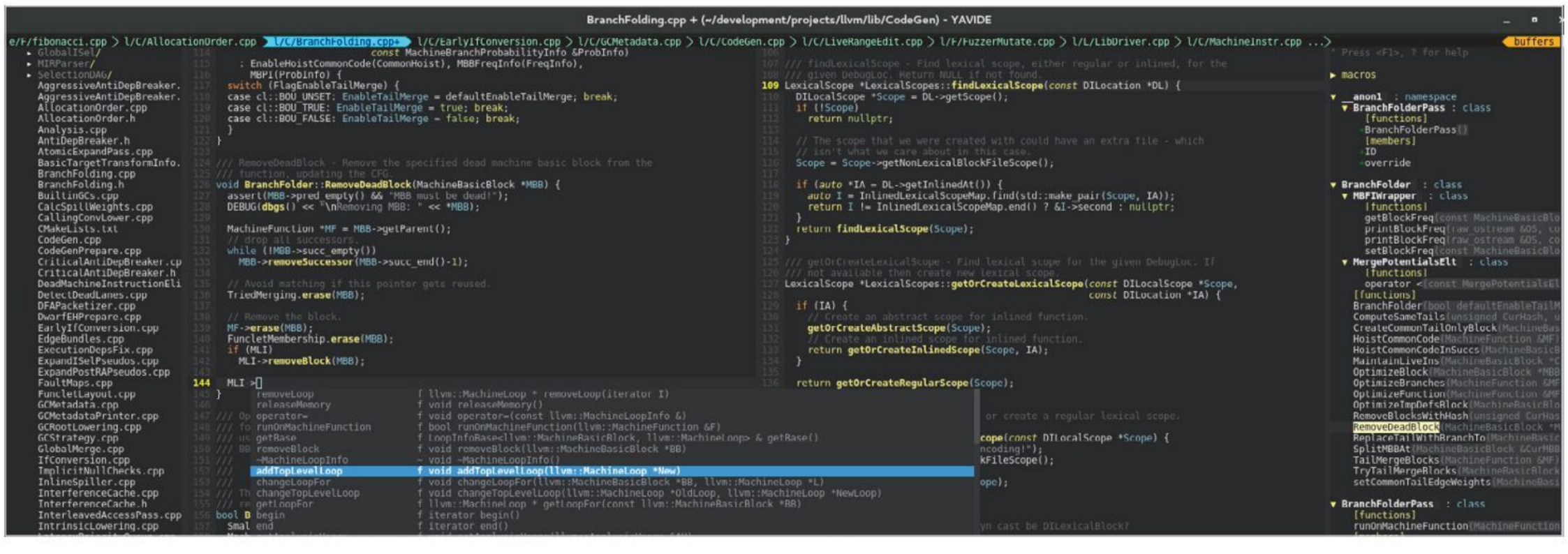
Generally speaking, Linux doesn't take up as many system resources as Windows or macOS; by system resources we mean memory, hard drive space and CPU load. The Linux code has been streamlined and is free from third-party 'bloatware' which hogs those systems resources. A more efficient system of course means more available resources for the coding and testing environment and the programs you eventually create. Less use of resources also means you can use Linux on older hardware that would normally struggle or even refuse to run the latest versions of Windows or macOS; so rather than throwing away an old computer, it can be reused with a Linux distro.

It's not all about C++, Python or any of the other more popular programming languages though. Using the command line of Linux, also called the Terminal, you're able to create Shell scripts, which are programs that are designed to run from the command line and made up of scripting languages. They are used mainly to automate tasks or offer the user some form of input and output for a certain operation.

Finally, although there are many more advantages we can list, there are thousands and thousands of free programs and apps available that cover nearly every aspect of computing. Known as packages, there are (at the time of writing) over 8,700 specific programming applications just on Linux Mint alone and an incredible 62,000+ overall packages catering for everything from Amateur Radio to WWW tools.

Linux then, is a great resource and environment for programming in. It's perfectly suited for developers and is continually improving and evolving. If you're serious about getting into coding, or you just want to try something new, give Linux a try and see how it works for you.

A Linux programming environment can be as simple or as complex as you need it to be.



The Best Linux Distributions

There are lots of versions of Linux available, known as Distributions. Each has a different ethos and approach. Here are five great distributions to try and where you can get them.

GOING LINUX

The installation process for most distributions is similar. You download a disk image from the website and burn it to an optical disk or create a USB Flash Drive installer. Just be careful to get the right distribution for your hardware and read the instructions carefully.

LINUX MINT

By far the most popular Linux distro (distribution) is Linux Mint. Mint began life back in 2006, as an alternative to the then most popular distro, Ubuntu. Although based on Ubuntu's Long Term Support build, Linux Mint took a different direction and offered the user a better overall experience.

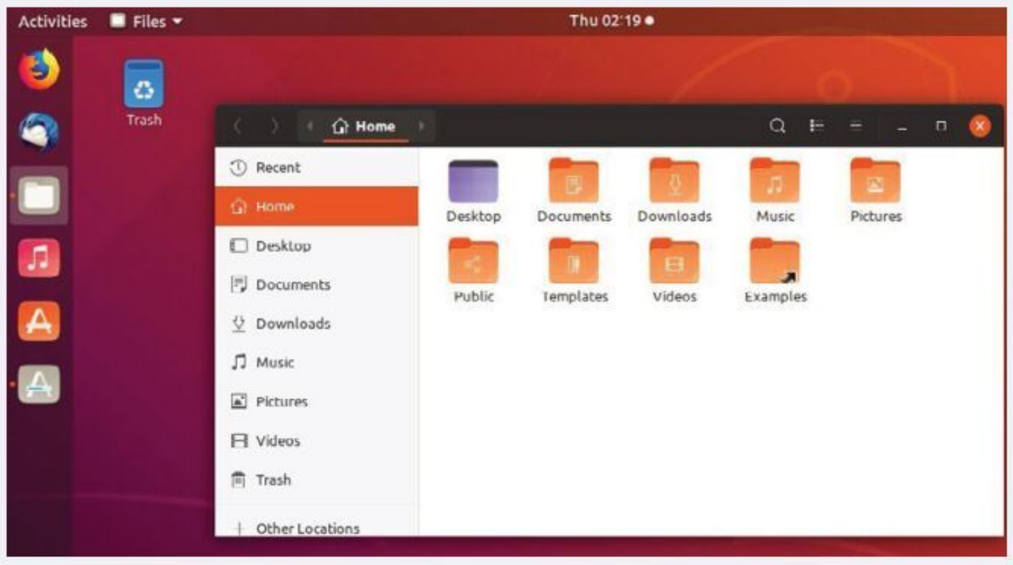
Linux Mint has three main desktop versions available with each new version of the core OS it releases. This may sound confusing at first but it's quite simple. Currently, Linux Mint uses the Cinnamon Desktop Environment as its flagship model; there's MATE and Xfce models available too.

Cinnamon is a graphically rich desktop environment, MATE uses less fancy graphics, and is more stable on a wider variety of desktop systems, and Xfce is an extremely streamlined desktop environment that's built for speed and ultimate stability.

Throughout this title we'll be using the Cinnamon version; however, you can try out any of the other desktop environments as you wish. In fact, it's recommended that you do spend some time trying different environments, and even different distros, to see which suits you and your computer best.

www.linuxmint.com

UBUNTU



The second most popular distro available is Ubuntu, which is an ancient African word meaning 'humanity to others'. Ubuntu's popularity has fluctuated over its fourteen year life. At one time, it was easily the most used Linux-based operating system in the world but some wrong choices along the way with regards to its presentation, and some unfavourable, controversial elements involving privacy, sadly saw it topple from the number one spot.

That said, Ubuntu has since made amends and is slowing crawling its way back up the Linux leader board. The latest versions of the OS use the GNOME 3 desktop environment, an impressive environment, although it can be a little confusing for former Windows users and is a little heavy on system resources, especially if you're planning on installing it on an older computer.

Ubuntu, for all its faults, is a good Linux distro to start experimenting with. It's a clean interface, easy to use and install and offers the user a complete Linux experience.

www.ubuntu.com

ARCH

Arch is one of longest running Linux distributions and forms the basis of many other versions of Linux. Why install Mint or Ubuntu when you can install Arch? Many users do exactly that but it's not ideal for beginners. Ubuntu and Mint both offer an easier installation path and come with software packages to help you get started.

Arch on the other hand, is a more 'bare bones' affair. Arch is committed to free software and its repositories contain over 50,000 apps to install, including multiple different Desktop environments, and use as you would with any other distro.

Arch is a distro for when you're more experienced with Linux. You start with nothing but the command line and from there you have to manually partition your hard drive, set where the installation files go, create a user, set the OS locale and finally install a desktop environment along with the apps you want.

The advantage though, for all this hard work, is a distro that you have created. This means your Arch distro won't come with all the unnecessary files and apps that others have preinstalled; it's custom made for you, by you.

www.archlinux.org

RASPBERRY PI DESKTOP

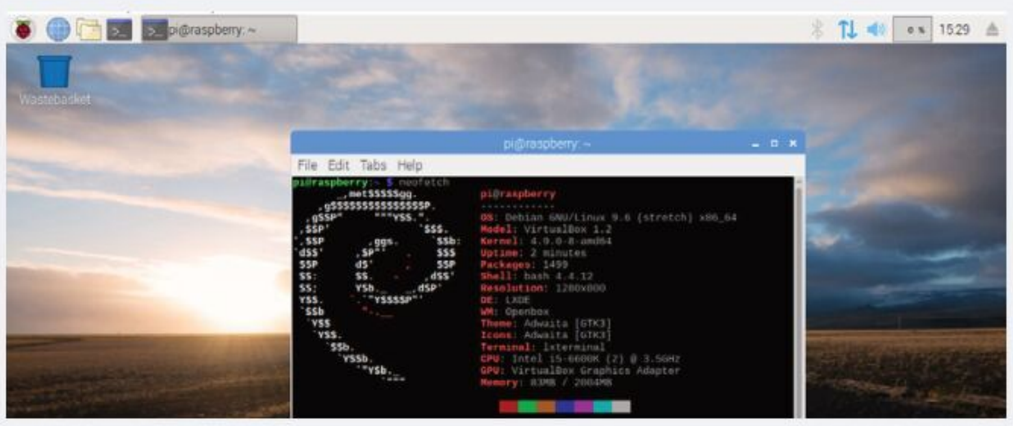
No doubt you've heard of the Raspberry Pi. It's hard not to have, as this remarkable, tiny computer has taken the technology world by storm for the last six years since it was introduced.

There are several aspects to the Raspberry Pi that make it such a sought after piece of the computing world. For one it's cheap, costing around £25 for what is essentially a fully working computer. It's small, measuring not much bigger than a credit card. You can build electronics with it, using a fully programmable interface; and it comes with Raspbian, its own custom-made, Debian-based operating system that includes an office suite alongside many different programming languages and educational resources.

Raspbian is exclusive to the Pi hardware, since the Raspberry Pi uses an ARM processor to power it. However, the Raspberry Pi Foundation has since released a PC version of Raspbian: Raspberry Pi Desktop.

Just like the Pi version, Raspberry Pi Desktop comes with the all the coding, educational and other apps you will ever need. It's quick, stable and works superbly. If you're interested in stretching your Linux experience, then this is certainly one of the top distros to consider.

www.raspberrypi.org/downloads/raspberry-pi-desktop



OPENSUSE

Most Linux distributions fall into two camps. There are ones with the latest features and technology like Ubuntu and Mint and those with few new features but rock solid reliability, like Debian.

Meanwhile, openSUSE attempts to cover both bases. OpenSUSE Leap is the rock solid system. It's developed openly by a community along with SUSE employees, who develop an enterprise-level operating system, SUSE; this powers the London Stock Exchange amongst other things. It is designed for mission critical environments where 'there is no scope for instability'. If you find all that too sensible, openSUSE Tumbleweed is a rolling release with all the latest features, and the occasional crash.

openSUSE is a highly respected Linux distribution and many of its core contributors work on the Linux Kernel, LibreOffice, Gnome and other key Linux areas. In short, openSUSE is where you'll find the pros hanging out.

www.openSUSE.org

Equipment You Will Need

The system requirements for successfully installing Linux Mint on to a PC are surprisingly low, so even a computer that's several years old will happily run this distro. However, it's worth checking you have everything in place before proceeding.

MINTY INGREDIENTS

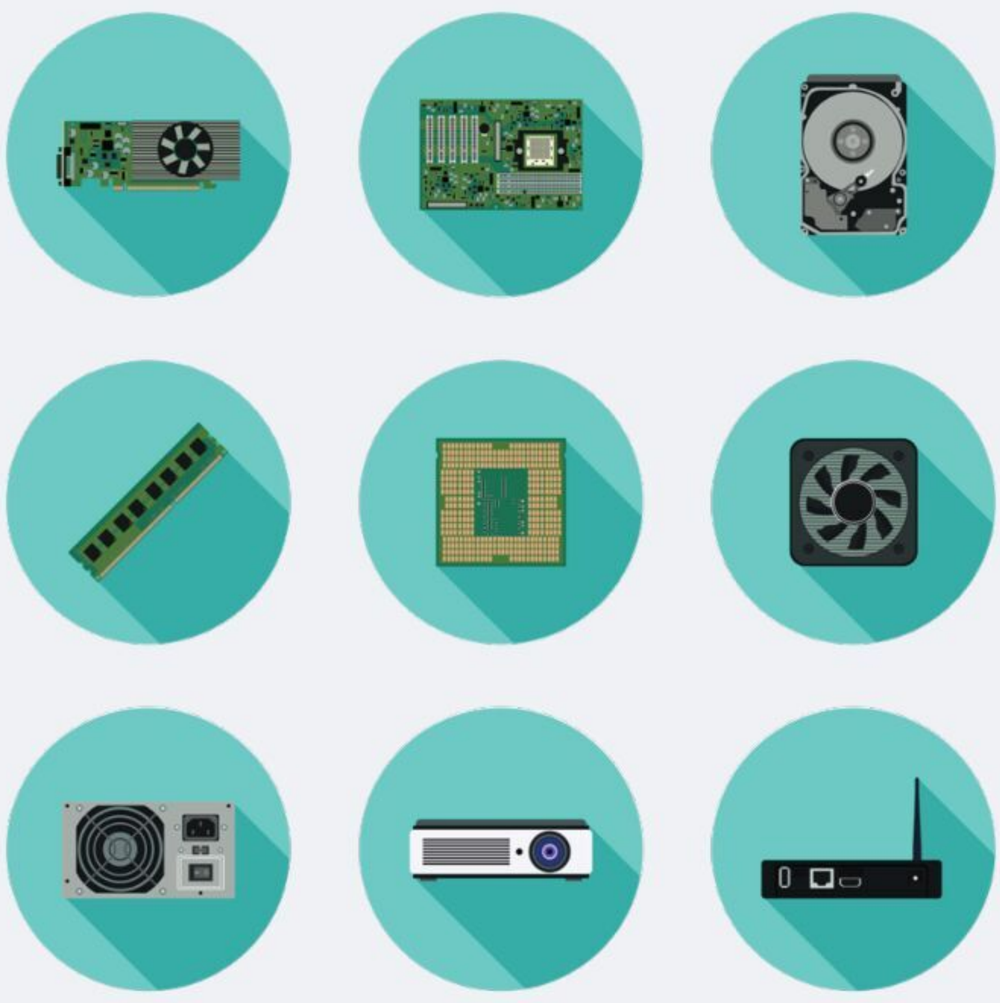
Before we start working our way through this book, here's what you need to install and run Linux Mint. You have several choices available, so take your time and see which works best for you.

SYSTEM REQUIREMENTS

The minimum system requirements for Linux Mint are as follows:

CPU – 700MHz
RAM/Memory – 512MB
Hard Drive space – 9GB (20GB recommended)
Display – 1024 x 768 resolution

Obviously the better the system you have, the better the experience will be and quicker too.



USB INSTALLATION

You can install Linux Mint onto your computer via USB or DVD. We look into each a little later on but if you're already familiar with the process, or thinking of USB and just gathering the hardware you need, then you're going to need a minimum 4GB USB flash drive to store the Linux Mint ISO.

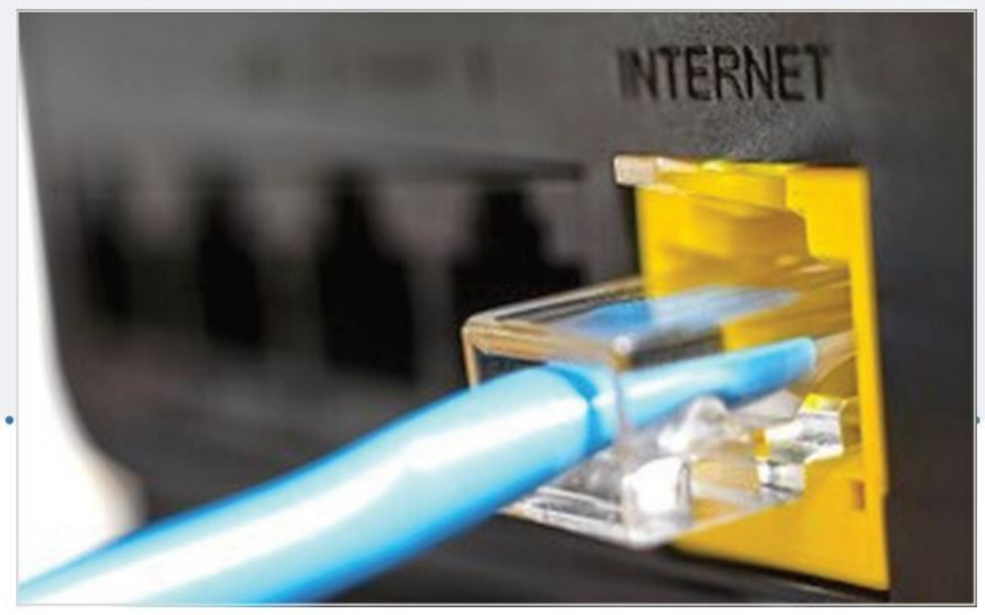
DVD INSTALLATION

DVD installation of Linux Mint simply requires a blank DVD-R disc. Of course, you also need an optical drive (a DVD Writer drive) before you're able to transfer or burn the ISO image to the disc.

VIRTUAL ENVIRONMENT

Installation to a virtual environment is a favourite method of testing and using Linux distros. Linux Mint works exceedingly well when used in a virtual environment but more on that later. There are many different virtual environment apps available; however, VirtualBox, from Oracle, is one of the easiest to into. You can find the latest version at **www.virtualbox.org**.

VirtualBox



INTERNET CONNECTION

It goes without saying really, that an Internet connection is vital for making sure that Linux Mint is up to date with the latest updates and patches, as well as the installation of further software. Although you don't need an Internet connection to use Linux Mint, you'll miss out on a world of free software available for the distro.

MAC HARDWARE

Although Linux Mint can be installed onto a Mac, there's a school of thought that recommends Mac owners use a virtual environment, such as Virtualbox or Parallels; and why not, macOS is already a splendid operating system. If you're wanting to breathe new life into an older Mac, make sure it's an Intel CPU model and not the Power PC models. Beware though, it's not as pain free as installing on to a PC.

Desktop Environments

Linux Mint comes in several different versions, or flavours: Cinnamon, MATE and Xfce; there are 32-bit and 64-bit versions of these too. What does it all mean though and which version should you choose for your installation?

WHICH MINT?

A Desktop Environment is the graphical interface which you use to interact with the core Linux system. Just as the graphical desktop for Windows 10 is also called Fluent Design.

Linux Mint offers the user a choice of versions of the distro: Cinnamon, MATE and Xfce. While that may sound a little confusing for the newcomer, essentially each of the versions available contains the same core Linux structure and kernel, the kernel is the core of the operating system, that handles all the instructions between the software and hardware.

Each version is simply a different desktop environment, the Graphical User Interface (GUI) that you use to interact with the operating system. Each of the desktop environments uses different apps to access or use the system, such as the file manager to browse the operating system's file structure or the way it launches other apps. Again though, the core available productivity, video and graphic suites are the same, and function in the same way.

Why bother then with a different desktop environment? Simply put, it's down to personal taste. Some users prefer MATE, as MATE is a fork of the classic GNOME 2 environment and is a little more menu-centric and performs well on older computers. Others prefer Cinnamon, which is a more modern environment that works better on recent hardware and features some cutting edge desktop code. Xfce, on the other hand, is a lightweight desktop environment that works well on older hardware due to its extremely low use of the available system resources.

In short, Cinnamon is the flagship desktop environment for Linux Mint. MATE is more compatible with a wider variety of hardware. Where Windows, for example, only offers one desktop environment to work in, Linux offers many. Linux Mint has therefore opted to bring the user a wealth of choice.

A DASH OF CINNAMON

Cinnamon has many benefits beyond its look and feel, although they are important factors. Here, we outline a couple of features to help you decide if it's the DE for you.

FAB FEATURE 1

It performs excellently on more modern systems but is a lot more stable than it used to be, and not quite the resource hog it once was. Therefore, don't be put off Cinnamon if you're using an older computer to install and test Linux Mint.

FAB FEATURE 2

Cinnamon features configurable Hot Corners, where each corner of the desktop can be clicked to perform a certain task, such as display the Workspaces, Show all Windows or Run a Command. The Hot Corners are an excellent way to switch between different views and help make the desktop a more efficient environment.

BEST MATE

MATE is a simple to use and intuitive DE that's fast and stable. In comparison to Cinnamon it looks a little antiquated but that's only on the surface. There's plenty to like with MATE.

FAB FEATURE 1

MATE is an excellent desktop environment for older computers. It works better with a larger number of hardware components that Cinnamon generally does but is also just as capable of delivering a great looking desktop as well as advanced customisations.

FAB FEATURE 2

Due to its highly configurable nature, MATE can be customised to a fine degree. There are plenty of options available to the user who demands a little more from their desktop environment, including Compiz Settings, where you're able to configure all manner of desktop effects, even a 3D desktop cube.

CHOLESTEROL FREE DESKTOP

Xfce started life as the Common Desktop Environment (CDE) in 1996. Since then it's changed name a few times and is now simply Xfce, with the nickname 'Cholesterol Free Desktop Environment'.

FAB FEATURE 1

Xfce really is a quick desktop and brings out the best in Linux as a fast operating system. In fact, it's taken us to this point from the top of the page to install it onto a new computer; a few hundred words written and we're using a brand new OS. Not bad.

FAB FEATURE 2

Just because it's quick and lightweight doesn't mean Linux Mint Xfce isn't a complete desktop operating system. Just like the other environments on offer, you get the latest Firefox, Libreoffice, VLC, Gimp, chat apps and even a bit torrent client.

Which Distro?

Up to now we've looked mainly at Linux Mint but there are other Linux distributions out there to try. In truth there are thousands of Linux distros available to download and install, so which one should you decide on to use?

DISTRO HOPPING

Distro hopping is a term used by the community for people who never stick to a single distribution. Instead, they hop from one to the other and back again, testing each, using them, then moving on to another or a newly released distro.

There's nothing wrong with distro hopping, as it's a good way to get to grips with what's out there and discover the elements of one distro over another that may or may not appeal to your tastes. The problem of course is which one do you use overall?

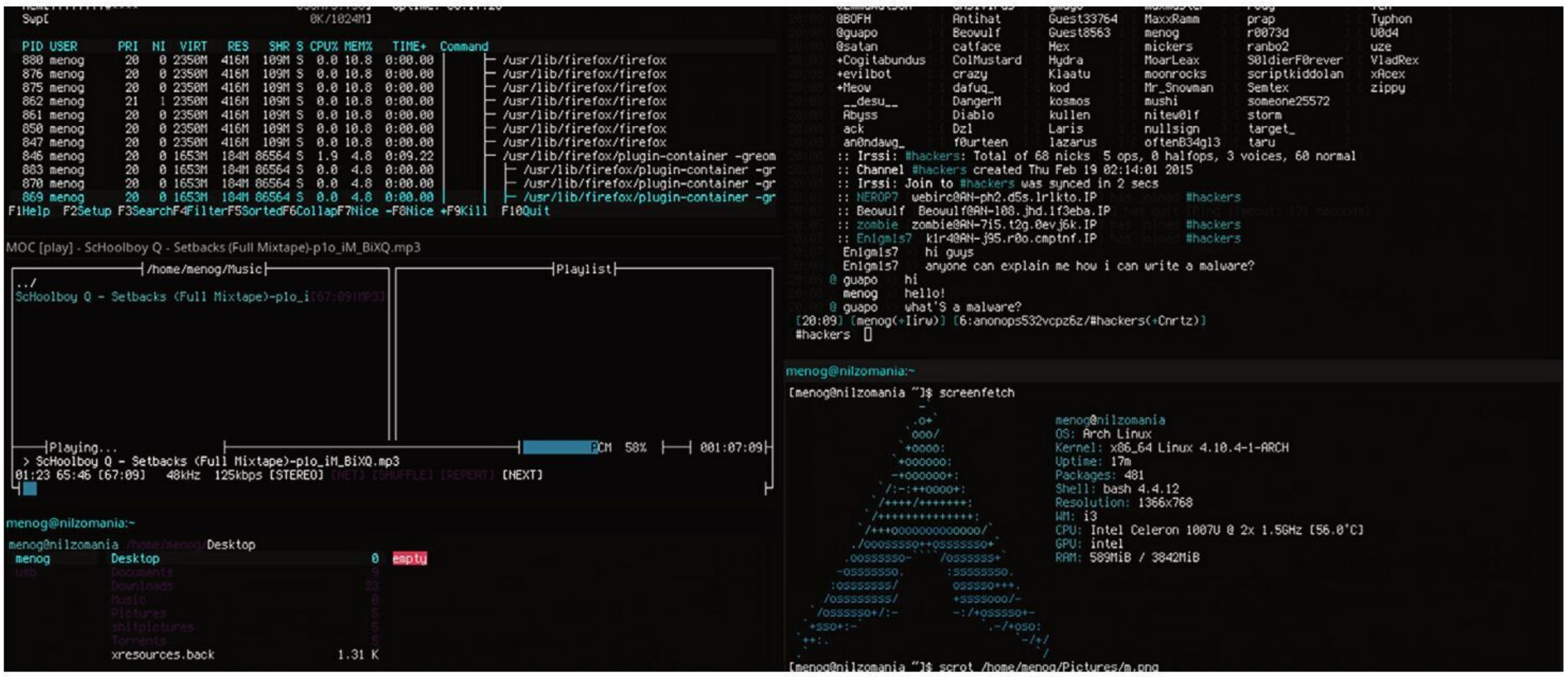
While distro hopping is a good thing, it's not exactly a stable way to enjoy Linux and get the most from it. We're not saying you should stick to one distro and never look elsewhere, as you would be missing a lot of great content out there, but instead we recommend you find a handful and slowly progress through them based on your increasing Linux skills.

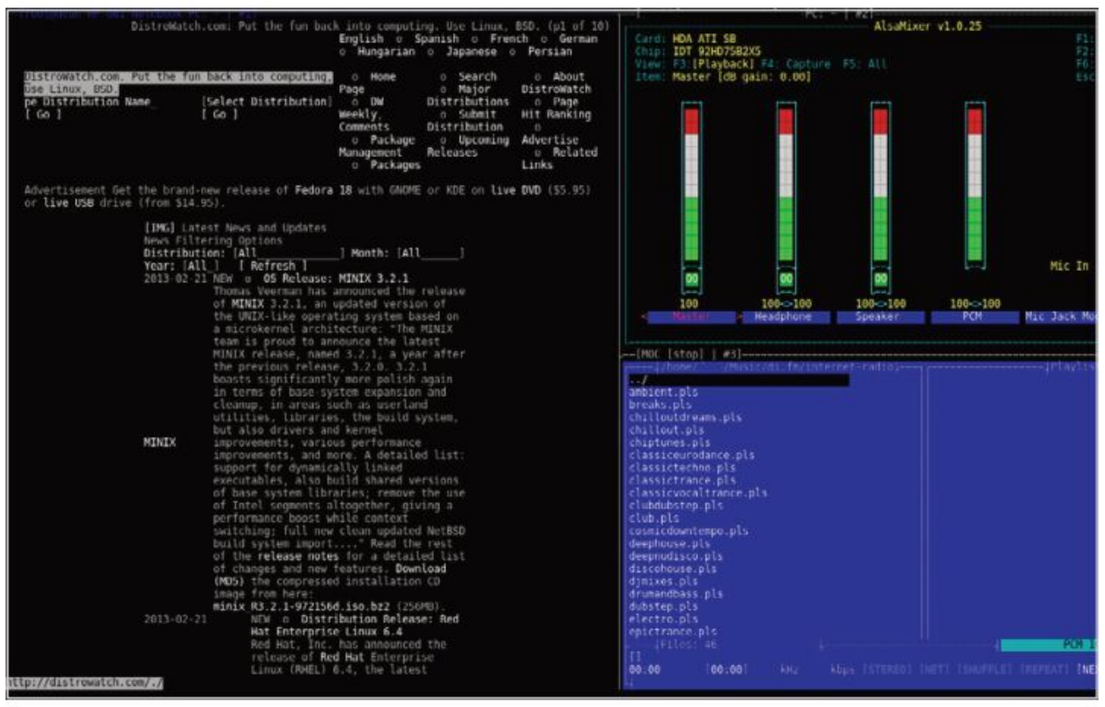
For example, Linux Mint is an ideal starting place. It's an easy to install and use distro, has all the software you would normally use on a day-to-day basis already installed out-of-the-box and gently eases you into the unique world of Linux and how it works and performs.

Ubuntu offers much the same experience but it does this in a slightly different way. There's generally less preinstalled with Ubuntu than with Linux Mint, so you would need to manually install it yourself. Another point worth considering is the sheer volume of content and help pages dedicated to Ubuntu users when using Linux. If you get stuck, you're never too far from a solution to the problem.

Moving on, as you begin to grow more confident with Linux, you may test out the likes of openSUSE, Fedora or Debian. These are all excellent distros and each offers the user a slightly different perspective on how the system runs. Some are more demanding, in terms of Linux skills, than others, but essentially they each have some valuable lessons to learn for the user.

You may find yourself moving to a particular distro because it offers something radically different from the norm. Tails Linux, for example, is a distro that's designed purely for online anonymity. It contains complex and military grade encryption tools as well as tools and browsers designed to help you browse the web without ever being detected, traced or monitored. Kali Linux is designed for security professionals and contains many different kinds of ethical hacking





tools preinstalled, that a user can run for penetration testing against their network. There was once even a Hannah Montana Linux distribution but the less we talk about that the better. The point being, there's a distro out there for you.

Needless to say, once you've mastered Linux to a relatively high degree, probably a power user ability, then you will want to expand your skills and begin to build your own Linux distro based on Arch, Debian or one of the many other distros available. Doing so involves a lot of command line knowledge, as well as knowledge on how the Linux system works and interacts with the hardware in the computer. You will need to partition your own hard drive, install a desktop environment and eventually install the apps and programs you want. Doing so takes time and again there are a lot of skills you're going to need to learn.

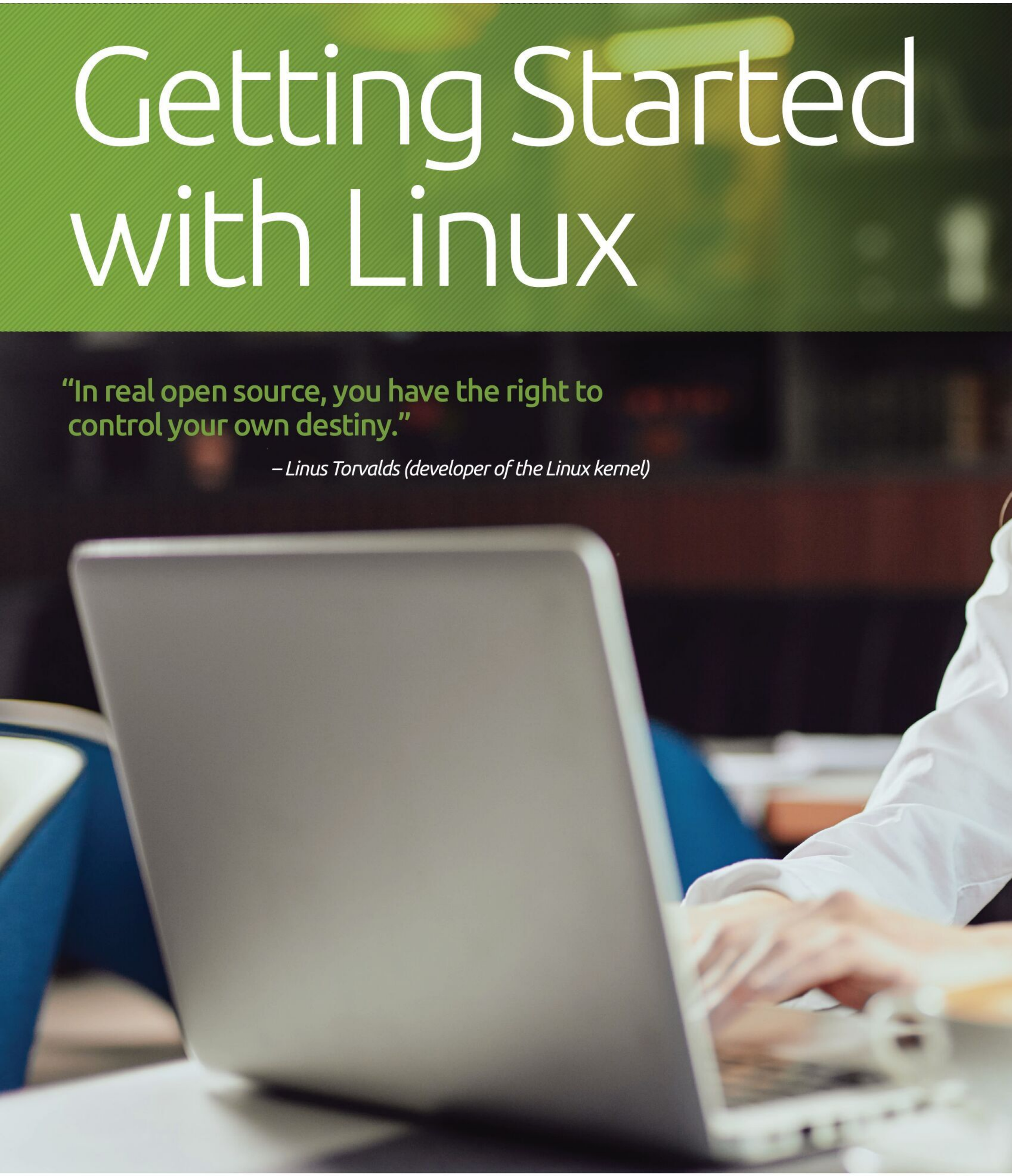
Eventually you can consider yourself a technical Linux user but never consider yourself an expert, after all we're always learning something new. You can build your own distro from scratch, help other Linux users out with problems, maybe even contribute to the improvement of a distro during its testing phase or build. Where next then?

Oddly enough, most higher-end technical users find themselves back at square one, using a distro like Linux Mint. The main reason is usually because it's an easy option, and it's a stable environment. Just because you know the system inside and out, doesn't mean you always want to be fixing potential issues. Most of us would prefer the easy life, especially where technology is concerned, so the logical choice would be to choose a distro that's simple, yet still powerful enough to do everything you want it to do, hence Linux Mint.

However, in the end, it's purely down to choice, your own personal choice. You may find that after going through the tutorials in this title you don't like Linux Mint or the Cinnamon desktop. Fine, you may prefer Ubuntu, Debian or openSUSE: that's the beauty of Linux. The freedom to change what you want, to distro hop from one to another without being penalised by cost or lack of access.

The answer to the question, which distro is: any which one you like! It can be as complex or easy as you need it to be, as long as it does what you want it to do, then it's perfect.

Getting Started with Linux



It's all fine and well talking about how good Linux is but how do you get it on your computer? Installing Linux is remarkably simple but there are several options available to you. This section looks into how you can download the Linux ISO, install it on a PC as your main operating system and even how to install a virtual environment.

With a virtual environment you can run Linux while still using your main operating system, be that Windows or macOS. Intrigued? Read on and find out more.

20 Creating a Linux Installer on Windows

22 Installing Linux on a PC

24 Installing a Virtual Environment

26 Installing Linux in a Virtual Environment

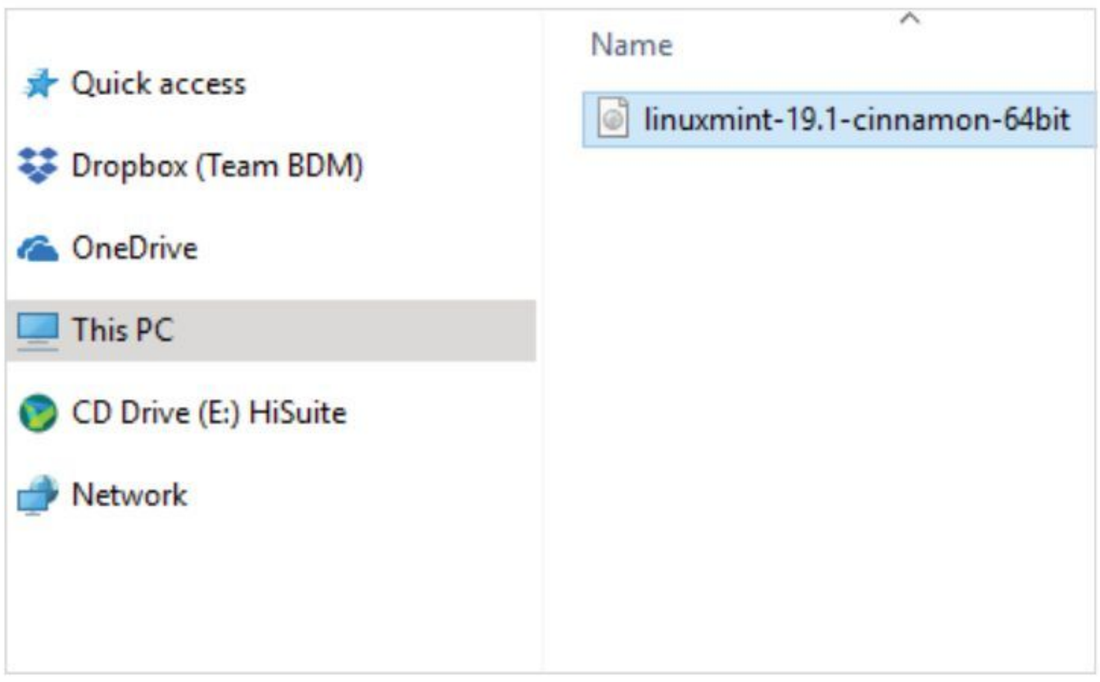
Creating a Linux Installer on Windows

You need to transfer the downloaded Linux ISO to either a DVD or a USB key before being able to install it onto a computer. This will be a live environment, which allows you to test the OS prior to installation, but first you need to create the bootable media.

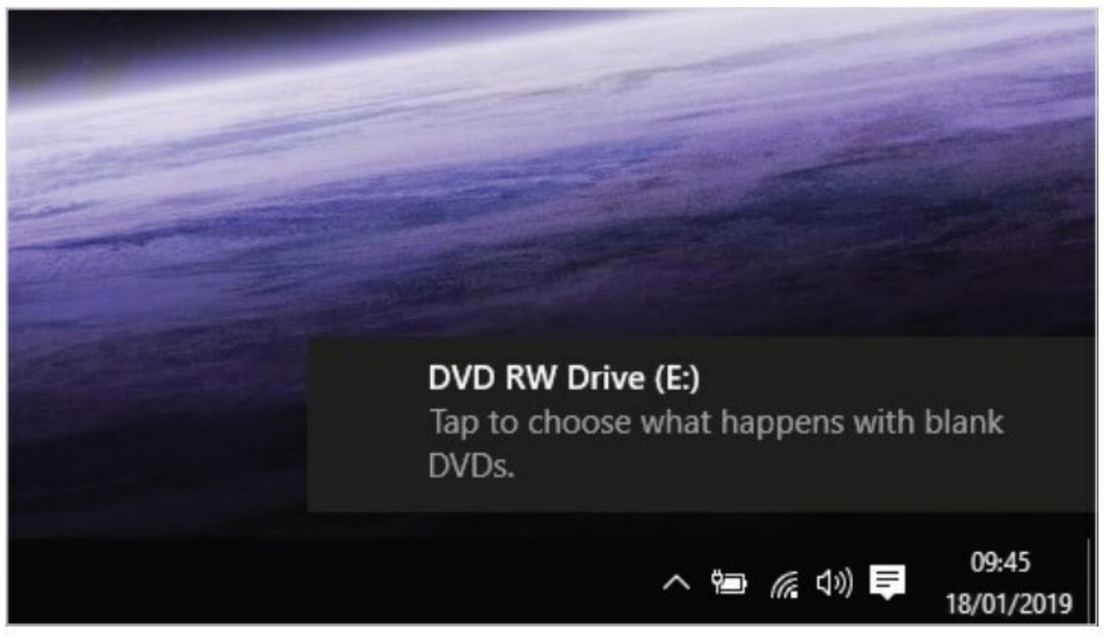
DVD BOOTABLE MEDIA

We're using a Windows 10 PC here to transfer the ISO to a DVD. If you're using a version of Windows from 7 onward the process is extremely easy.

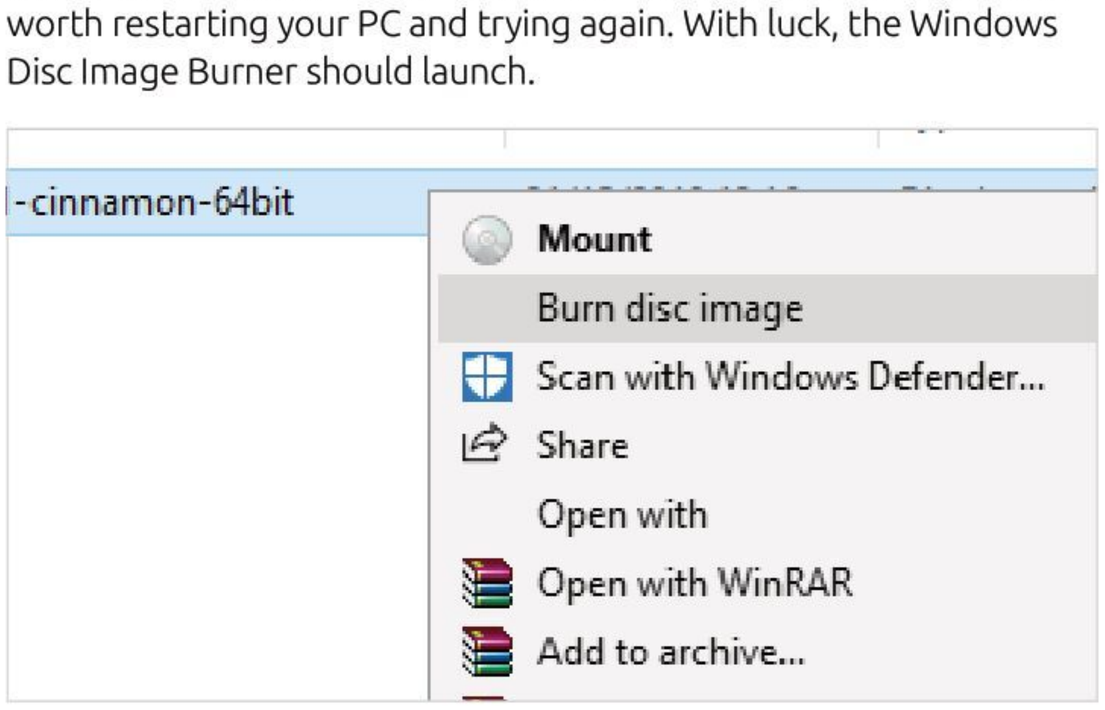
STEP 1 First locate the ISO image of Linux you've already downloaded. You can usually find it in the Downloads folder in Windows 7, 8.1 and 10 computers, unless you specified a different location when saving it.



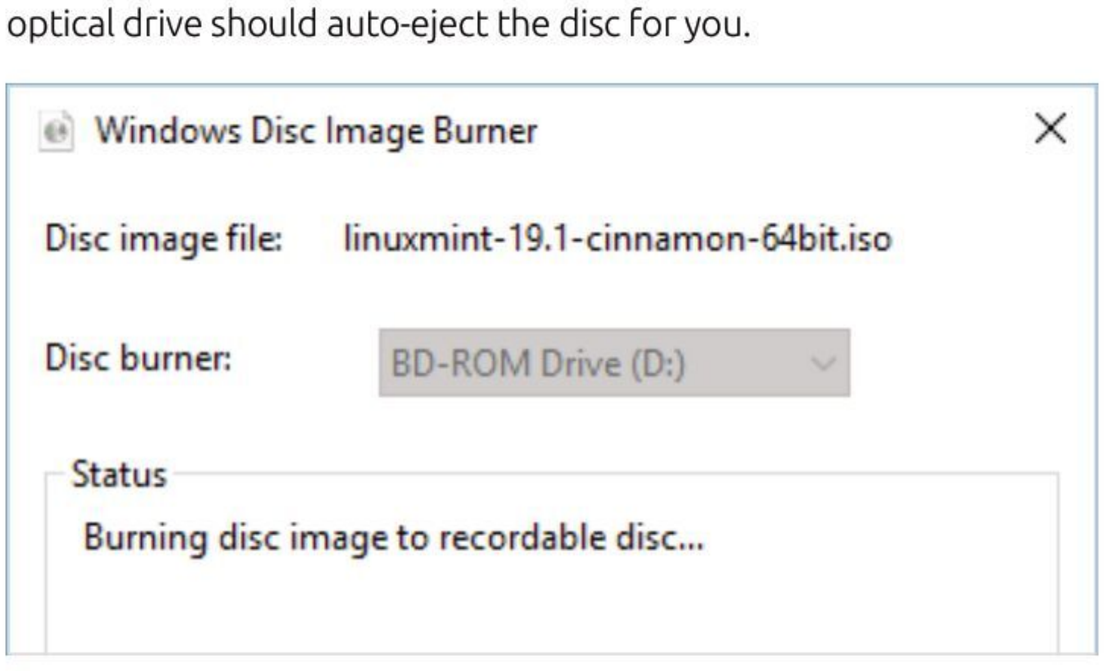
STEP 2 Next insert a recordable DVD disc into your computer's optical drive. After a few seconds, while the disc is read, Windows displays a pop-up message asking you what to do with the newly inserted disc. Ignore this, as we're going to use the built-in image burning function.



STEP 3 Right-click the Linux ISO and from the menu select Burn Disc Image. Depending on the speed of the PC, it may take a few seconds before anything happens. Don't worry too much, unless it takes more than a minute, in which case it might be worth restarting your PC and trying again. With luck, the Windows Disc Image Burner should launch.



STEP 4 With the Windows Disc Image Burner dialogue box open, click on the 'Verify disc after burning' tick box, then the Burn button. The process should take a few minutes, depending on the speed of your PC's optical drive. Once it's complete it runs through the verification stage and when done the optical drive should auto-eject the disc for you.

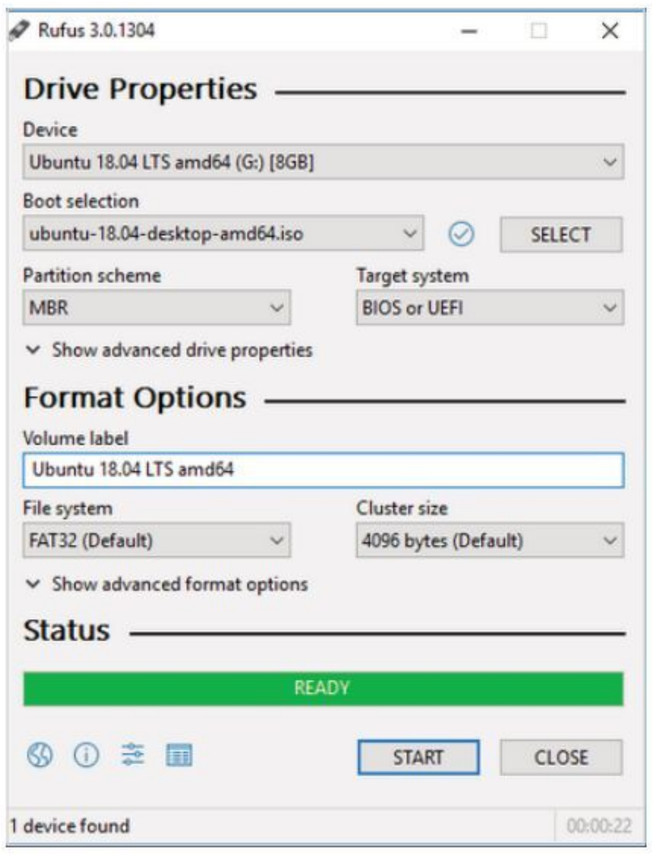


USB BOOTABLE MEDIA

USB media is faster than a DVD and often more convenient, as most modern PCs don't have an optical drive installed. The process of transferring the image is easy but you need a third-party app first and a USB flash drive of 4GB or more.

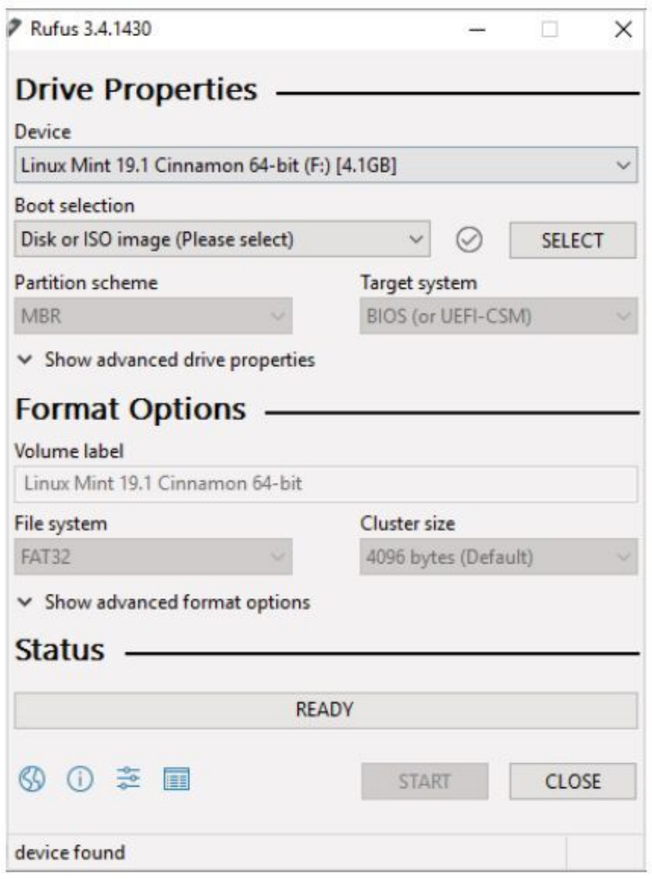
STEP 1

First open up a web browser and go to **www.rufus.akeo.ie/**. Scroll down the page a little and you come to a Download heading, under which is the latest version of Rufus. Left click the link to start the download.



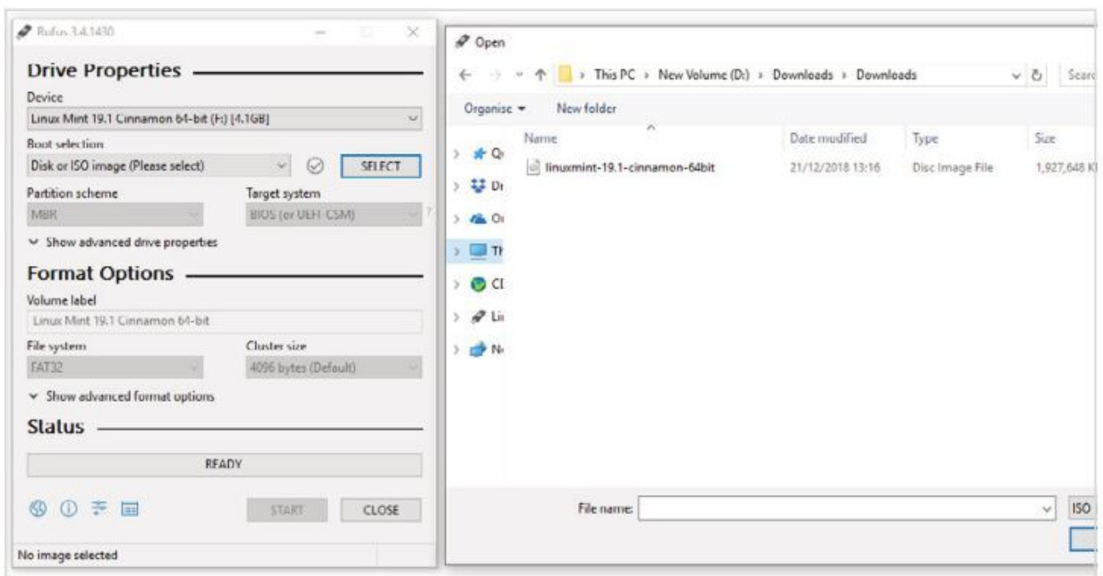
STEP 2

Double-click the downloaded Rufus executable and click Yes to the Windows security question and Yes to checking for updates. With Rufus launched it should have already identified your inserted USB flash drive; if not, just remove and reinsert.



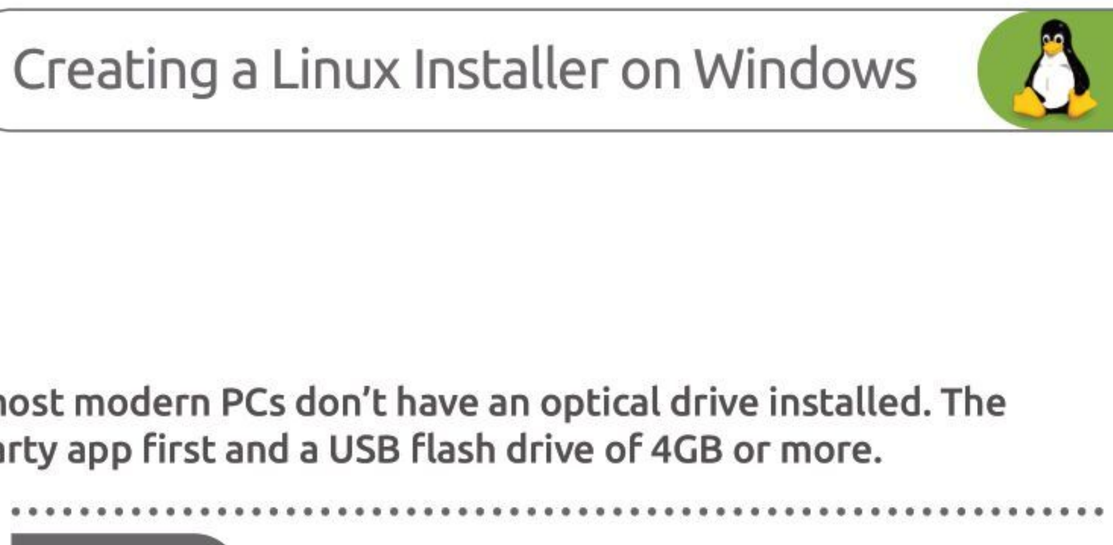
STEP 3

At first glance the Rufus interface can look a little confusing but don't worry, it's really quite simple. To begin with, click on the SELECT button next to the 'Disk or ISO Image (Please select)' pull-down menu. This launches a Windows Explorer window where you can locate and select the Linux ISO.



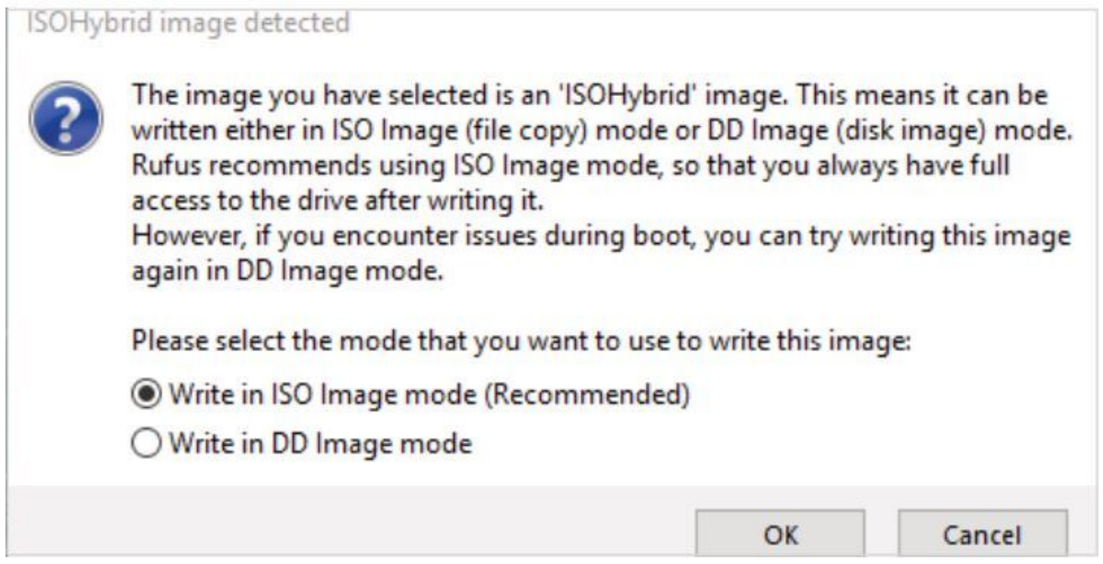
STEP 4

When you're ready, click on the Start button at the bottom of the Rufus app. This may open up another dialogue box asking you to download and use a new version of SysLinux. SysLinux is a selection of boot loaders, used to allow a modern PC to access and boot from a USB flash drive. It is necessary, so if asked click on 'Yes' to continue.



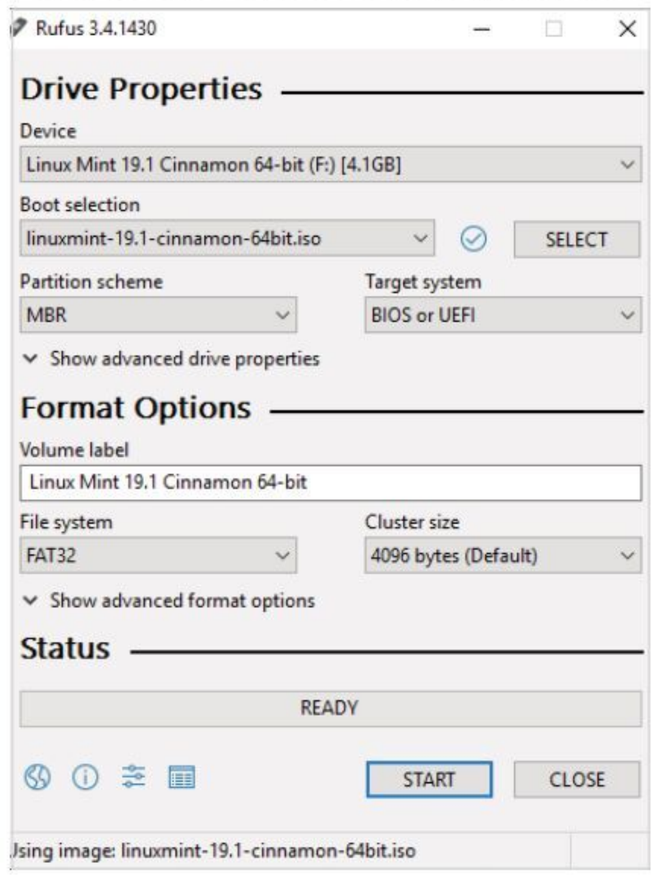
STEP 5

The next step asks which image mode you want the Linux ISO to be written to the USB flash drive in. Both methods work for different situations but generally, the recommended ISO Image Mode is the more popular. Make sure this mode is preselected and click OK to continue, followed by OK again to confirm the action.



STEP 6

The Linux ISO is now transferred to the USB flash drive. The process shouldn't take too long, again depending on the speed of the USB device and the PC. You may find Rufus auto-opens the USB drive in Windows Explorer during the process; don't worry you can minimise or close it if you want. When the process is complete, click on the Close button.



Installing Linux on a PC

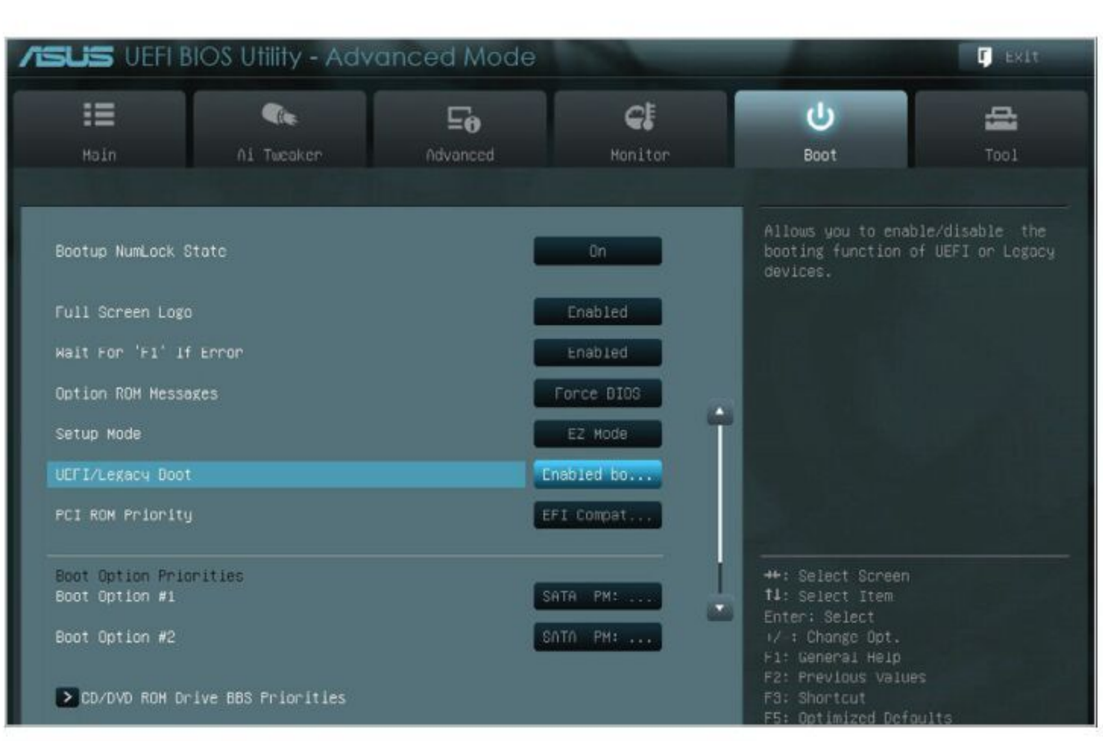
Most Linux distros come as a Live Environment. This means you can boot into an actual, fully-working distro straight from the DVD or USB that you just created. Let's see how that works and how you go about installing Linux from there.

UEFI BIOS

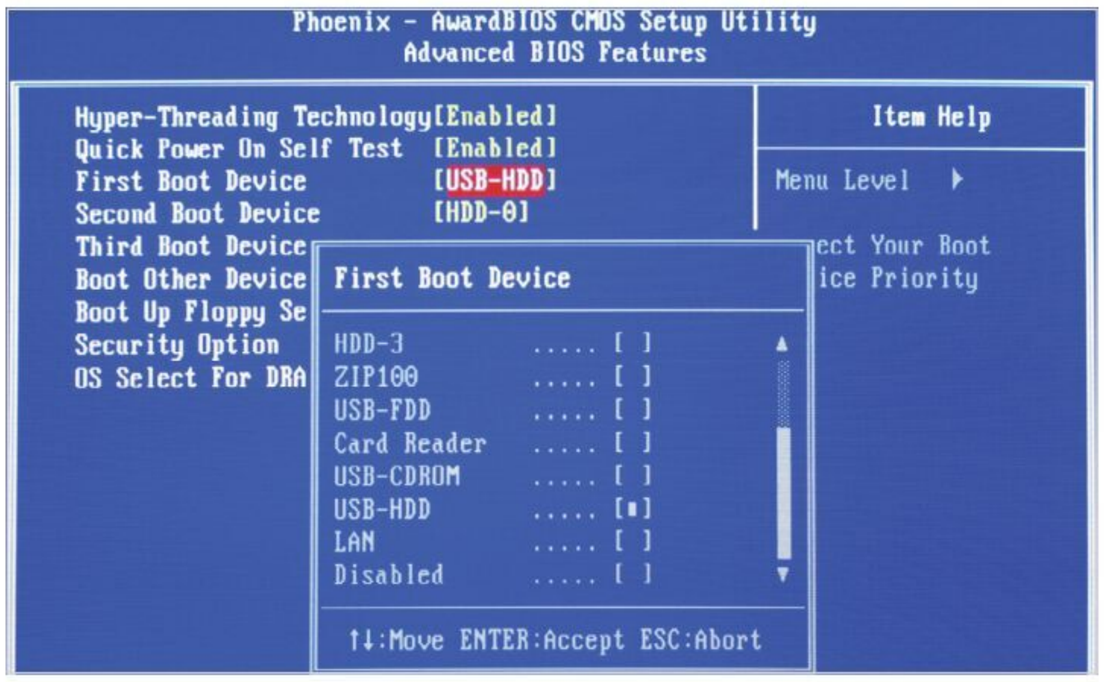
The Unified Extensible Firmware Interface (UEFI) is used to identify hardware and protect a PC during its boot-up process. It replaces the traditional BIOS but can cause issues when installing Linux.

STEP 1 Insert your DVD or USB flash drive into your PC and, if you haven't already, shutdown Windows. In this instance we're using the USB boot media but the process is virtually identical. Start the PC and when prompted press the appropriate keys to enter the BIOS or SETUP, which could for example be: F2, Del or even F12.

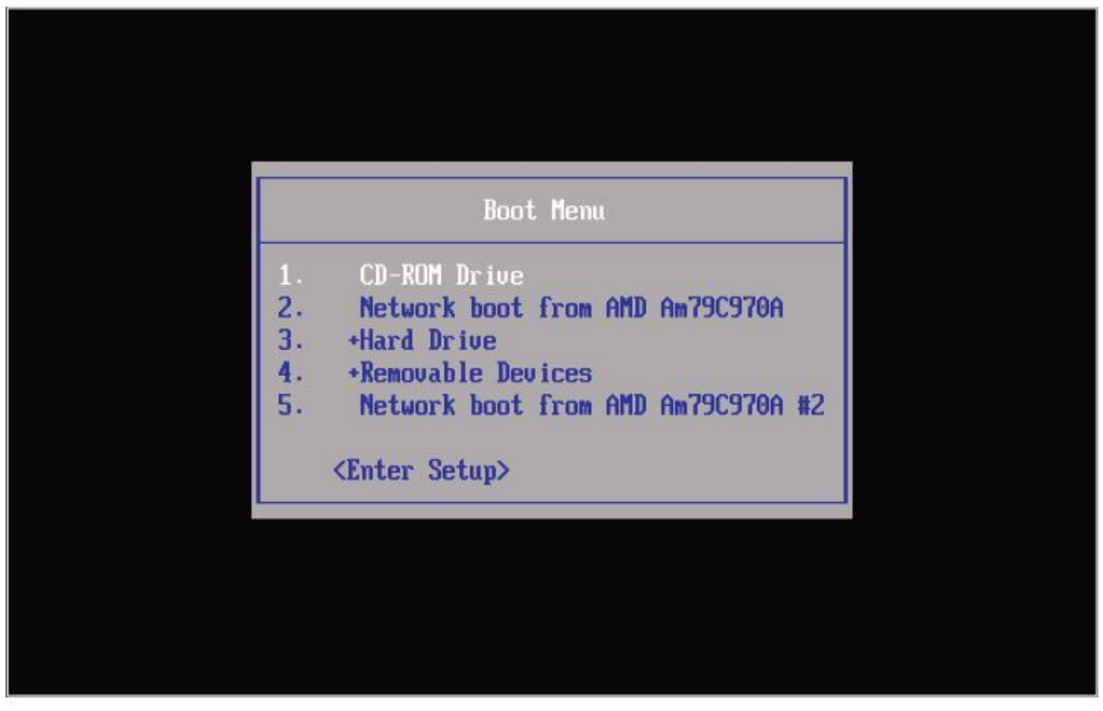
STEP 2 There are different versions of a UEFI BIOS, so covering them all would be impossible. What you're looking for is a section that details the Boot Sequence or Boot Mode. Here you have the option to turn off UEFI and choose Legacy or disable Secure Booting. Most distros work with UEFI but it can be a tricky process to enable it to boot.



STEP 3 With UEFI turned to Legacy mode, there are now two ways of booting into the Live Environment. The first is via the BIOS you're already in. Locate the Boot Sequence and change the first boot device from its original setting, usually Internal HDD or similar, to: USB Storage Device for the USB media option, or DVD Drive for the DVD media option.



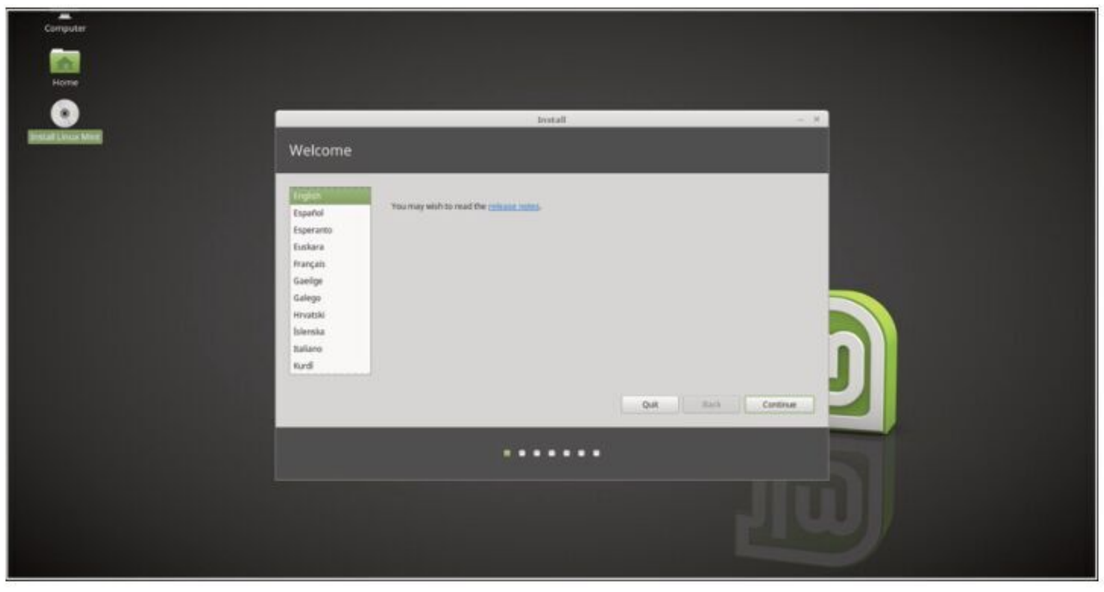
STEP 4 Alternatively use the Boot Option Menu. With this option you can press F12 (or something similar) to display a list of boot media options; from there, you can choose the appropriate boot media. Either way, you can now Save and Exit the BIOS by navigating to the Save & Exit option and choosing 'Save Changes and Exit'.



INSTALLING LINUX

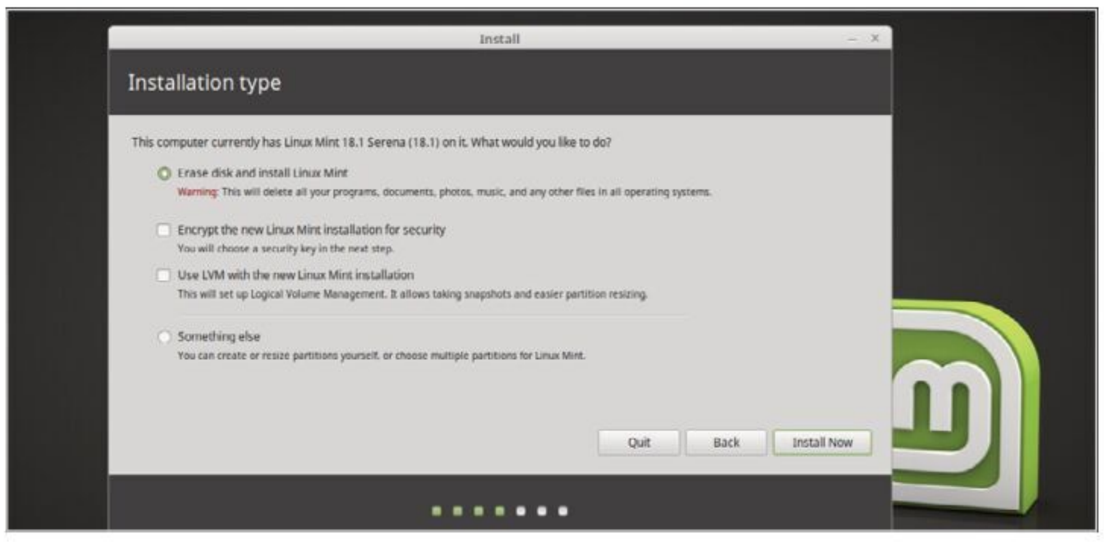
Once the Live Environment has booted, you see the option to install the distro to your computer. Have a look around and when you're ready, look for the Install option on the desktop.

STEP 1 Providing you're connected to the internet (if not, then do so now) and you're in the Live Environment, start the installation process by double-clicking on the Install Linux Mint icon on the desktop. Other distros display their own name, of course, but the process is the same. Click Continue when you're ready.

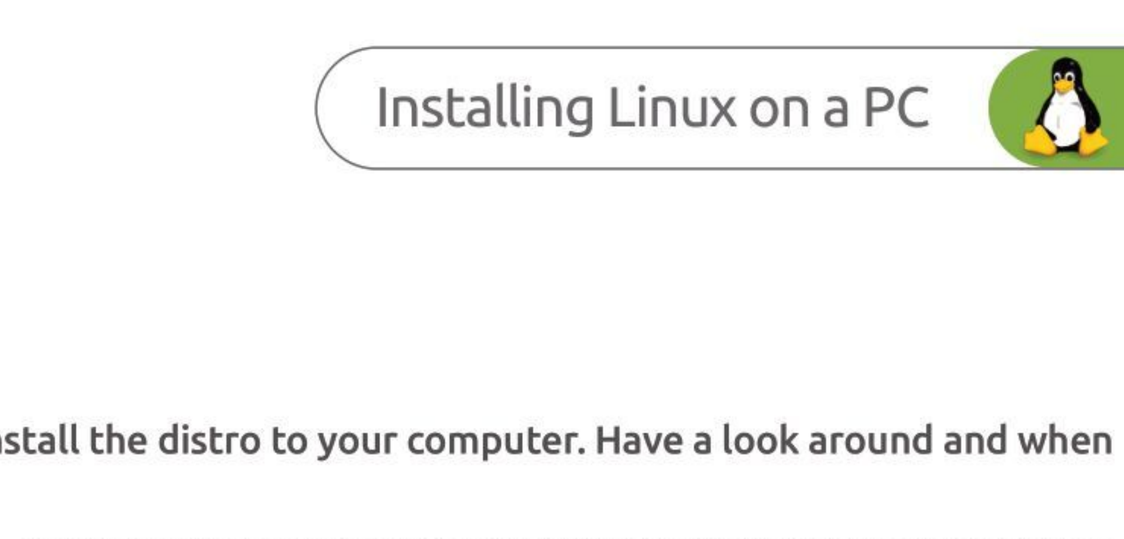


STEP 2 While the installation process is very similar across most Linux distros, some offer different questions during the installation. Generally, the questions aren't too difficult, nothing very technical, but some such as 'Installing third-party software…' can be confusing. In this case you can click Continue but if you're unsure, have an Internet-connected device available to ask any questions.

STEP 3 When installing a new operating system it's recommended that you wipe the old OS, replacing it with the new. When you reach this stage of the installation process, ensure the 'Erase disk and install Linux…' option is selected. NOTE: This completely wipes Windows 10 from your computer, so make sure you have backups of all your personal files and data.

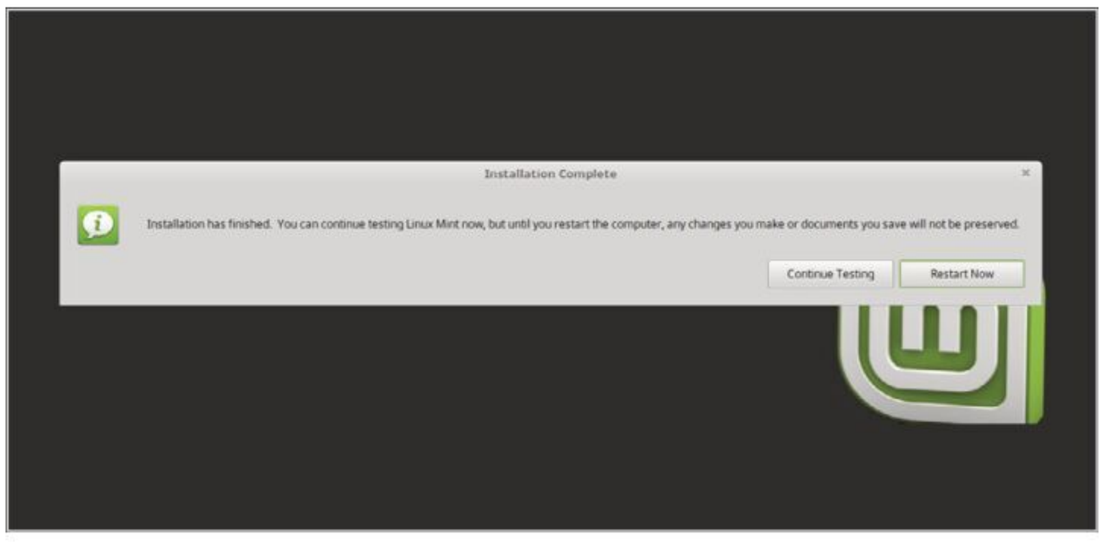


STEP 4 Before the installation process begins, you're asked if the choice you made regarding the erasure of the hard drive is correct. This is your last chance to back out. If you're certain you don't mind wiping everything and starting again with Linux Mint, click Continue. If you need to backup your files remove the Linux disc/USB, reboot, backup and start again.



STEP 5 Eventually you are asked to set up your Linux username and password. Enter your Name to begin with, then the Computer Name, which is the name it is identified on the network as. Next choose a Username, followed by a good Password. You can tick the Login Automatically option but leave the Encrypt Home Folder option for now.

STEP 6 The installation process can be quick, and there may be more questions to answer, or it may simply start installing Linux based on your previous answers. Either way, you end up being asked to Continue Testing the Live Environment or Restarting to use the newly installed OS. If you're ready to use Linux, then click Restart Now.



Installing a Virtual Environment

A Virtual Environment is a simulated computer system. Using a Virtual Machine, you can mimic a standard PC and install an entire operating system to it without affecting the one installed on your computer. It's a great way to test and use Linux, while still having Windows 10 as your main OS.

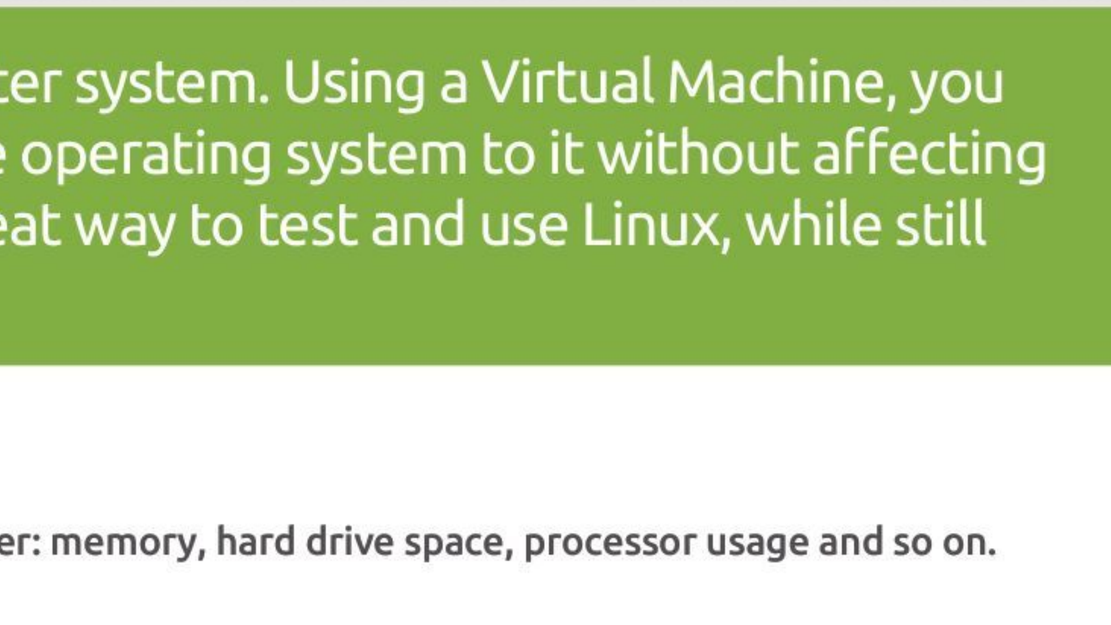
GOING VIRTUAL

Using a Virtual Machine (VM) takes resources from your computer: memory, hard drive space, processor usage and so on. Make sure you have enough of each before commencing.

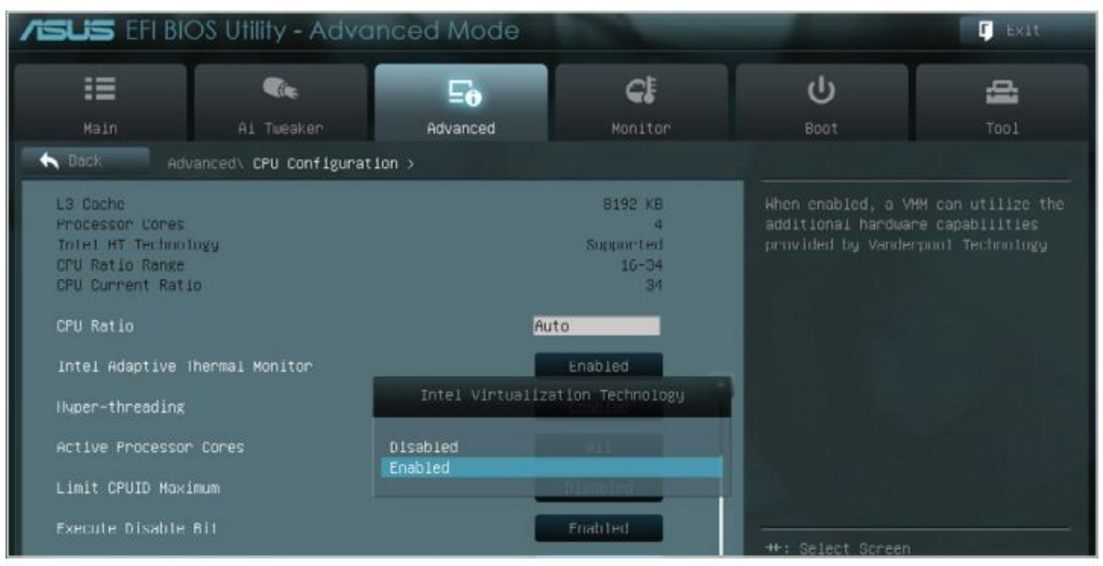
STEP 1 We're using VirtualBox in this instance, as it's one of the easiest virtual environments to get to grips with. Enter www.virtualbox.org and click on 'Download VirtualBox'. This takes you to the main download page. Locate the correct host for your system: Windows or Mac, the Host being the current installed operating system, and click the link to begin the download.

STEP 2 Next, while still at the VirtualBox download page, locate the VirtualBox Extension Pack link. The Extension Pack supports USB devices as well as numerous other extras that can help make the VM environment a more accurate emulation of a 'real' computer.

STEP 3 With the correct packages downloaded, and before you install anything, you need to make sure that the computer you're using is able to host a VM. To do this, reboot the computer and enter the BIOS. When the computer starts up, press the Del, F2 or whichever key is necessary to Enter Setup.

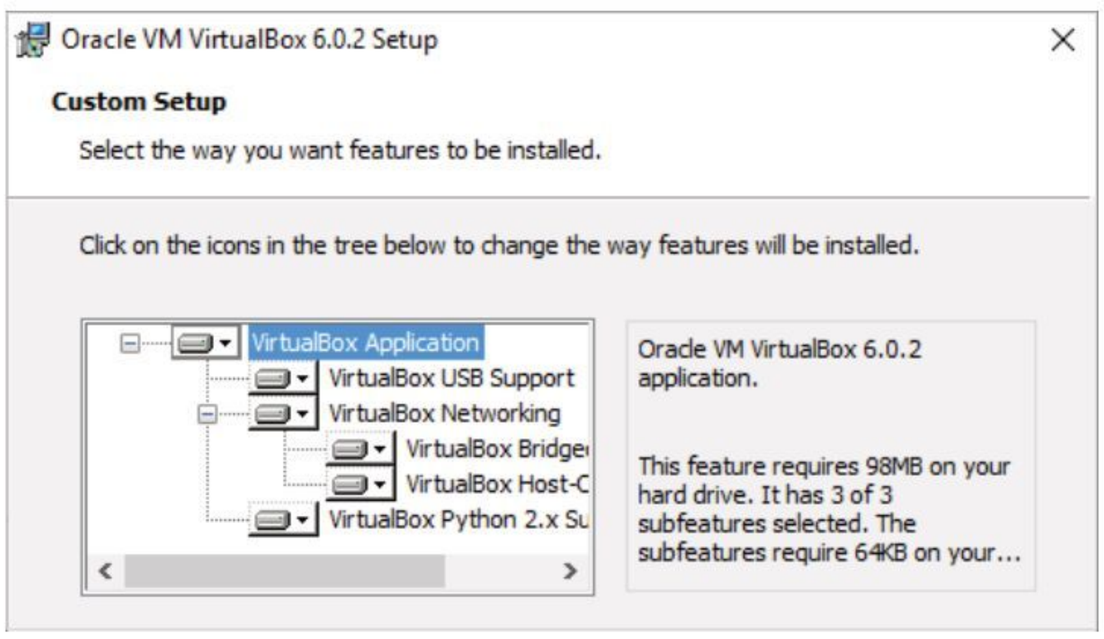


STEP 4 Each BIOS is laid out differently and it's very difficult to assess where to look in each personal example. However, as a general rule of thumb, you're looking for Intel Virtualisation Technology, or simply Virtualisation, found usually within the Advanced section of the BIOS. When you've located it, Enable it, save the settings, exit the BIOS and reboot the computer.

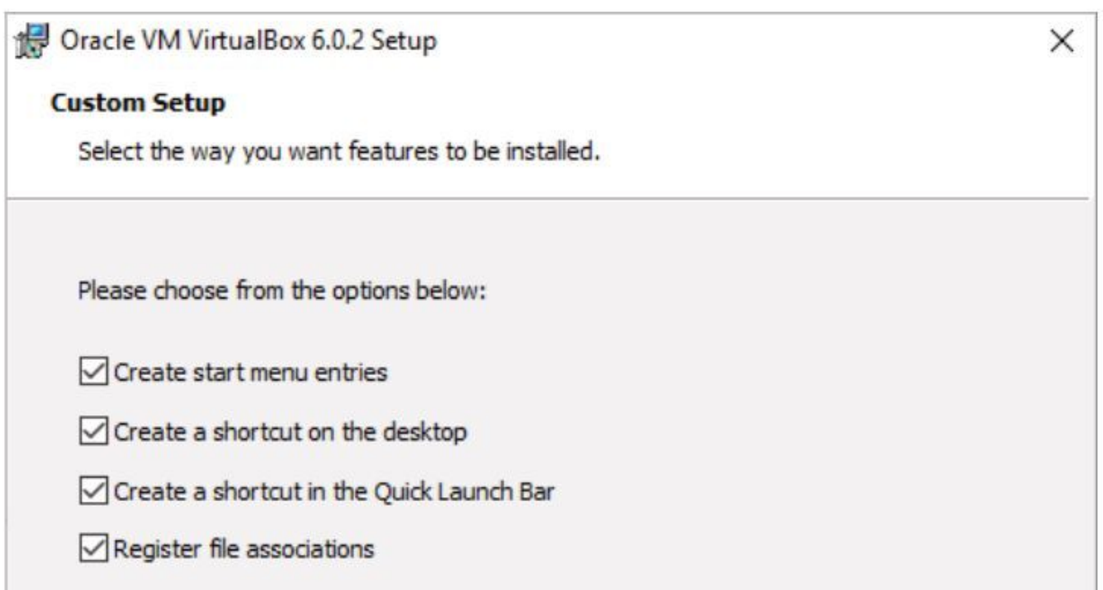


STEP 5 With the computer back up and running, locate the downloaded main VirtualBox application and double-click to begin the installation process. Click Next to continue, when you're ready.

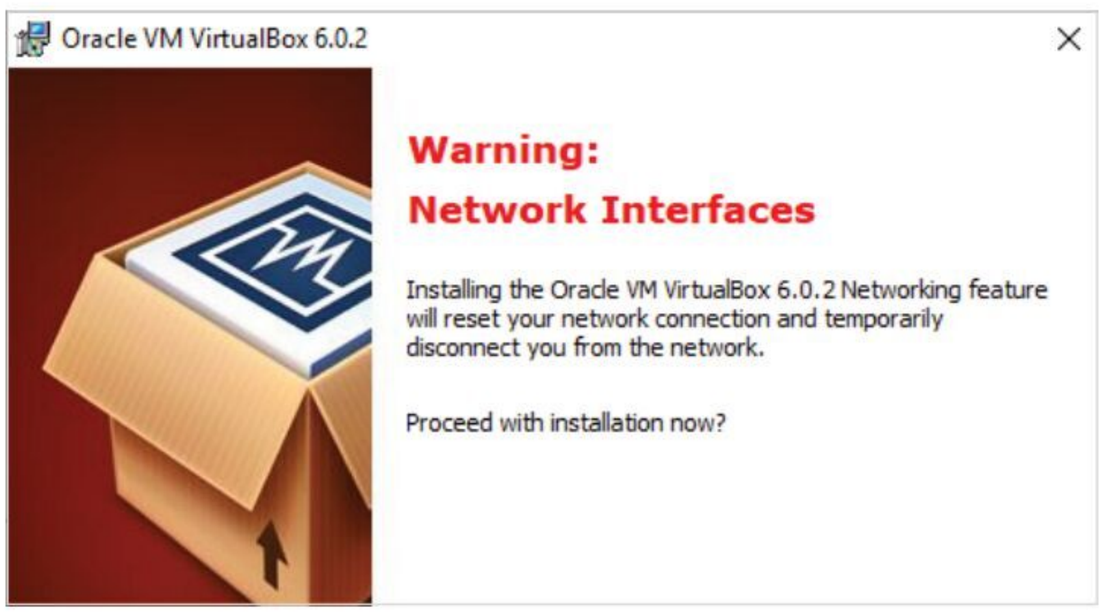
STEP 6 The default installation location of VirtualBox should satisfy most users but if you have any special location requirements click on the 'Browse' button and change the install folder. Then, make sure that all the icons in the VirtualBox feature tree are selected, i.e. none of them have a red X next to them. Click Next to move on.



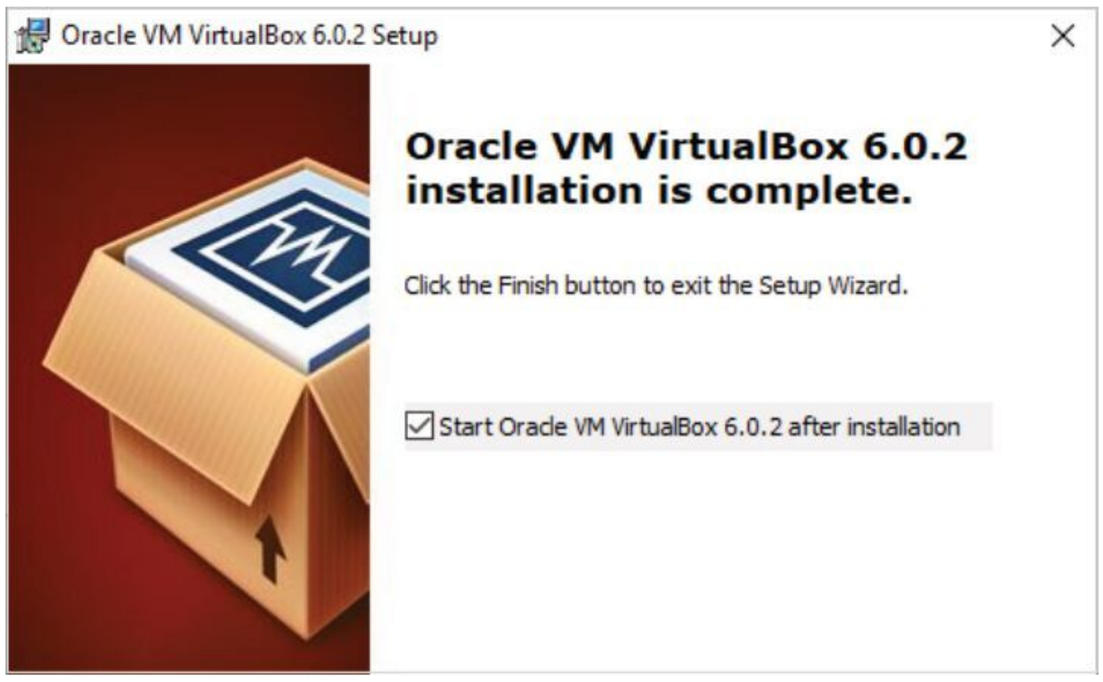
STEP 7 This section can be left alone to the defaults, should you wish. It simply makes life a little easier when dealing with VMs, especially when dealing with downloaded VMs, which you may encounter in the future. Again, clicking Next moves you on to the next stage.



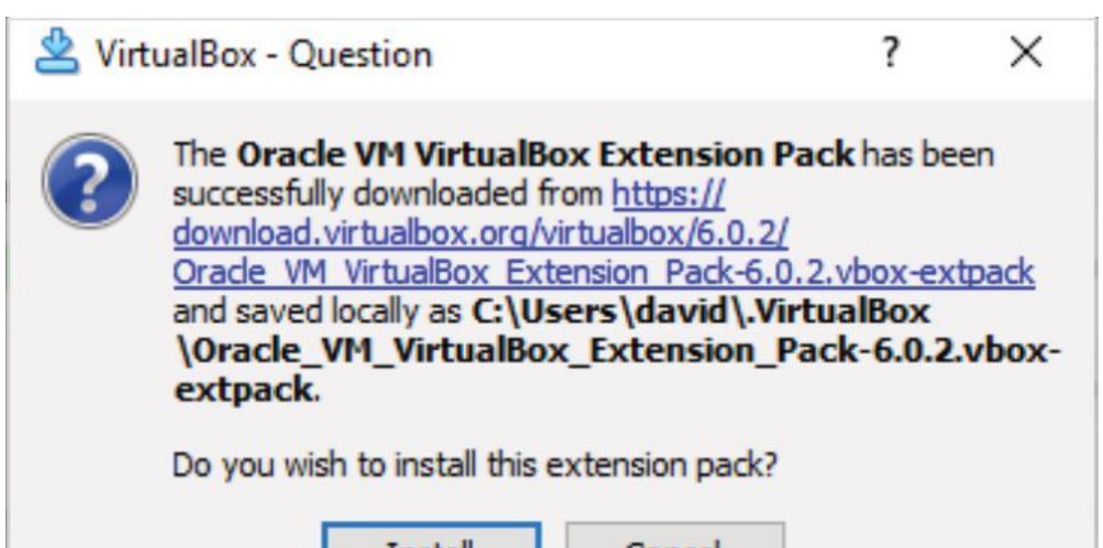
STEP 8 When installing VirtualBox your network connection is disabled for a very brief period. This is due to VirtualBox creating a linked, virtual network connection so that any VM installed is able to access the Internet, and your home network resources, via the computer's already established network connection. Click Yes, then Install to begin the installation.



STEP 9 You'll probably be asked by Windows to accept a security notification. Click Yes for this and you may encounter a dialogue box asking you to trust the installation from Oracle; again, click yes and accept the installation of the VirtualBox application. When it's complete, click finish to start VirtualBox.



STEP 10 With VirtualBox up and running you can now install the VirtualBox Extension Pack. Locate the downloaded add-on and double-click. There may be a short pause while VirtualBox analyses the pack but you eventually receive a message to install it; obviously click Install to begin the process, scroll down the next screen to accept the agreement and click I Agree.



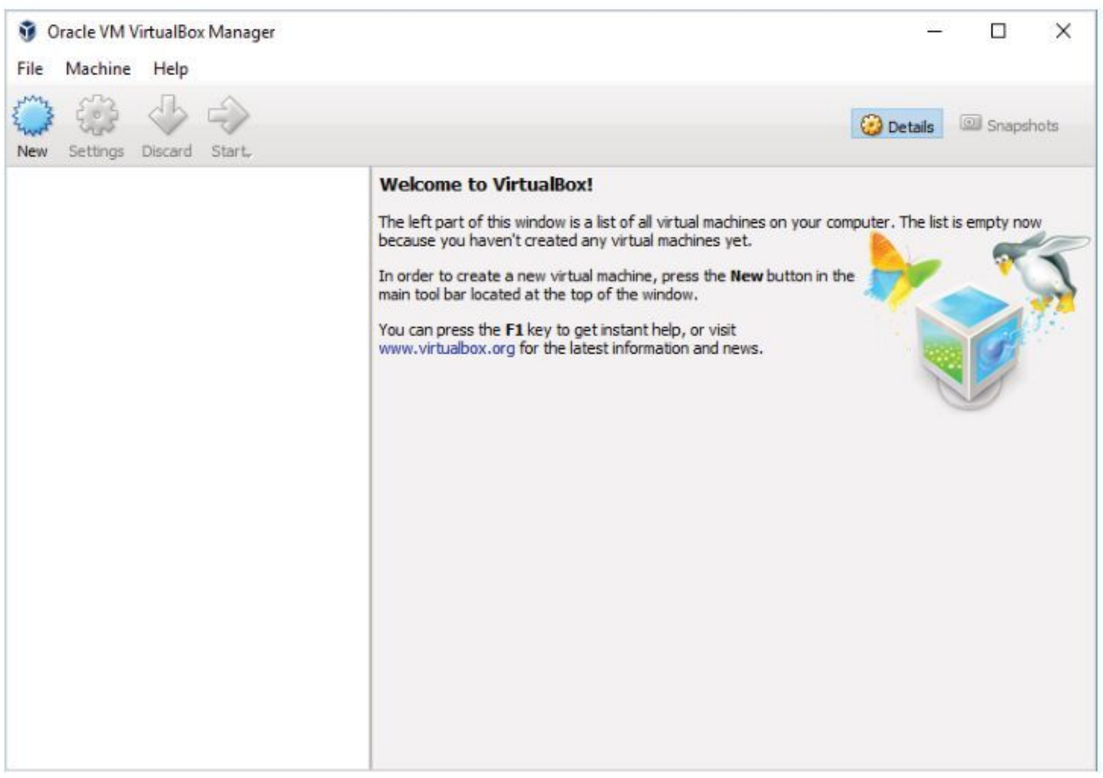
Installing Linux in a Virtual Environment

With Oracle's VirtualBox now up and running, the next task is to create the Virtual Machine (VM) environment into which you install Linux. This process won't affect your currently installed operating system, which is why a VM is a great choice.

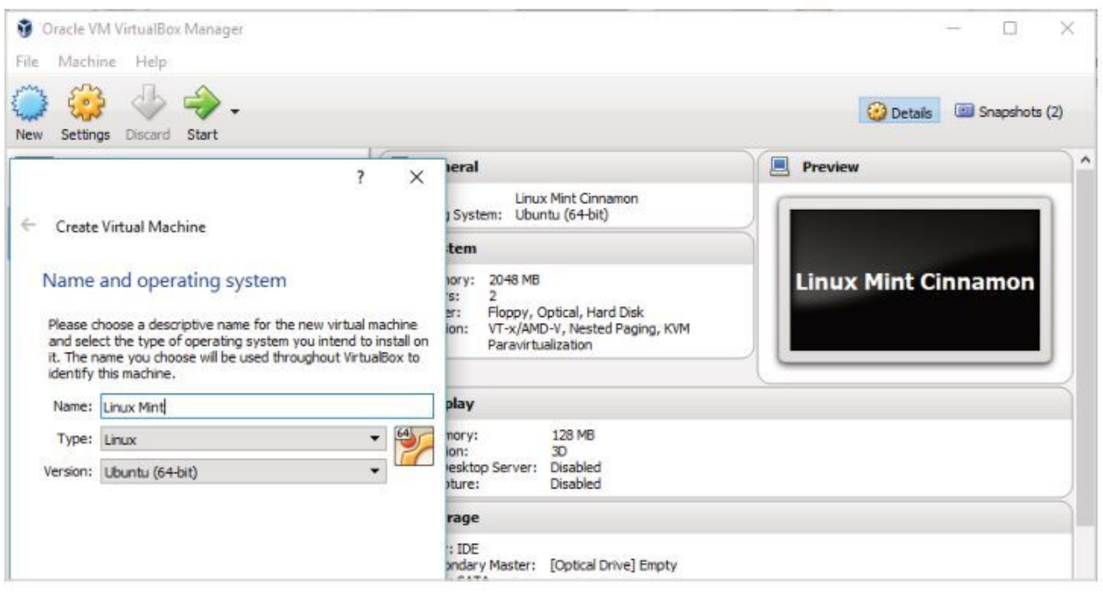
CREATING THE VM

There are plenty of options to choose from when creating a VM. For now though, let's set up a VM adequate to run Mint Cinnamon and perform well.

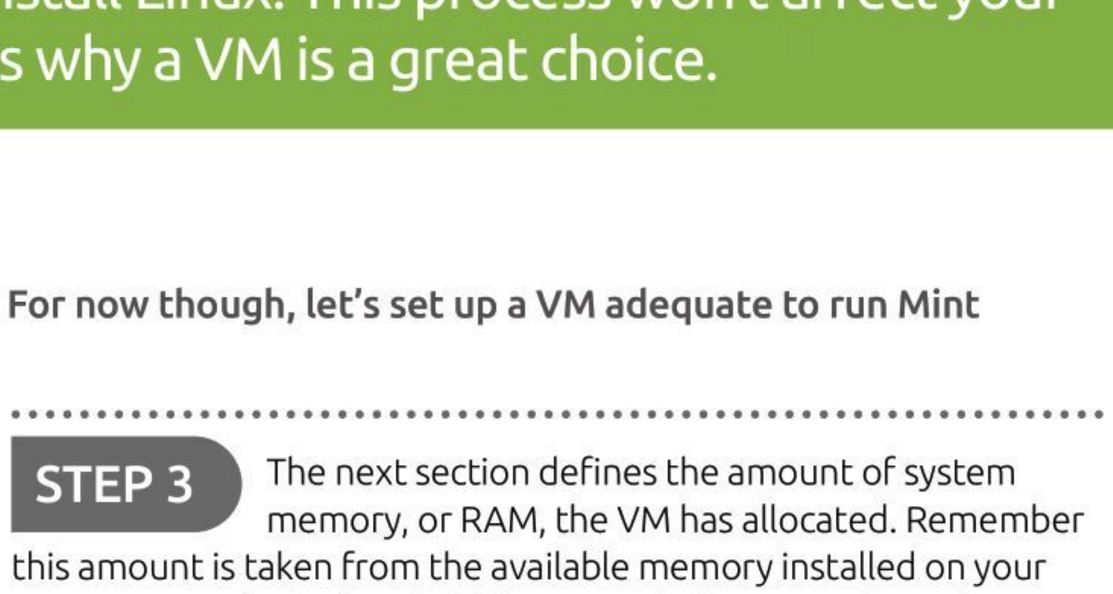
STEP 1 With VirtualBox open, click on the New icon in the top right of the app. This opens the new VM Wizard.



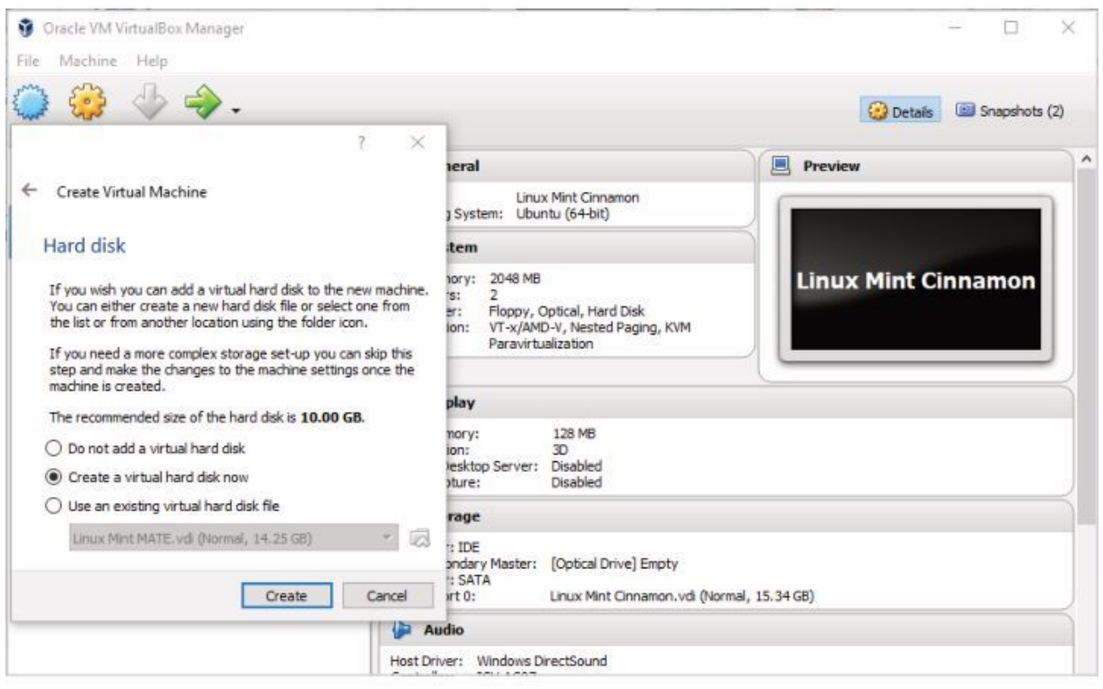
STEP 2 In the box next to Name, type Linux Mint and VirtualBox should automatically choose Linux as the Type and Ubuntu (64-bit) as the Version; if not then use the drop-down boxes to select the correct settings (remember Mint mainstream is based on Ubuntu). Click Next when you're ready to proceed.



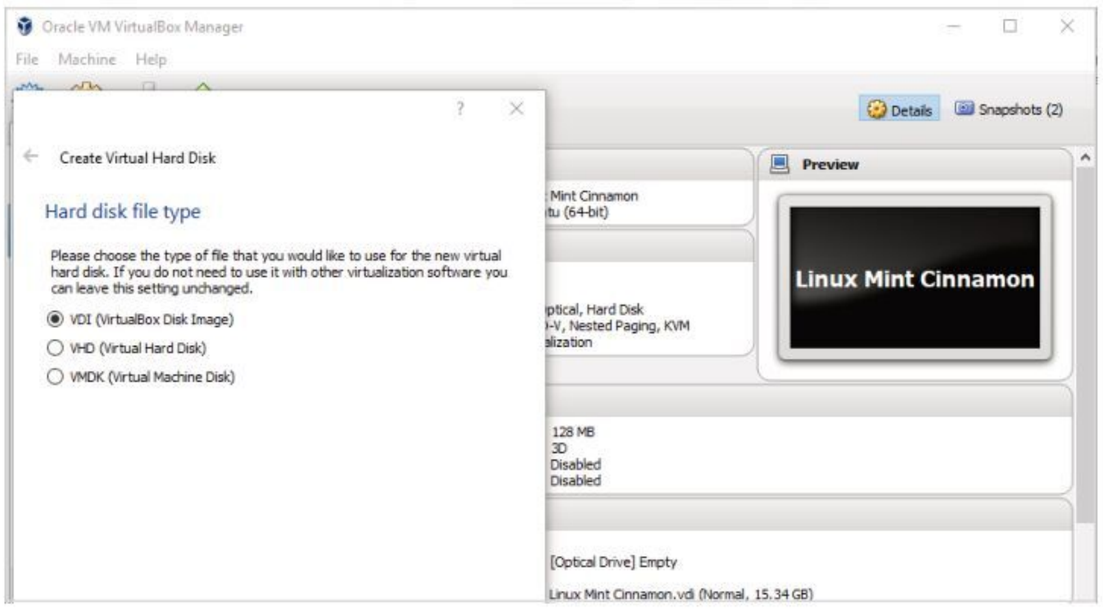
STEP 3 The next section defines the amount of system memory, or RAM, the VM has allocated. Remember this amount is taken from the available memory installed on your computer, so don't give the VM too much. For example, we have 8GB of memory installed and we're giving 2GB to the VM. When you're ready, click Next to continue.



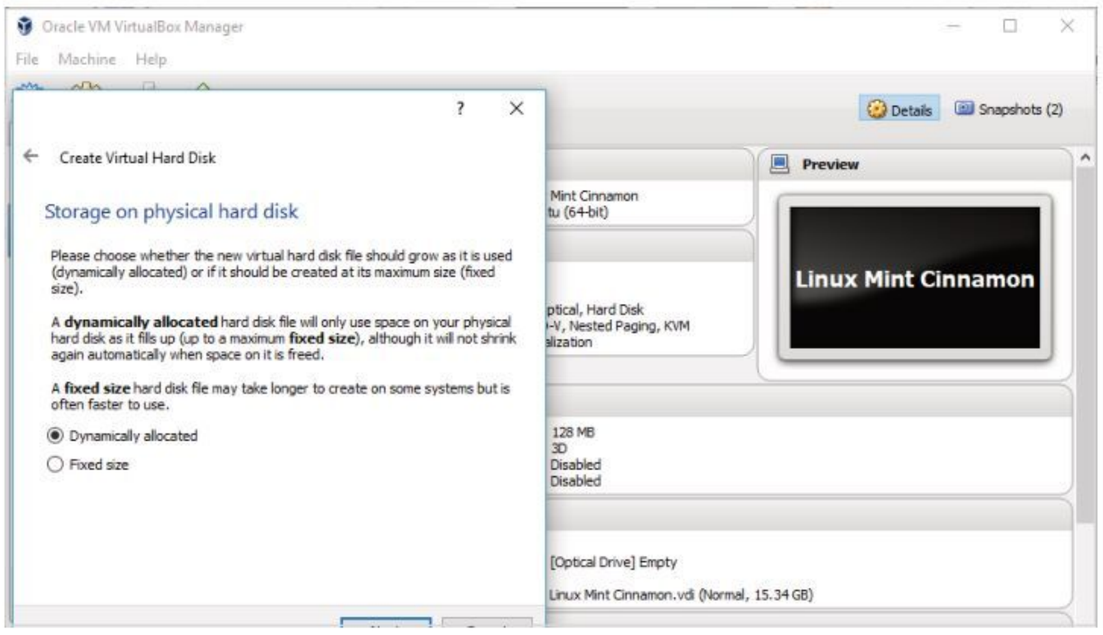
STEP 4 This section is where you start to create the virtual hard disk that the VM uses to install Mint on to. The default option, 'Create a virtual hard disk now', is the one we're using. Click Create to move on.



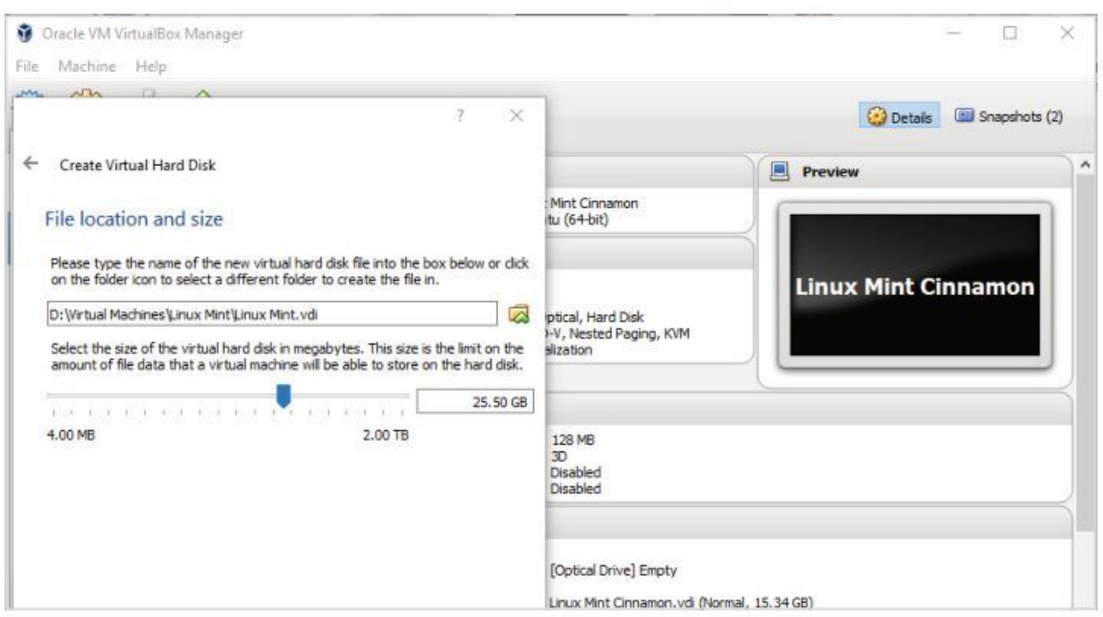
STEP 5 The pop-up window that appears after clicking Create is asking you what type of virtual hard disk you want to create. We're going to use the default VDI (VirtualBox Disk Image) in this case, as the others are often used to move VMs from one VM application to the next. Make sure VDI is selected and click Next.



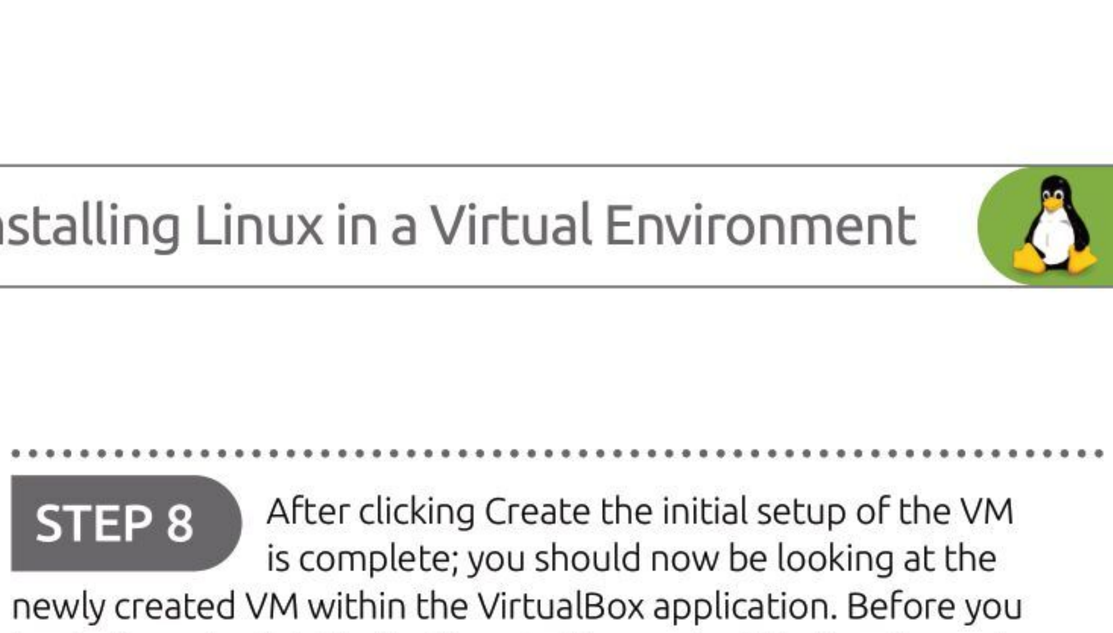
STEP 6 The question of whether to opt for Dynamically or Fixed sized virtual hard disks may come across as being somewhat confusing to the newcomer. A Dynamically Allocated virtual hard disk is a more flexible storage management option and won't take up much space within your physical hard disk to begin with. Ensure Dynamically Allocated is selected and click Next.



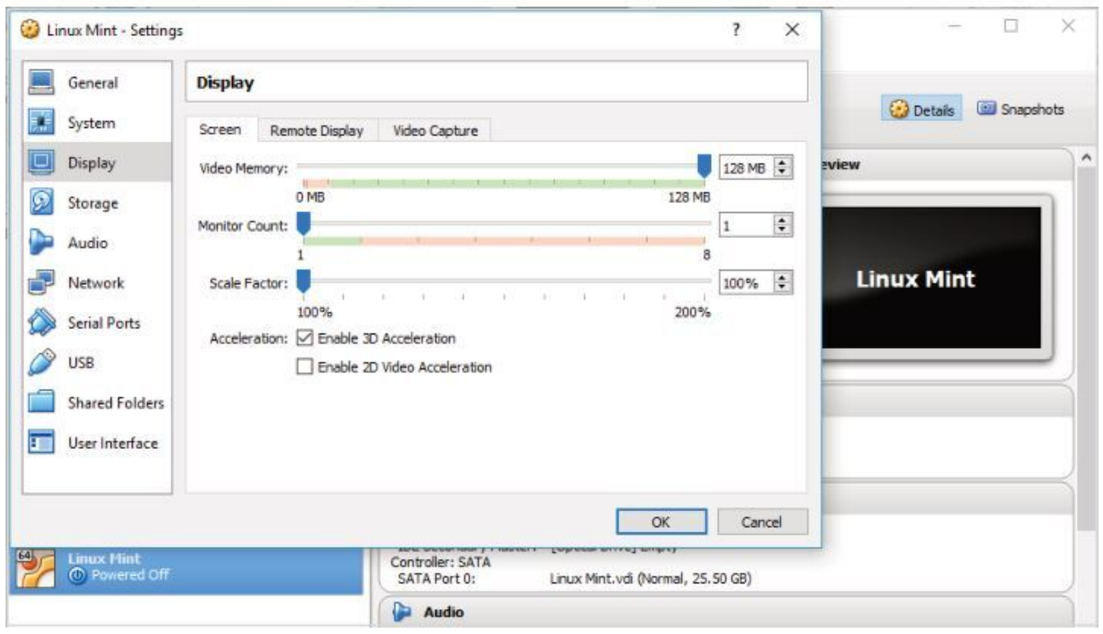
STEP 7 The virtual hard disk is a single folder, up to the size you state in this section. Ensure the location of the virtual hard disk, on your computer, has enough free space available. For example, we've used a bigger storage option on our D:\ drive, named it Linux Mint and allocated 25.50GB of space to the virtual hard disk.



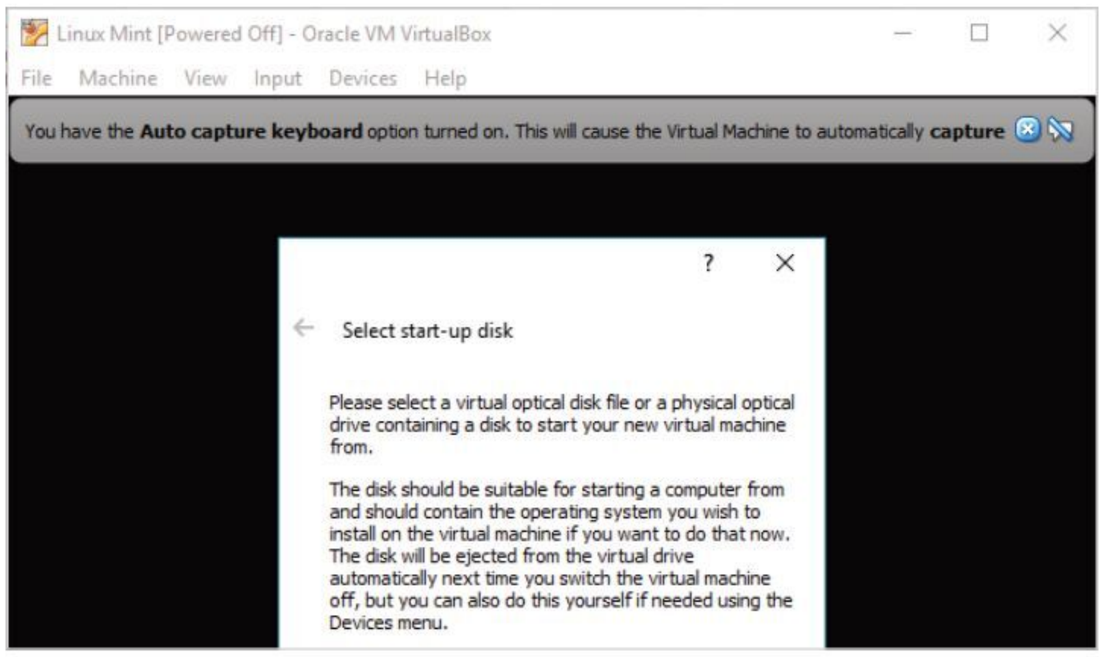
STEP 8 After clicking Create the initial setup of the VM is complete; you should now be looking at the newly created VM within the VirtualBox application. Before you begin though, click the Settings button and within the General section click the Advanced tab. Using the pull-down menus, choose 'Bidirectional' for both Shared Clipboard and Drag'n'Drop.



STEP 9 Follow that by clicking on the System section, then the Processor tab. Depending on your CPU, allocate as many cores as you can without detriment to your host system; we've opted for two CPUs. Now click on the Display section, slide the Video Memory up to the maximum and tick 'Enable 3D Acceleration'. Click OK to commit the new settings.



STEP 10 Click on the Start button and use the explorer button (a folder with a green arrow) in the 'Select Start-up Disk' window to locate the downloaded ISO of Mint; then click Start to boot the VM with the Linux Mint Live Environment. You can now install Linux as per the standard PC installation requirements.



Getting to Know Linux

"Anyone can build a fast processor. The trick is to build a fast system."

– Seymour Cray (Electrical Engineer, and designer/founder of Cray Supercomputers)

We've used Linux Mint as a guideline here, as it's an easy to use distro and perfect for former Windows users. It's also one of the most developed and well documented Linux distros as well as having some fantastic configuration options.

In this section, we introduce the Linux Mint Cinnamon Menu and Desktop Environment, how it works and what you can do to customise it. Want to create another user or even discover how to become anonymous online? Then read on.

30	Introduction to the Cinnamon Menu
32	Navigating the Cinnamon Desktop
34	10 Things to do After Installing Linux Mint
36	Did You Know…Apollo 11
38	Creating Users
40	Customising the Desktop
42	Becoming Anonymous Online

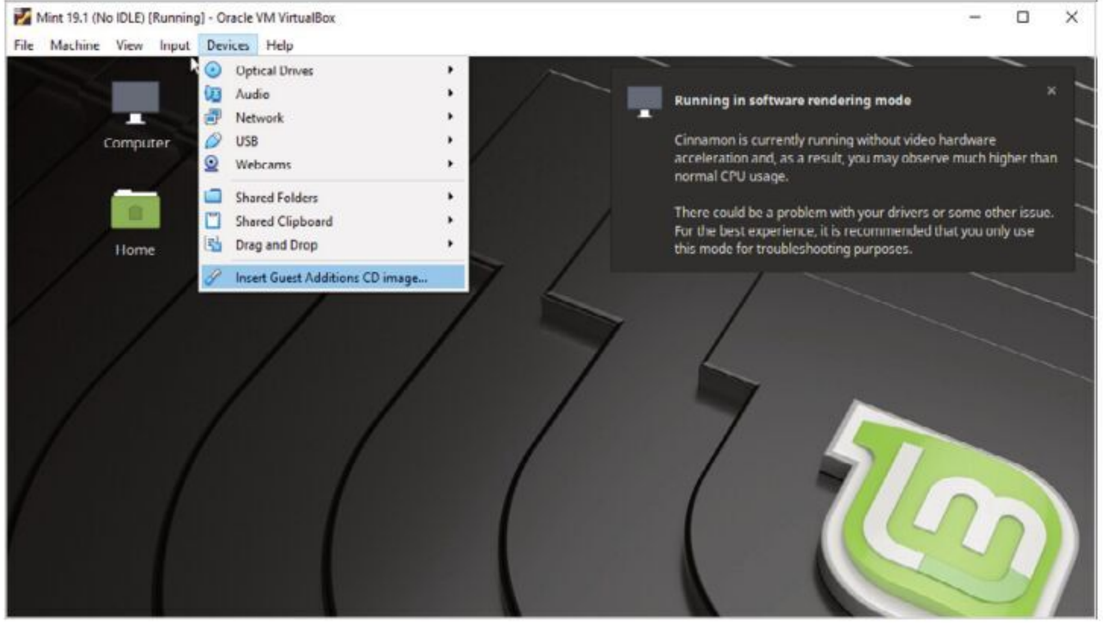
Introduction to the Cinnamon Menu

Now that you have Linux installed it's time to have a good look around. First though, if you're using Virtualbox you'll have a notification regarding Software Rendering; here's how to fix it.

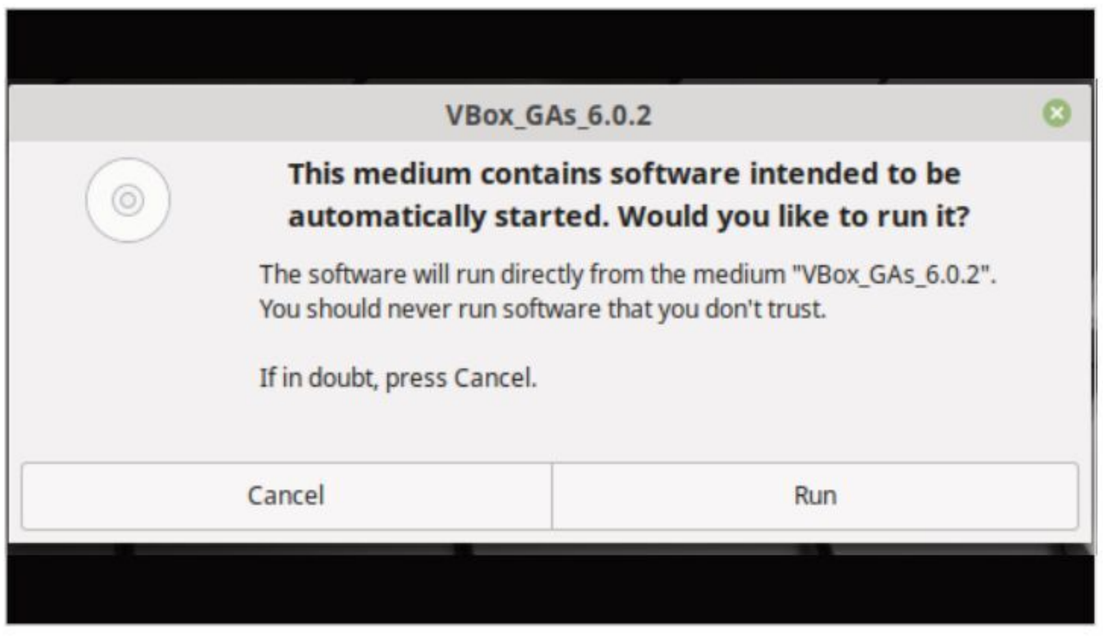
CINNAMON VIRTUALBOX FIX

You've already looked at some list functions, using .insert, .remove, and .pop but there are also functions that can be applied to strings.

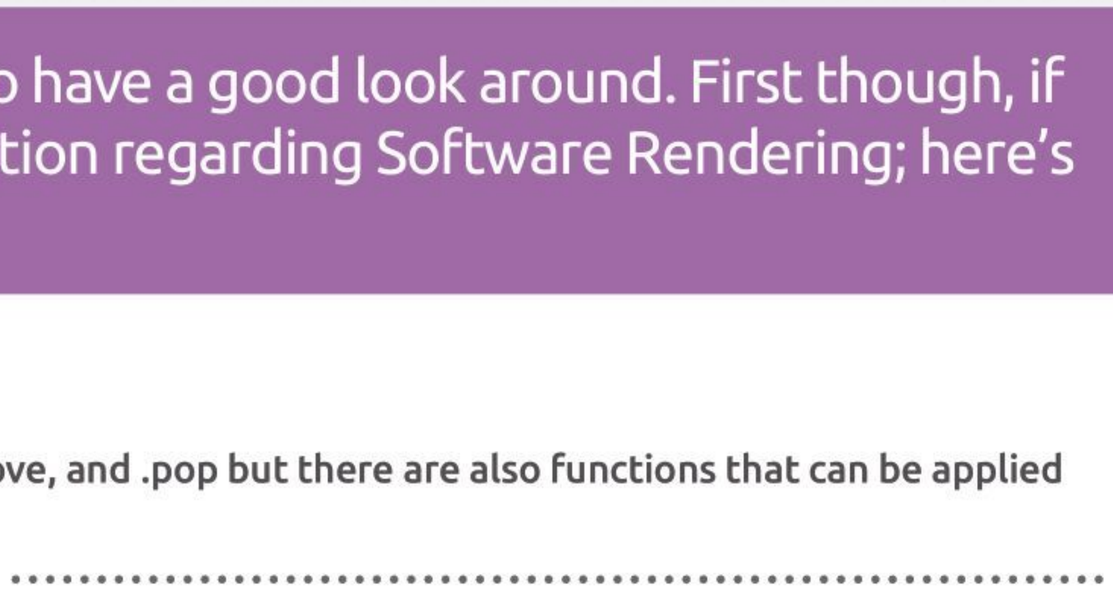
STEP 1 Before we begin, we're assuming you're having this issue within Virtualbox. The Software Rendering message appears in the top right of the desktop. To fix this, click on Devices in the Virtualbox window, followed by Insert Guest Additions CD Image.



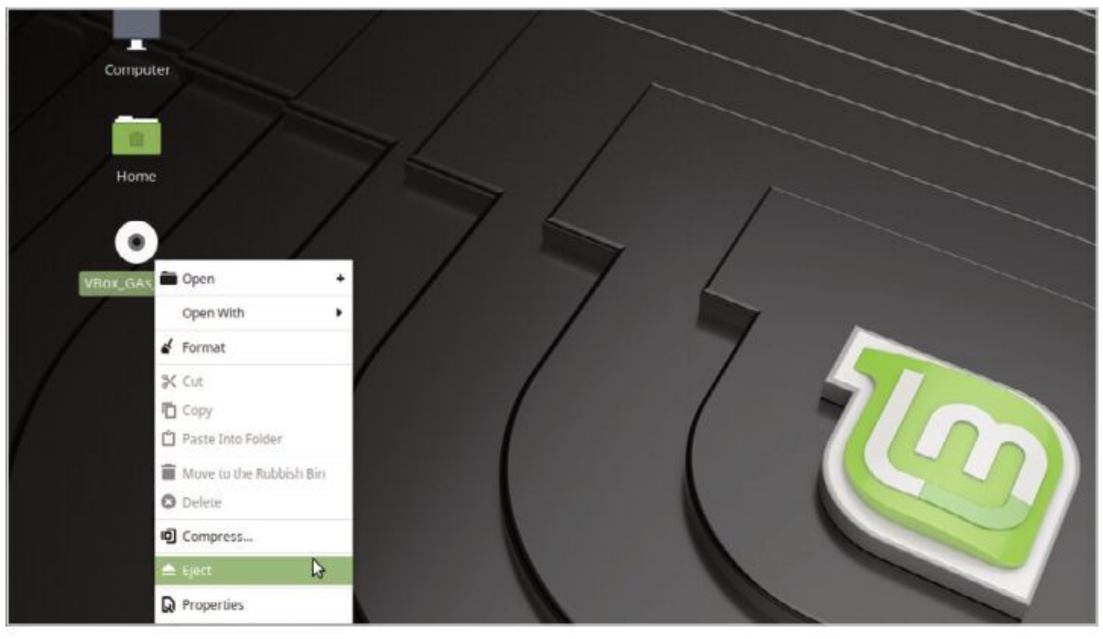
STEP 2 The Guest Additions CD contains drivers for Virtualbox, including the virtual video hardware. When it's loaded in, you get a 'software needs running' notification box with two options, Cancel and Run. Click the Run button and enter your Linux user password.



STEP 3 After a moment or two you're automatically dropped into a command line view, called the Terminal in Linux. This details the installation of the new VirtualBox drivers, removing any old drivers it has detected, and installing the latest versions. It won't take long and when it's done you will be asked to hit Return to exit the Terminal.



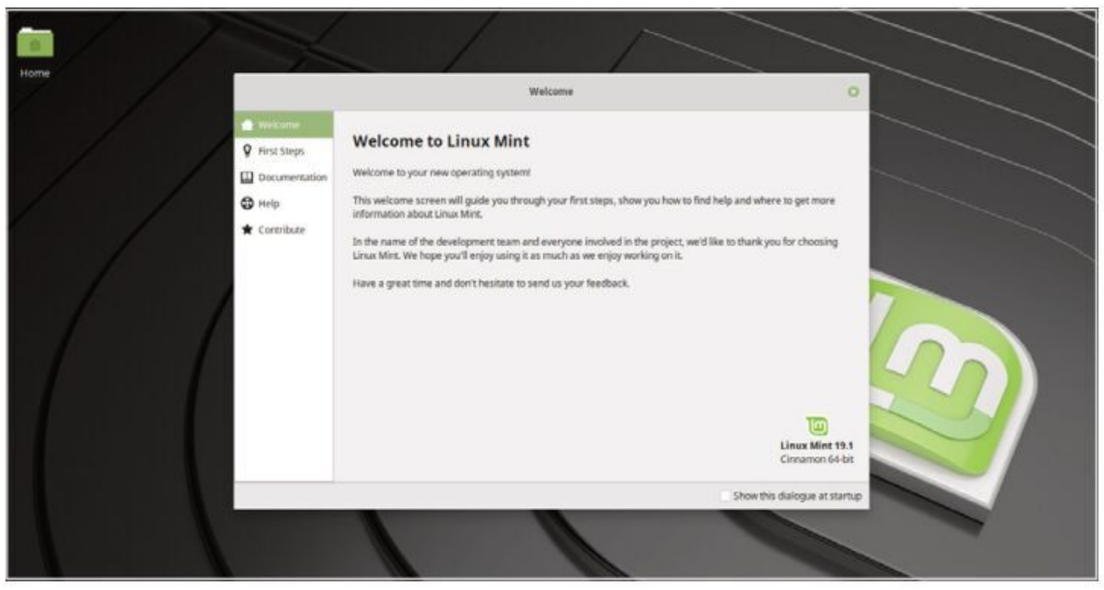
STEP 4 The Virtualbox Additions CD icon is on the desktop, so right-click it and then scroll down the menu to click Eject. You can now restart Linux Mint by clicking the Menu, the bottom icon in the strip to the left, then the Restart button. This reboots Mint and the problem is fixed.



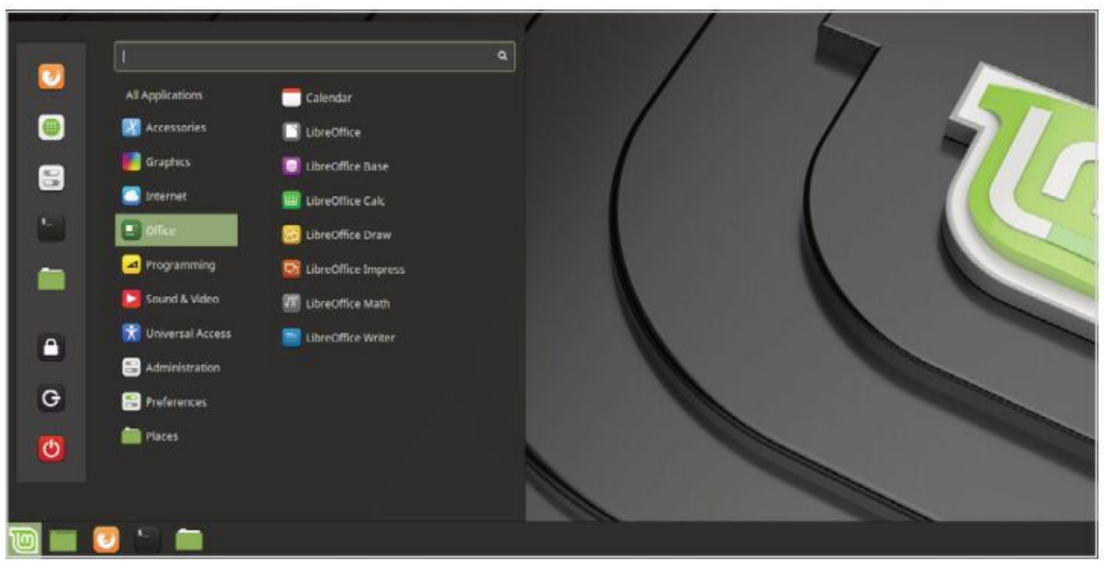
CINNAMON ON THE MENU

Now the Software Rending issue for Virtalbox users is out of the way, let's take a look at the Mint Menu and how it all works. Remember, this is just for Mint Cinnamon, other distros look and behave differently.

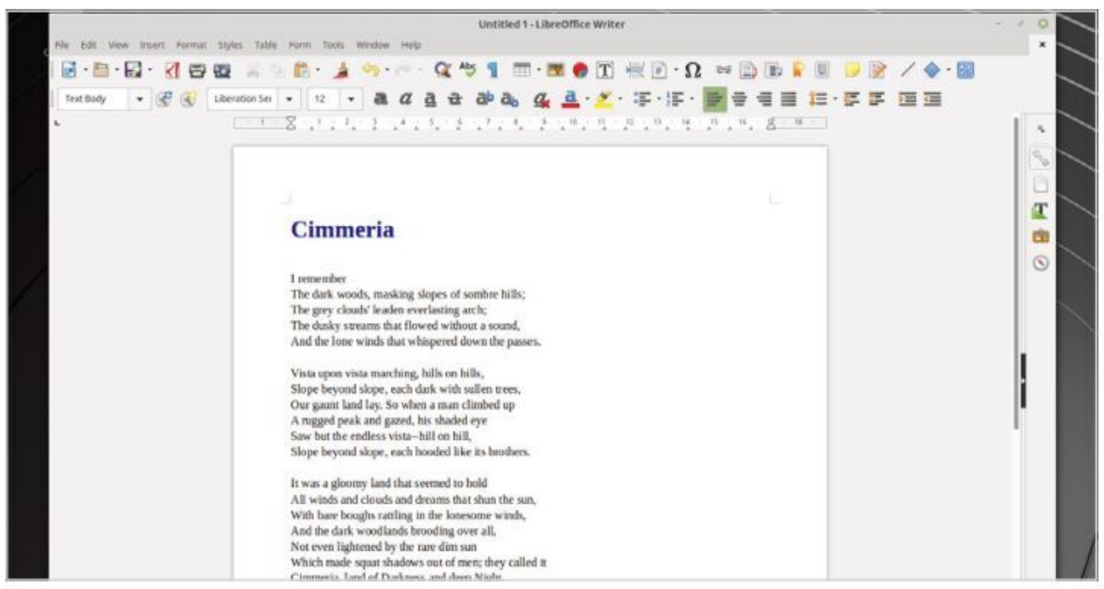
STEP 1 First off, you may have already noticed the Welcome Screen that pops up when you login to Linux Mint. Take a moment to browse through the options, read the First Steps option and so on. When you're done, click the X in the top right corner of the window to close it.



STEP 2 You've already used the Mint Menu to reboot the system and when you first used the Live Environment. This time, click the Menu button and hover over the Office entry in the middle column. This changes the icons represented in the right-hand column, detailing what apps are installed under that section.

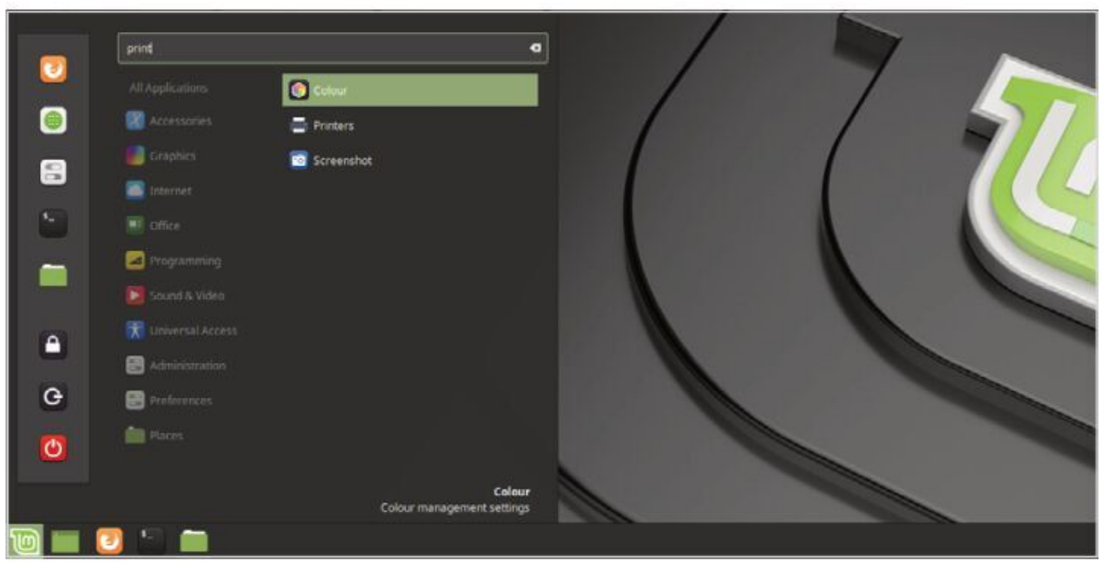


STEP 3 Launching any of the apps from the Menu is as simple as finding one and clicking it. For example, under the Office section, click on LibreOffice Writer. Writer is the preinstalled word processor for Linux Mint. It opens and saves as Microsoft Word and functions in almost the same way.

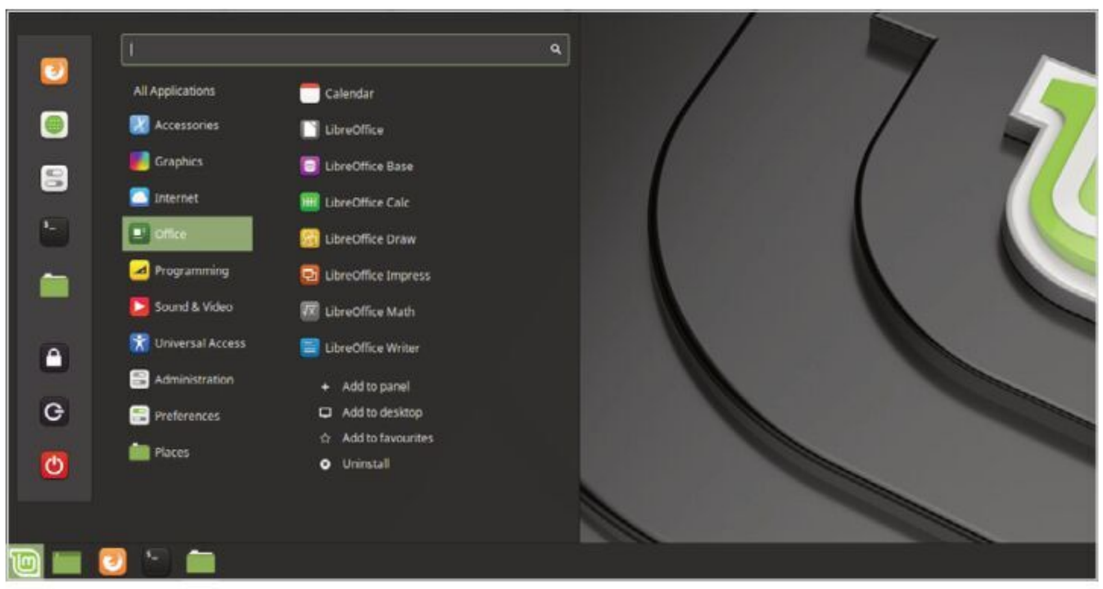


STEP 4 Open up the Menu again but this time hover the mouse pointer over Graphics, then click on GNU Image Manipulation Program. GIMP is a powerful image manipulation app that's probably as effective as Adobe's Photoshop but requires a little more work to get the results you want. It's certainly worth taking the time to master, though.

STEP 5 If you're looking for a particular function or app, such as setting up a printer, click on the Search box at the top of the Menu box. Start typing the app or function you want, such as printers, and the Mint Menu displays the relevant options below. This works with most modern Linux menus, regardless of the distro.



STEP 6 Hover over any of the apps listed on the right-hand column and right-click, to be presented with a list of options: Add to Panel, Add to Desktop, Add to Favourites and Uninstall. Most of these options are obvious in their use. Add to Favourites though places the app in the left-hand, quick-access column.



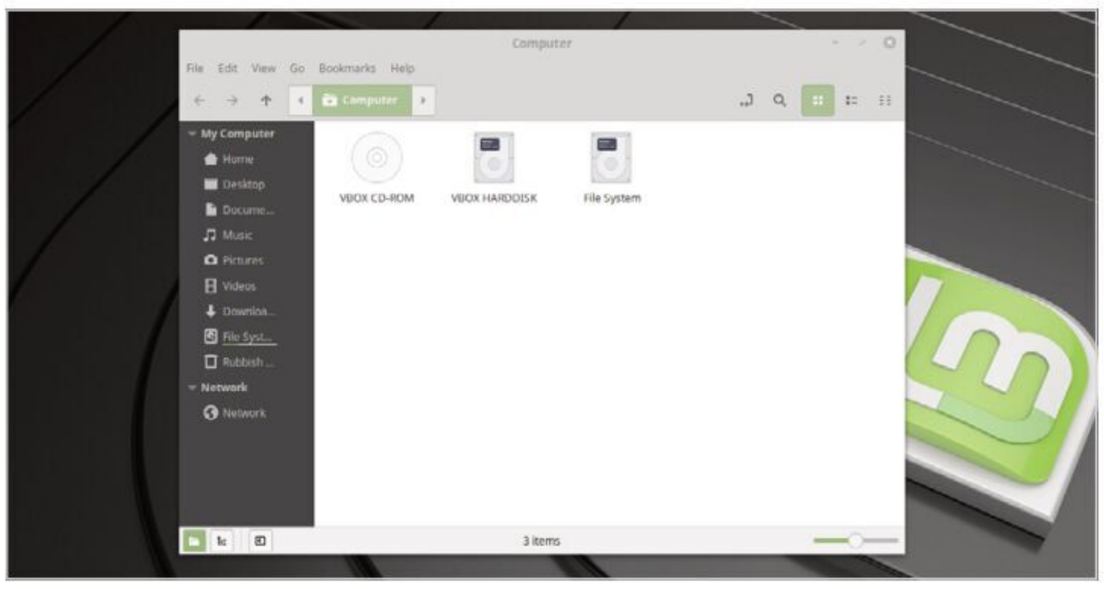
Navigating the Cinnamon Desktop

Each desktop environment behaves differently to that of the next. Some DEs offer widgets that can be customised and placed on the desktop, others instead opt for a clean, sharp look to keep everything running as fast as possible. Let's see what the Cinnamon desktop has to offer.

A TOUCH OF SPICE

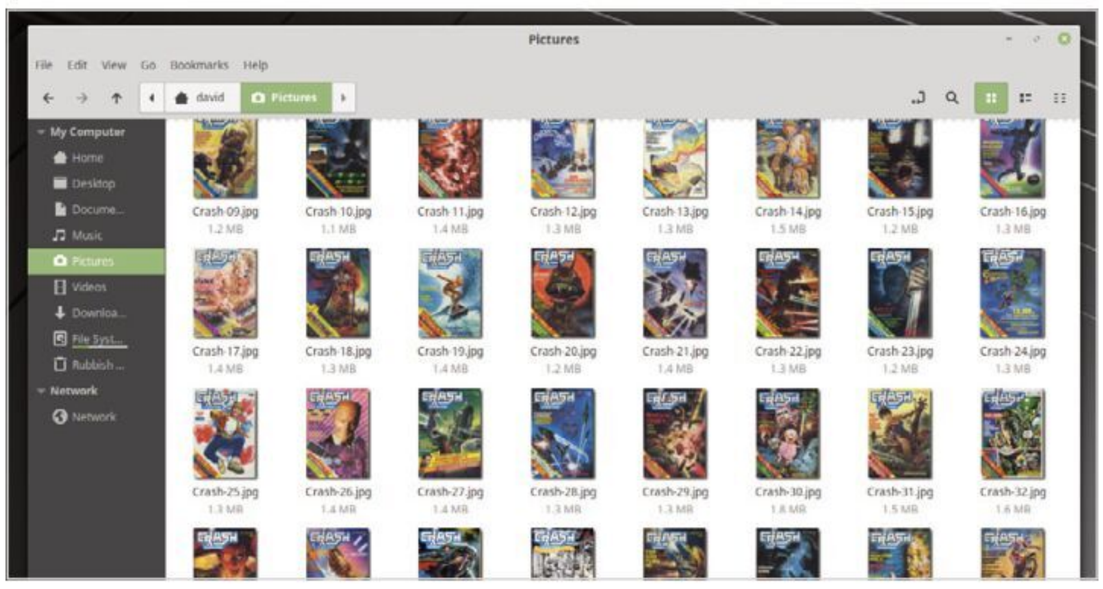
The Cinnamon desktop environment is a great blend of style and performance. There's lots to like about it, which is why it's such a popular DE.

STEP 1 Begin exploring the Cinnamon desktop by double-clicking the Computer icon. This brings up Nemo, the file manager used in Cinnamon. The Computer icon opens up the root level file system, with access and views to the optical drive (if you have one installed), hard drive and core Linux file system.

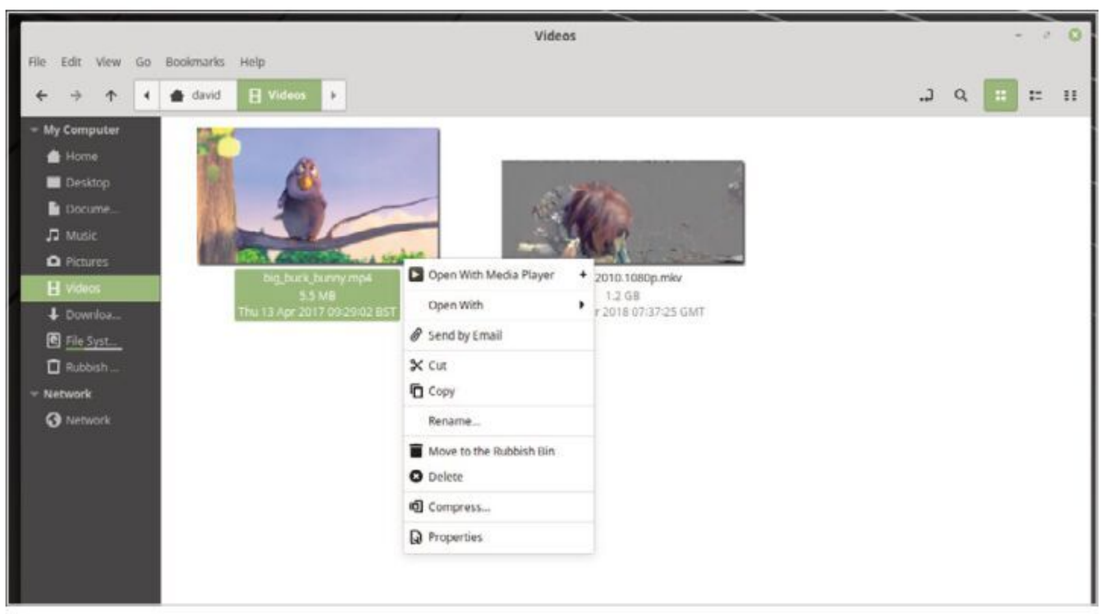


STEP 2 The Linux file system can appear confusing to a former Windows user, so until you're a little more knowledgeable on how it all works, we'd recommend you concentrate on the Home icon on the desktop instead. In here is everything relating to your user account: where you store Pictures, Videos, Music, Documents, Downloaded items etc.

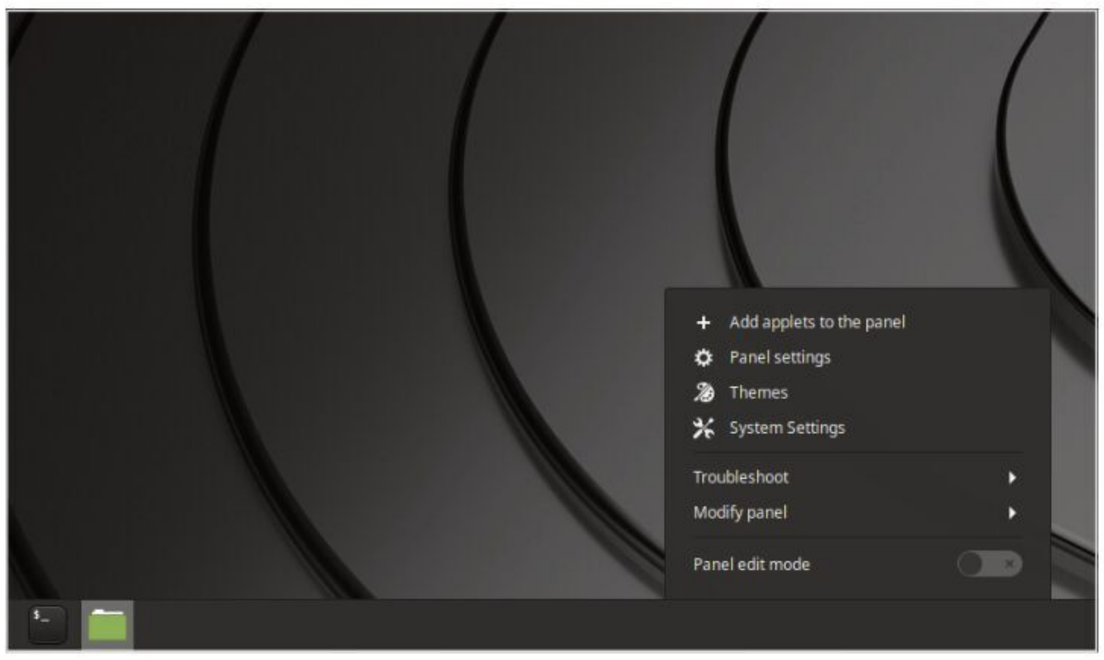
STEP 3 Nemo has many different features, views and ways so that you can view and manipulate files and folders. For example, if you have any images in the Pictures folder, you can select the icon zoom level for the images by using the slider located in the bottom right of Nemo, labelled Adjust Zoom Level.



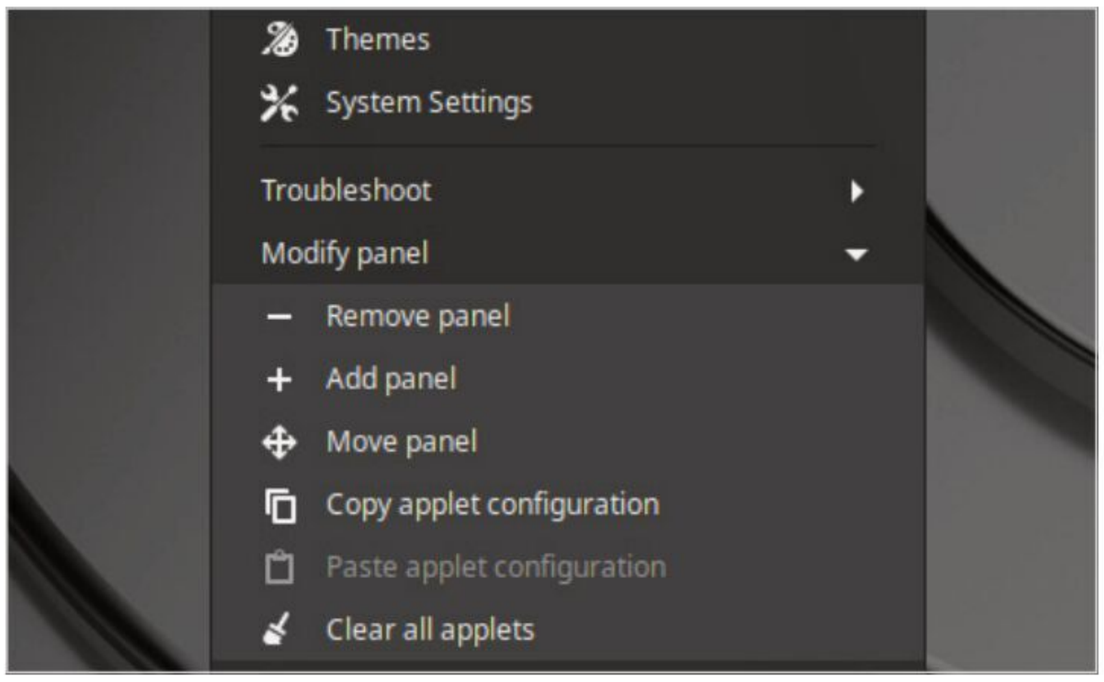
STEP 4 Just like any good file manager, if you right-click any of the files or folders within you get a wealth of options. In the case of Cinnamon, the defaults allow you to play or view a file depending on what type of file it is and copy, cut, delete, compress, rename, send via email and view its properties.



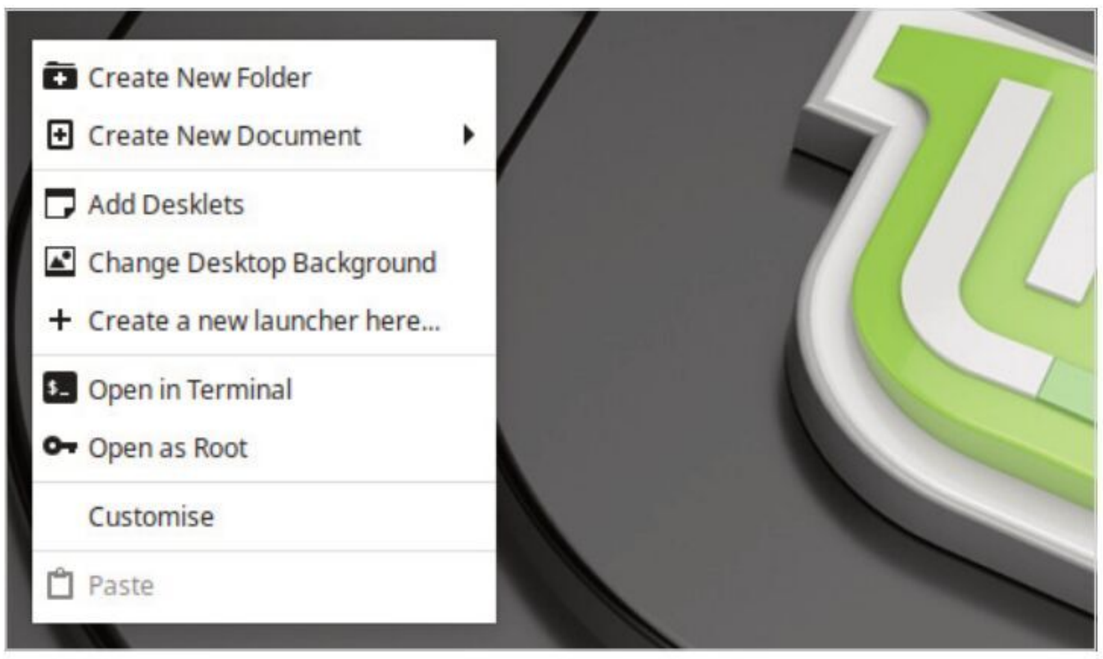
STEP 5 At the bottom of the desktop there's the Panel. We've already looked at one section of the Panel, the Menu. If you right-click anywhere on the Panel, other than on a Panel app, you see a menu allowing you to edit, add and set up the Panel in a different way.



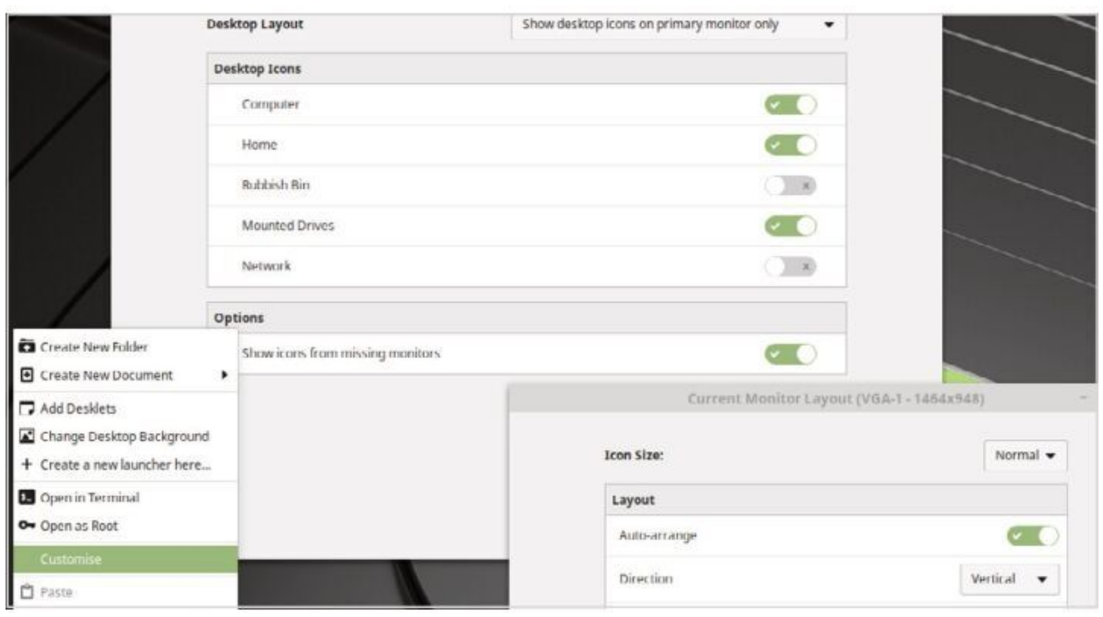
STEP 6 For example, if you click the option Modify Panel, you can remove, move, remove the Panel, add a new one and clear it of any Applets that are currently present. An Applet, by the way, is an app that's designed to work and fit into the Cinnamon Panel.



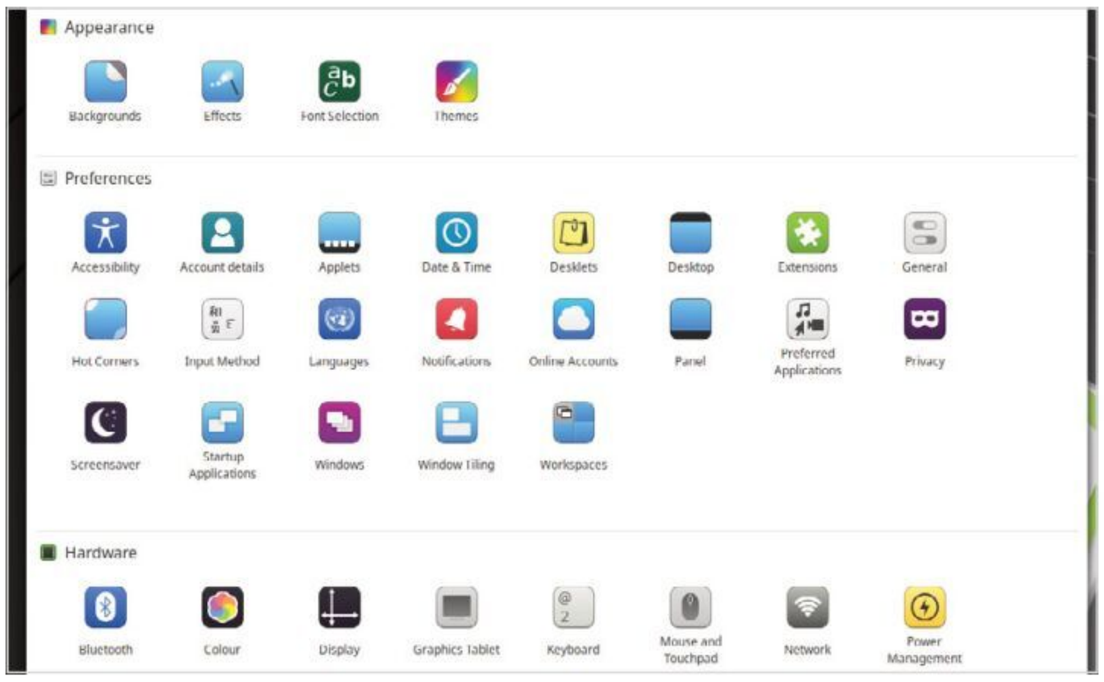
STEP 7 If you right-click anywhere on the Cinnamon desktop you see a set of options that allows you to further add to, edit or view the desktop content differently. It's very similar to that of Windows, which is why Mint is a good choice for ex-Windows users.



STEP 8 From that desktop right-click, context menu, select Customise, followed by the Desktop Settings link at the bottom of the newly opened window. This opens a new window where you're able to edit which desktop icons are present. If you're using a setup with multiple monitors attached, you can also choose which monitor displays which icons.



STEP 9 Click on the Menu and type system settings and open the resulting icon. This takes you to the System Settings options. From here you're able to control and edit the way Linux Mint Cinnamon looks and works as well add new users, manage the firewall and enable accessibility options.



STEP 10 In short, Linux Mint Cinnamon can be configured to look quite extraordinary. There are many examples available of how good it can get and what can be achieved. You can go as complex or as simple as you want, adding different component and animations or just keeping it plain and easy to read.

10 Things to do After Installing Linux Mint

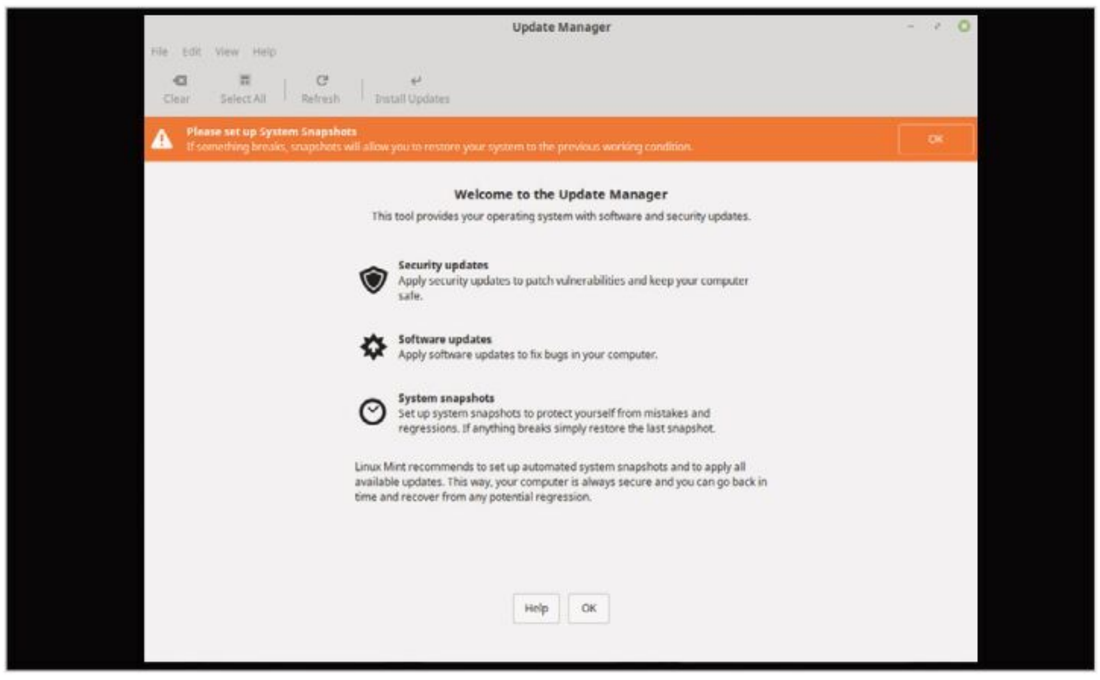
Linux Mint is a polished distro out of the box but, as with most Linux distros, there are some tweaks that can be applied to improve the way it works. Although these are Mint-specific tweaks, most can be applied to other distros.

LINUX TWEAKS

Some of these post-installation actions are highly recommended, while others are just handy additions and simply tweak the system or add a customisation.

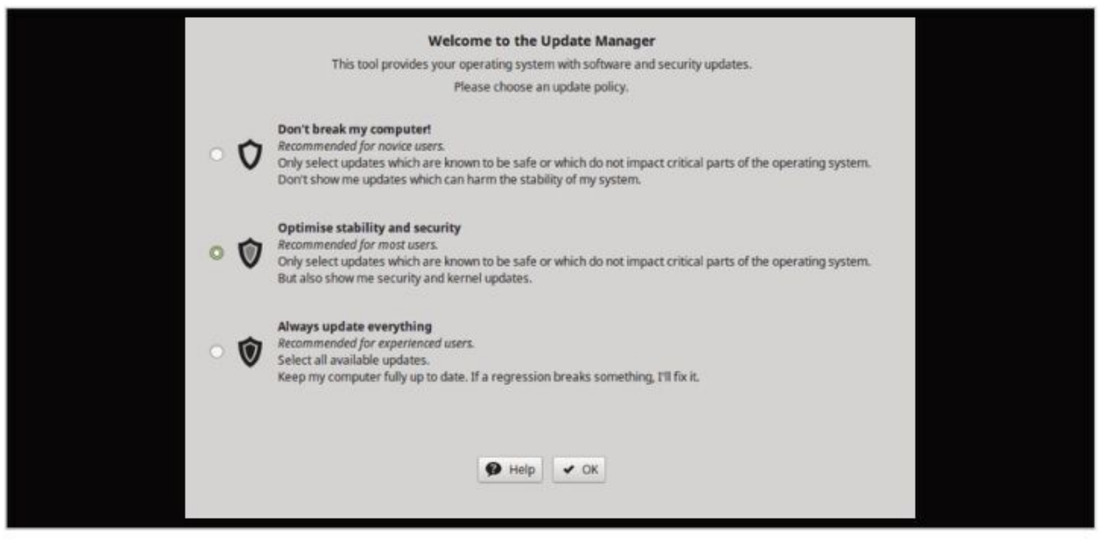
STEP 1

The first, and most important, post-installation action is to update the system. Click on the shield icon in the Panel, found at the bottom right of the desktop next to the time and date. This launches the Update Manager.



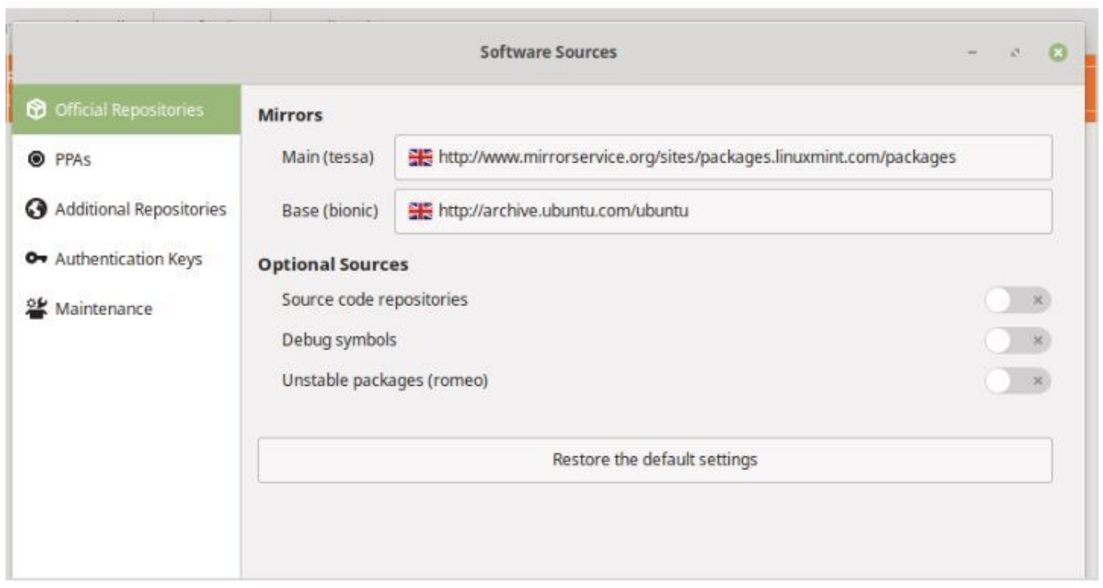
STEP 2

Linux Mint offers the user a three level policy approach to updates: Don't Break My Computer, Optimise Stability and Security and Always Update Everything. The recommended option is the Optimise Stability and Security, which only updates safe, essential patches that won't impact critical elements of the core OS. Read through the descriptions but choose the middle, and recommended option.



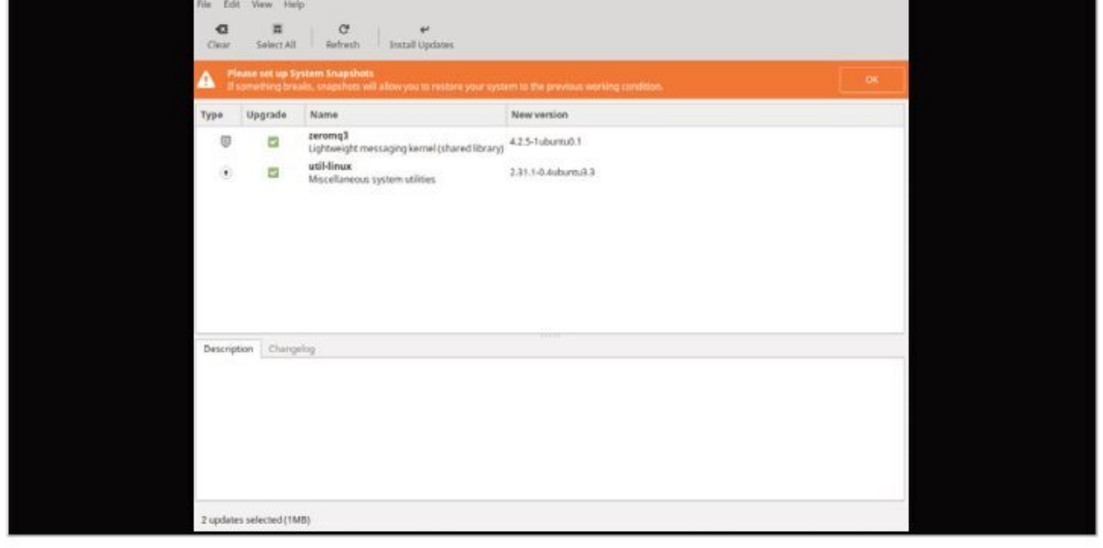
STEP 3

Click the OK button and you can see a couple of updates ready for installation. Before you update though, click on the blue bar OK button to switch to a Local Mirror. This opens the Software Sources option. In the Mirrors section, click on the Main and Base drop-down menus and select a server closest to your current location.

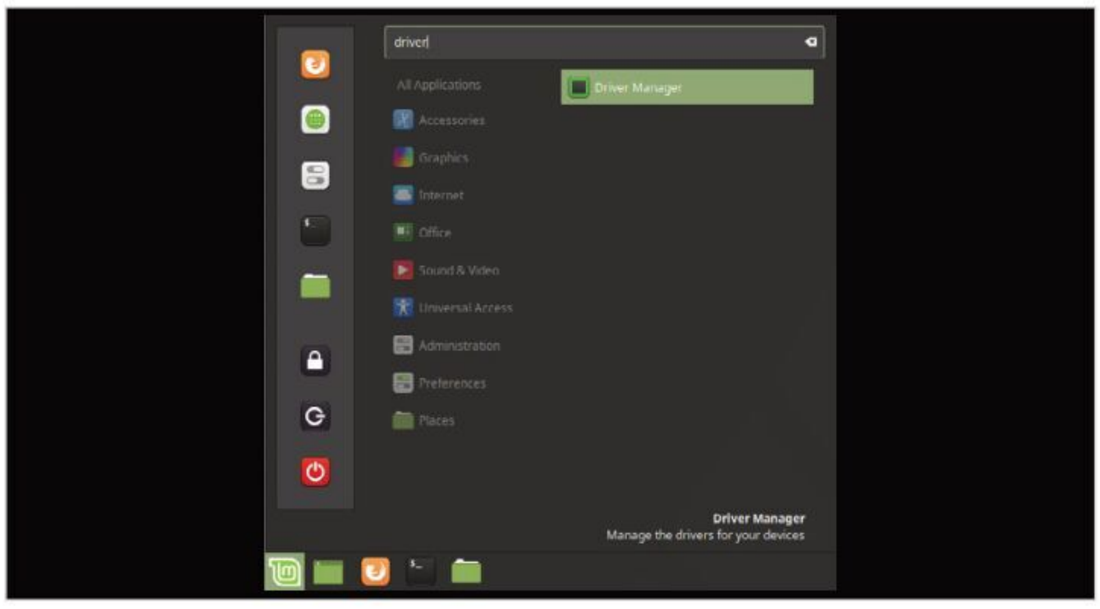


STEP 4

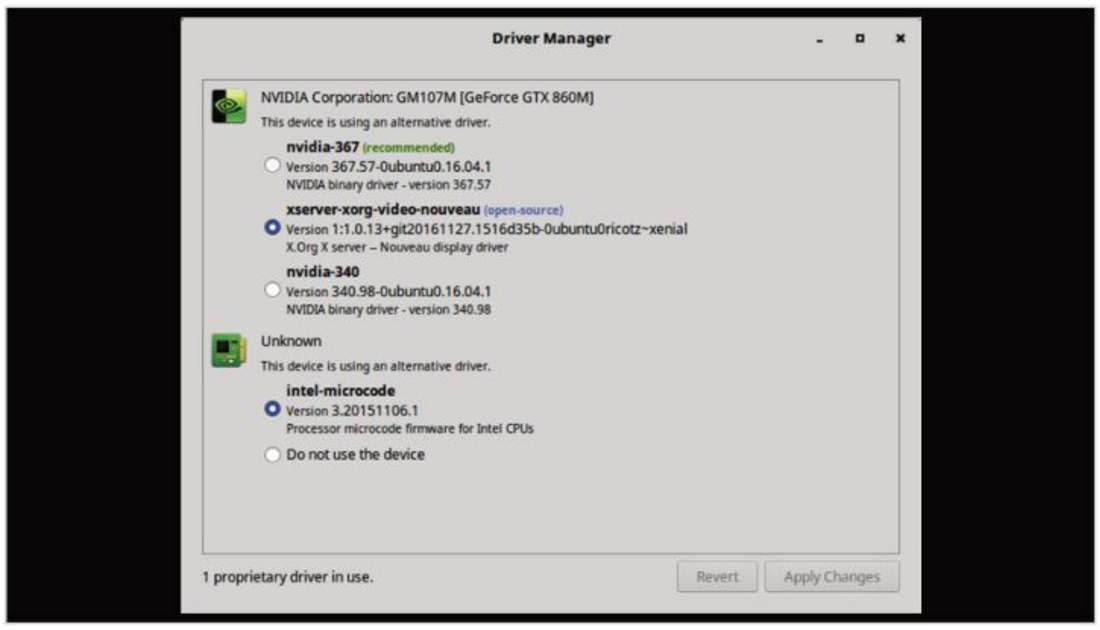
Click the Update the Cache button and close the Software Sources window. Back in the Update Manager, click on the Install Updates icon and enter your password. The updates automatically apply themselves and relaunch Update Manager, this time with a lot more updates. Again, click Install Updates, OK any messages and wait for them to finish.



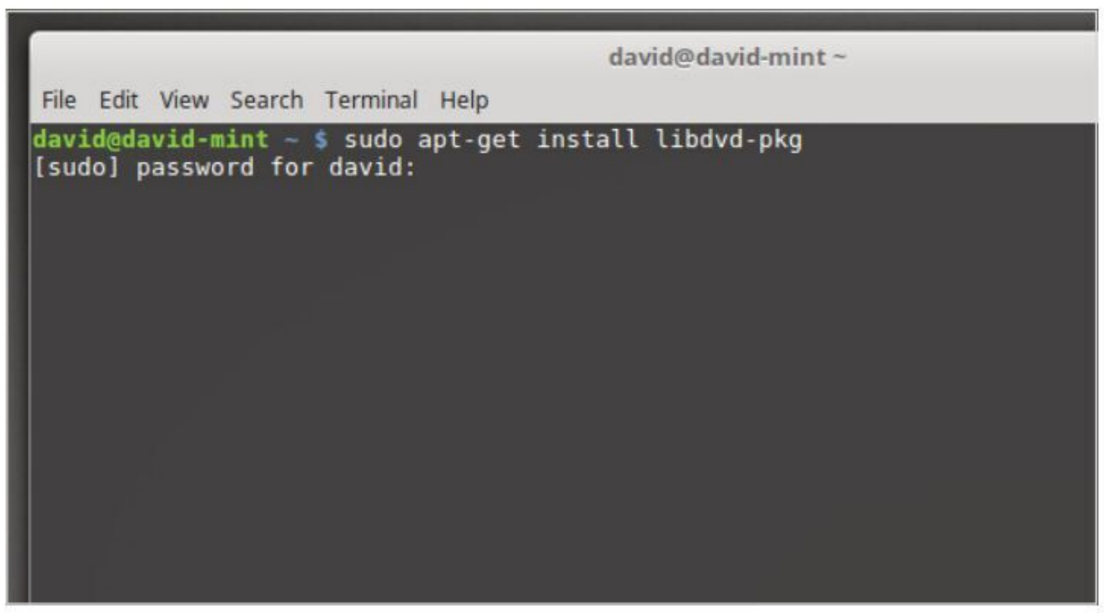
STEP 5 The updates are graded by level, 1 being a low level update, level 5 being a dangerous one. Stick to level 3 updates, is our advice; and click Replace for any messages regarding overwriting a configuration file. With regards to the Software Rendering issue and lack of drivers for non-VirtualBox users, click the Menu and type 'driver' into the search box.



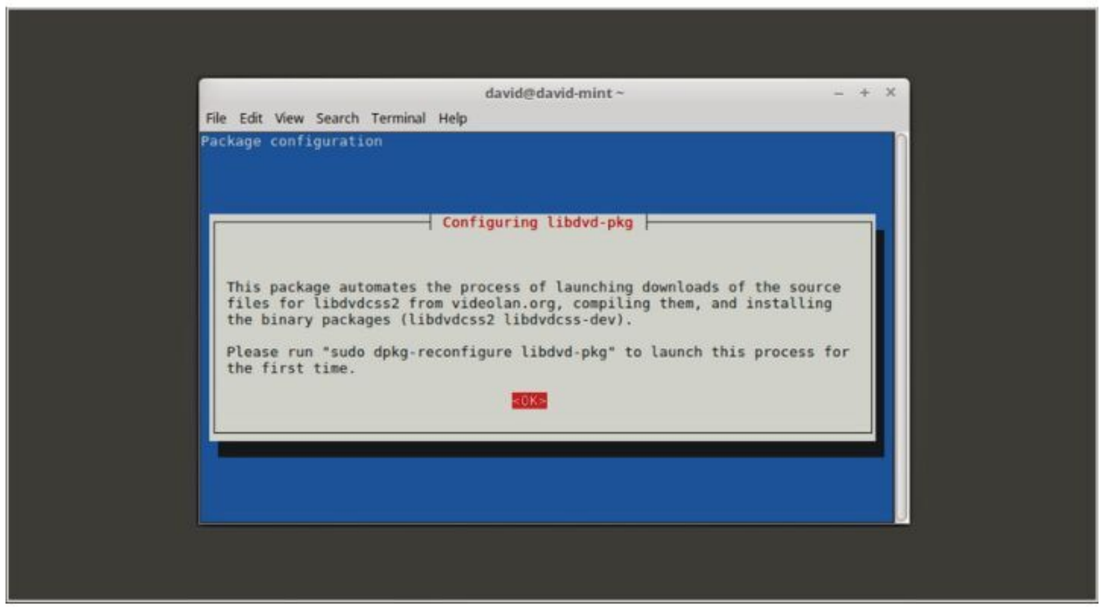
STEP 6 Click the Driver Manager app that appears as a result of the search and enter your password. Mint takes a moment to analyse what's available and presents you with a selection of potential drivers based on your detected hardware. Those with graphical problems, such as Software Rendering, should opt to use the latest, recommended Graphics driver.



STEP 7 At this point you'll probably need to restart Linux Mint, so do that now. After a reboot click the Menu button again, followed by the Terminal. The Terminal icon is found in the left-hand column, above the Files icon. With the Terminal open, enter: `sudo apt-get install libdvd-pkg`, press Enter and type in your password. This enables encrypted DVD playback.



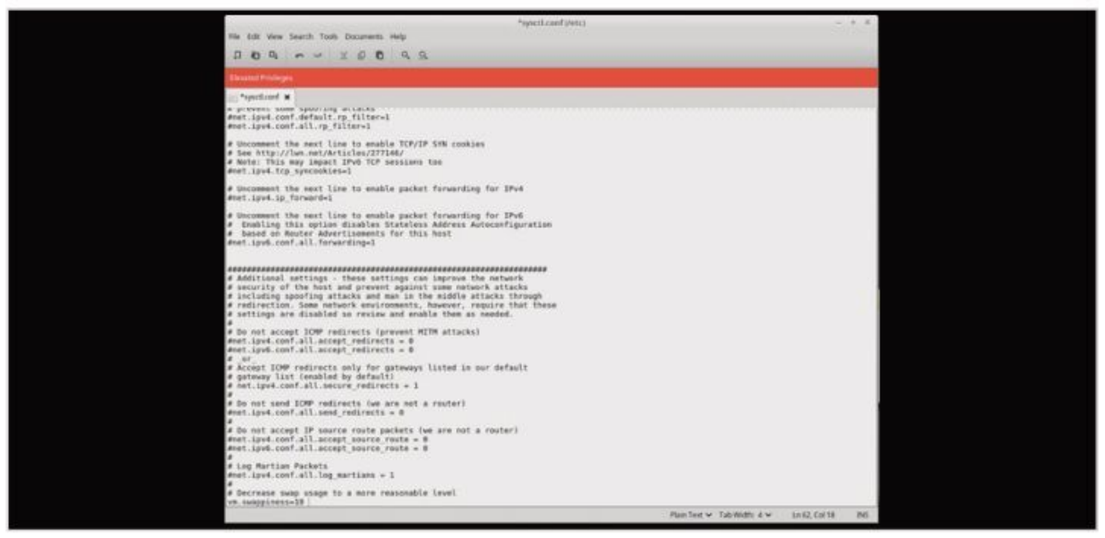
STEP 8 When asked to accept the changes, enter Y and also when asked to configure libdvd. Make sure OK is highlighted and press Enter, then Yes to any further questions. Next up, still in the Terminal, enter: `cat /proc/sys/vm/swappiness`; the result should be 60. If your computer has less than 4GB of RAM/memory, then enter: `gksudo xed /etc/sysctl.conf`.



STEP 9 This tweak helps speed up systems with less than 4GB RAM/memory. Scroll down to the bottom of the file you just opened and add the following new lines:

```
# Decrease swap usage to a more reasonable level
vm.swappiness=10
```

Click File > Save, then File > Quit. Reboot Linux Mint and you should notice a slight hike in performance.



STEP 10 Security is always a concern in this modern digital age. While Linux Mint is a secure system, it's advisable to always try and improve it. Click the Menu button and search for Firewall; click the Firewall Configuration icon and enter your password. In the Firewall window, click the Status slider to On.

Apollo 11

DID YOU KNOW... July 20th 1969 is remembered as a turning point in human history, when mankind first stepped on the moon. The software that helped the crew make that pivotal voyage didn't even exist at the time and had to be developed from scratch. Using a new method of storing programs called Rope Memory, the engineers at MIT Instrumentation Laboratory populated the memory with a special version of assembly.

Incredibly, that code is freely available to view on GitHub. Uploaded by a researcher in 2003, you can view the code by visiting **www.github.com/chrislgarry/Apollo-11/**. Look out for code snippets, such as Burn_Baby_Burn, in particular line 925 with the comment "Goodbye. Come again soon."

Have a look through, and see what else you can find.

```
                TS      TEMPR60         # FOR GENERALIZED RETURN TO OTHER BANKS.
P40A/P          TC      BANKCALL        # SUBROUTINE TO CHECK PGNCS CONTROL
                CADR    G+N,AUTO        # AND AUTO STABILIZATION MODES
                CCS     A               # +0 INDICATES IN PGNCS, IN AUTO
                TCF     TURNITON        # + INDICATES NOT IN PGNCS AND/OR AUTO
                CAF     APSFLBIT        # ARE WE ON THE DESCENT STAGE?
                MASK    FLGWRD10
                CCS     A
                TCF     GOBACK          # RETURN
                CAF     BIT5            # YES, CHECK FOR AUTO-THROTTLE MODE
                EXTEND
                RAND    CHAN30
                EXTEND
                BZF     GOBACK          # IN AUTO-THROTTLE MODE -- RETURN
TURNITON        CAF     P40A/PMD        # DISPLAYS V50N25 R1=203 PLEASE PERFORM
                TC      BANKCALL        # CHECKLIST 203 TURN ON PGNCS ETC.
                CADR    GOPERF1
                TCF     GOTOP00H        # V34E TERMINATE
                TCF     P40A/P          # RECYCLE
GOBACK          CA      TEMPR60
                TC      DANKJUMP        # GOODBYE.  COME AGAIN SOON.

P40A/PMD        OCT     00203

# Page 749
                BANK    36
                SETLOC  P40S
                BANK

                COUNT*  $$/P40

#       ********************************
#       CONSTANTS FOR THE IGNITION ROUTINE
#       ********************************

SERVCADR        =       P63TABLE +7

P40ADRES        ADRES   P40TABLE

P41ADRES        ADRES   P41TABLE -5

P42ADRES        ADRES   P42TABLE

                EBANK=  DVCNTR
DSP2CADR        2CADR   P63DISPS -2

                EBANK=  DVCNTR
```

Programmers aren't without a sense a humour. See what else is in there.

One giant leap for mankind, and coding.

Credit: Apollo 11/NASA

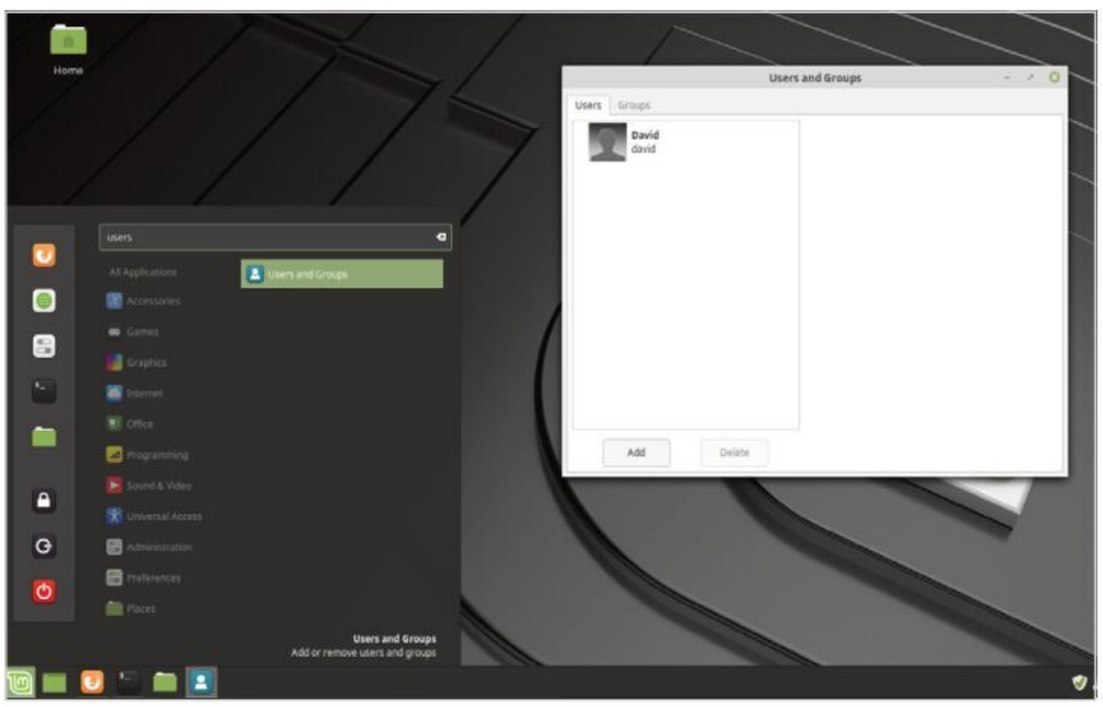
Creating Users

When you first install Linux Mint it is configured for use with a single user. While sharing a user account with the entire family is fine, there may come a time when you need to create separate users with their own unique Home folders.

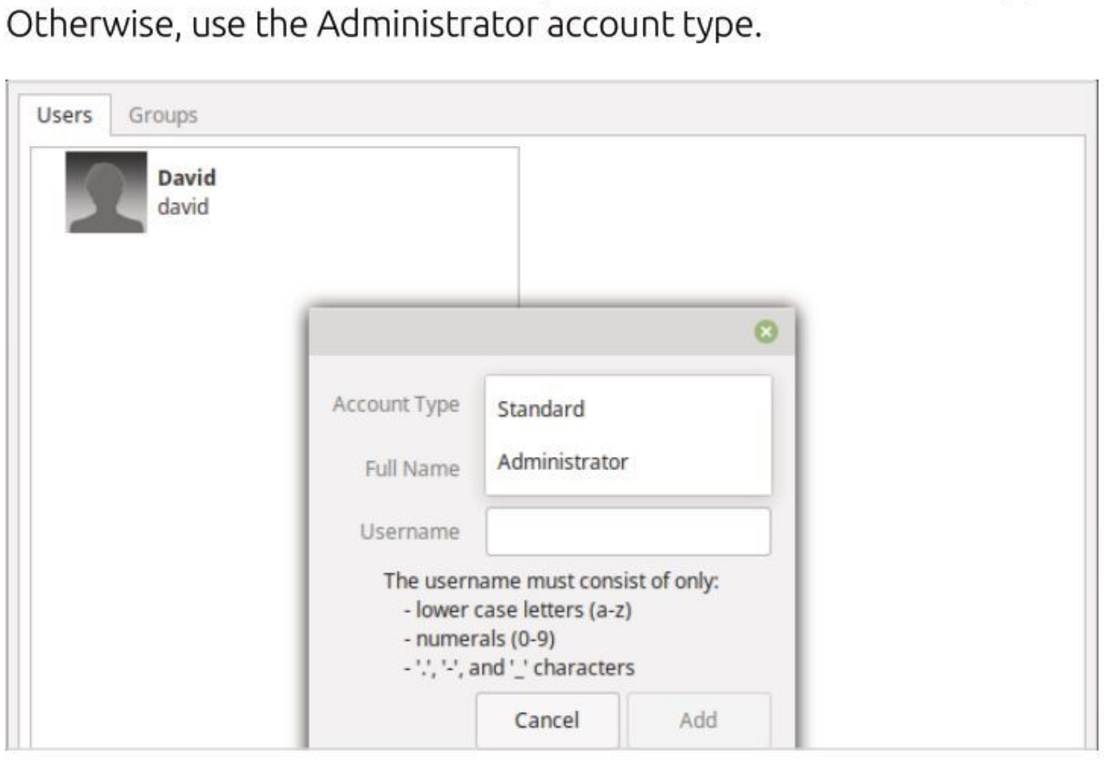
NEW USERS

Having different users means each user has access to his or hers own areas on the system. Documents, pictures, videos and so on are separate, as with multiple users on other operating systems.

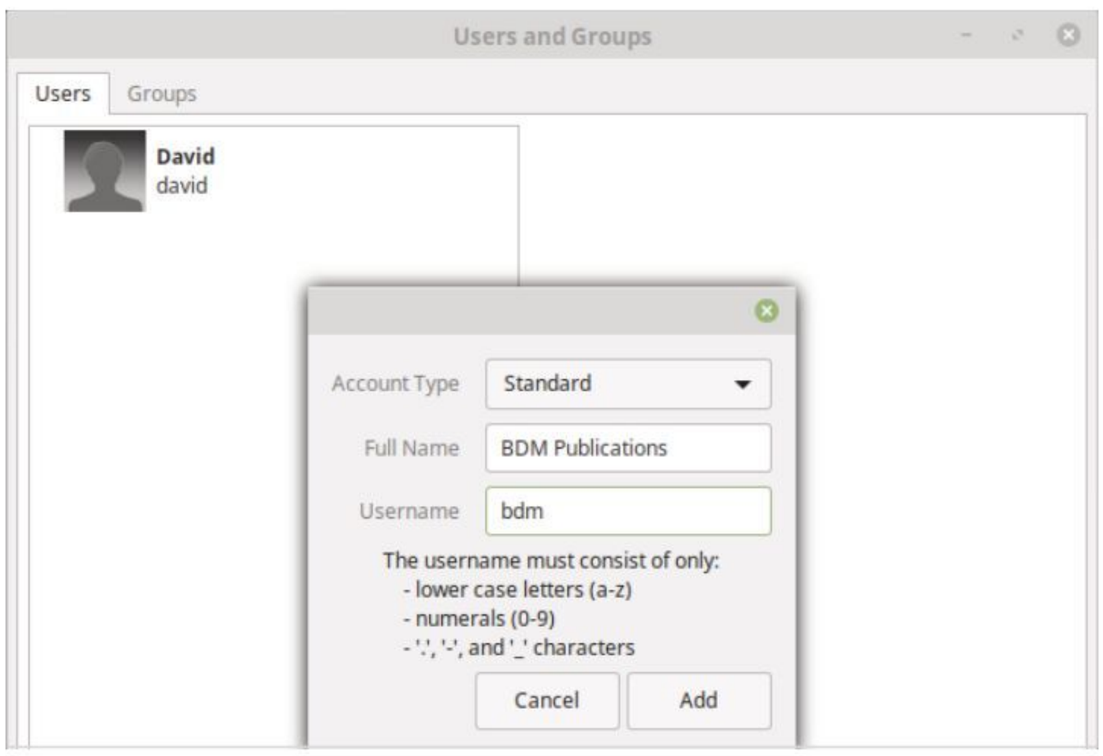
STEP 1 Click on the Linux Mint Menu and type 'users' to begin searching for the relevant console. From the search results, choose Users and Groups and enter your password. The Users and Groups console is quite basic looking, and thankfully easy to use. At first, you can just see your own username from when you installed Linux Mint.



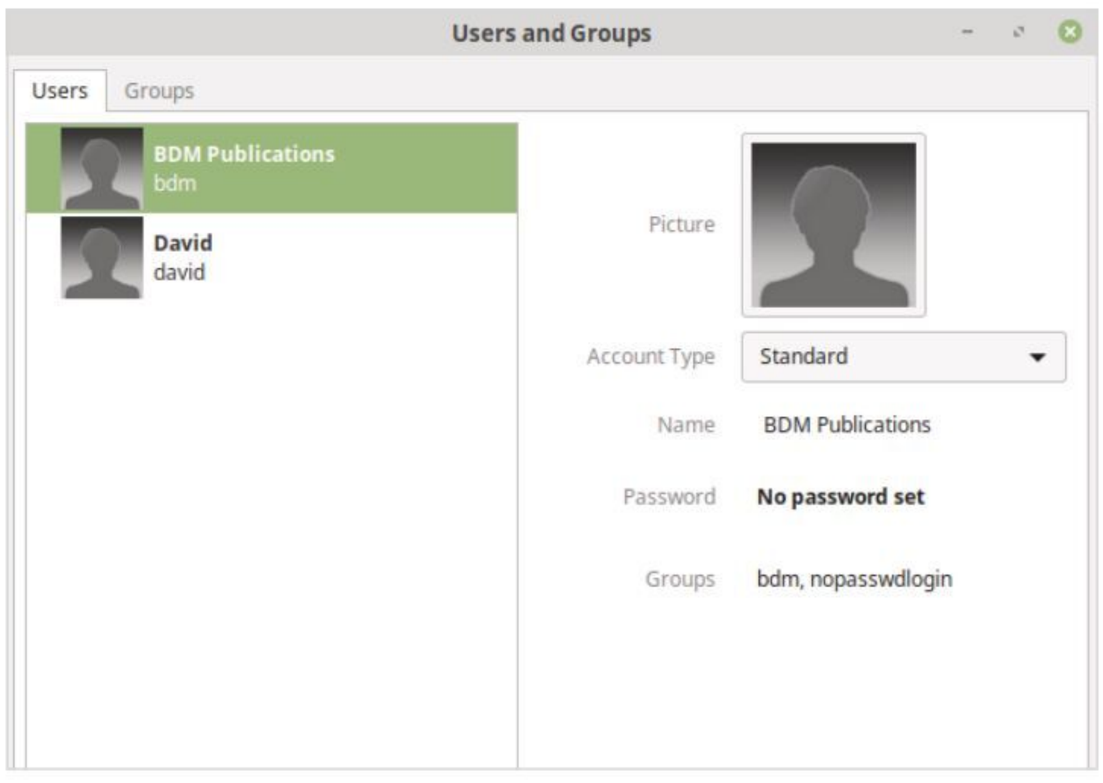
STEP 2 To add a new user, click the Add button at the bottom of the console. There are two types of user you can create, Standard and Administrator. Unless the new user has need to install new apps or access parts of the file system beyond their Home folder, then opt for the Standard account type. Otherwise, use the Administrator account type.



STEP 3 Enter the new user's Full Name, followed by the Username they need when logging into Linux Mint. Make sure the username is all in lower case, a-z and 0-9 characters only. You can have full stops, underscores or hyphens if you wish. Click the Add button when you're ready to continue.

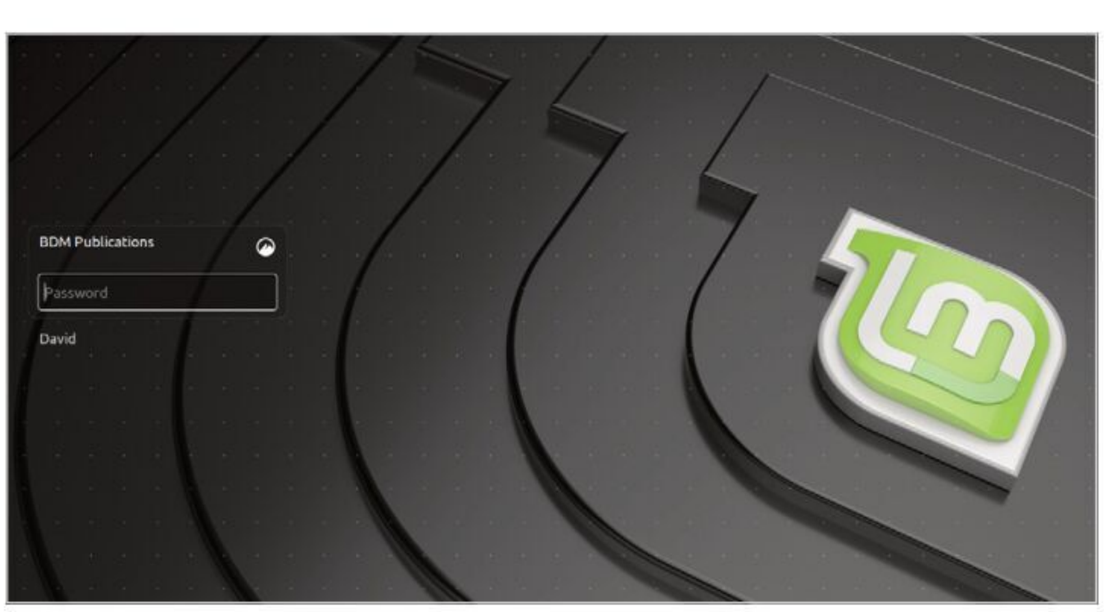


STEP 4 The new user appears in the list of current Linux Mint users, in alphabetical order. At present, there's no password set so click the user in the list of current users, then click the No Password Set option under the user's username.

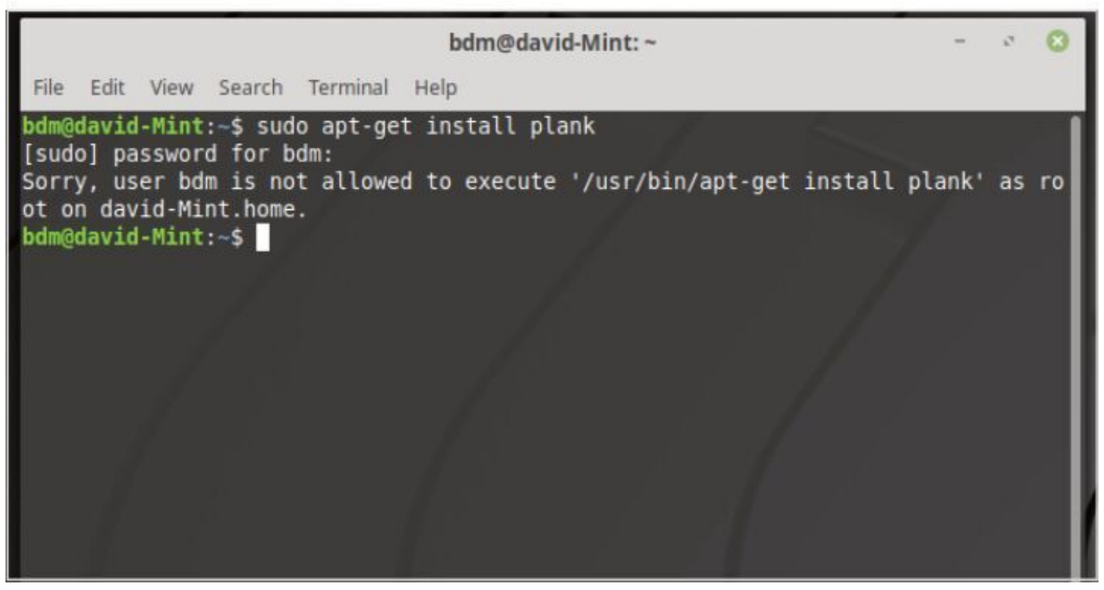


STEP 5 You can now enter a password for the new user or click the curled arrow at the end of the New Password text box to generate a password for you, as well as displaying it. Naturally, it's a good idea to come up with as strong a password as possible. When you're done, click the Change button.

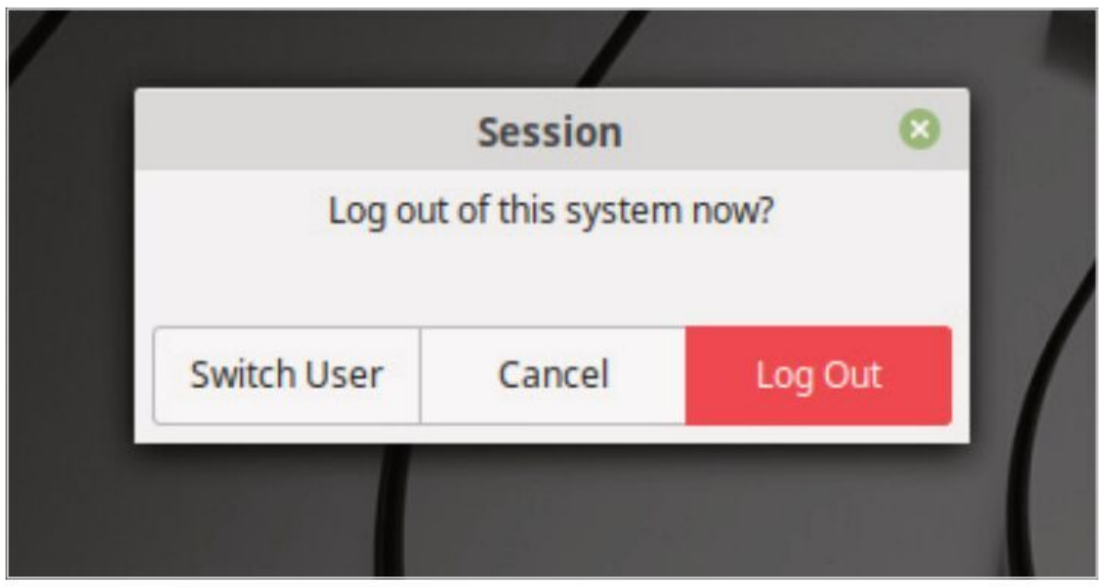
STEP 6 You can close the Users and Groups console window now, as the new user has been created. If you click the Mint Menu, followed by Logout, you are presented with the Mint Login Manager. The new user is now present in the list of currently available users. Click on him/her to log them in.



STEP 7 Once logged in the new user is required to set up their own desktop wallpaper, icons, Panel, Menu and so on. Depending on what Account Type you set up for them, Standard or Administrator, they won't be able to install any new apps. This screenshot is from a Standard user account type.



STEP 8 You can create as many new accounts as you need and you're able to switch between them when required. It's best to have just one account that's capable of installing new software, that way you can keep track of what's on your system.



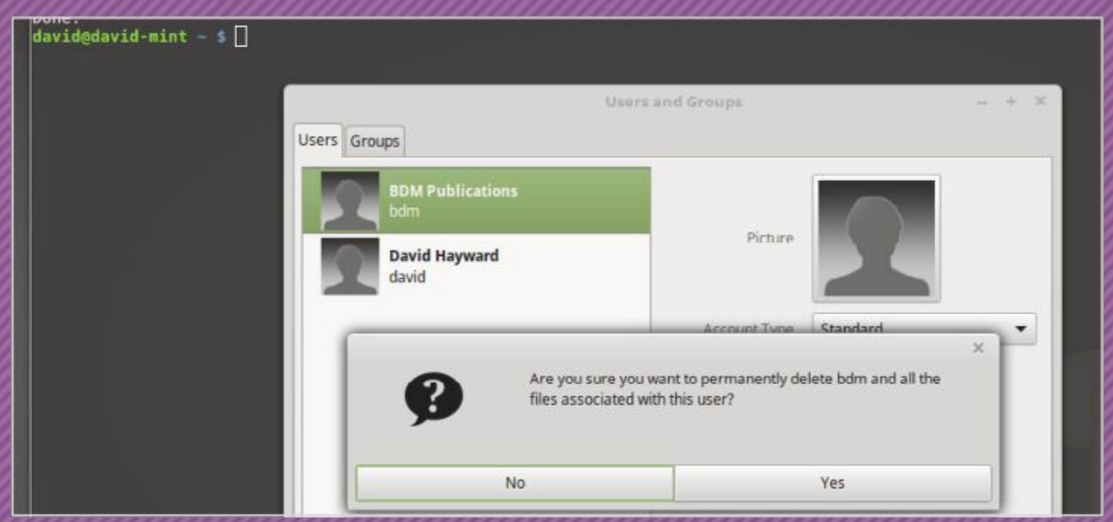
COMMAND LINE ACCOUNTS

Just as you'd expect, you can also create a new user within the command Line. Open up a Terminal session under the main (yours) Administrator account.

STEP 1 The process for adding a new user from the command line is relatively simple. To begin with, type: `sudo adduser <username>`, where `<username>` is the new user's login name. You're then asked to create a new password for the user, along with their full name and other details. Click y to confirm the details and create the user account.

```
david@david-mint ~ $ sudo adduser jim
Adding user `jim' ...
Adding new group `jim' (1002) ...
Adding new user `jim' (1002) with group `jim' ...
Creating home directory `/home/jim' ...
Copying files from `/etc/skel' ...
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Changing the user information for jim
Enter the new value, or press ENTER for the default
        Full Name []: Jim Gale
        Room Number []:
        Work Phone []:
        Home Phone []:
        Other []:
```

STEP 2 You can check the details and account type for the new user from within the Users and Groups console. If you want to delete a user from Mint, you can either enter: `sudo deluser <username>` in the Terminal or click the Delete button in Users and Groups.



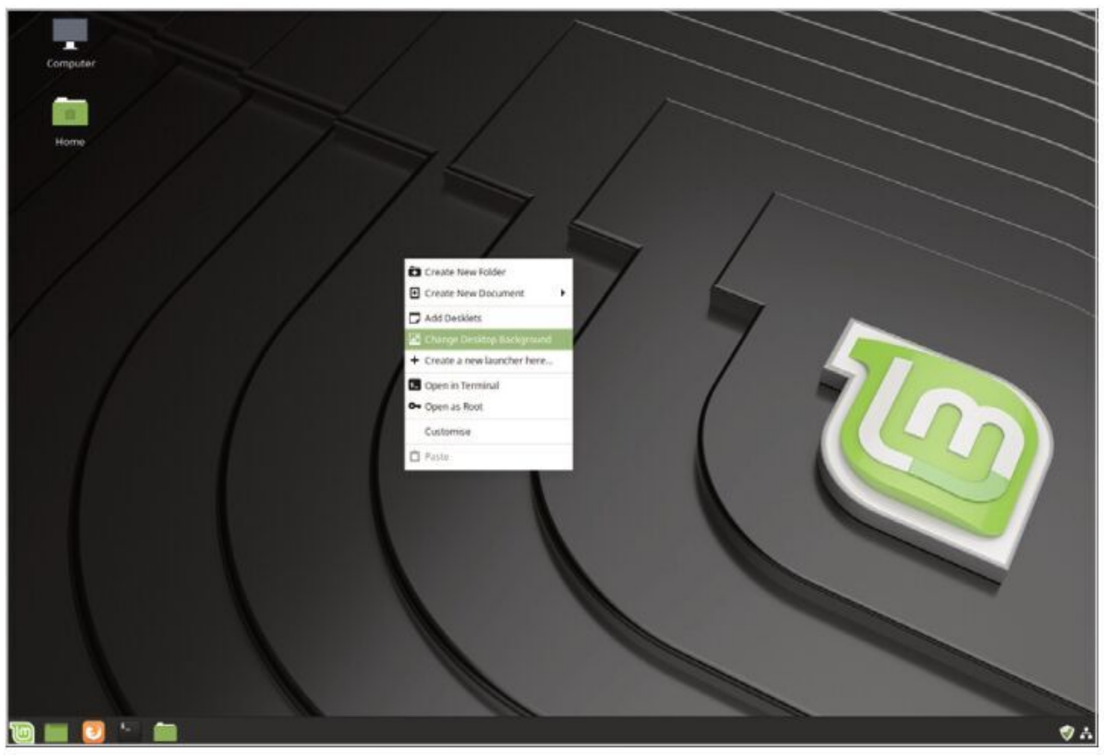
Customising the Desktop

Customising the operating system desktop is one way to make it your own: a personalised workspace that is there to help you work, inspire you or feature the company logo. Whatever your reasons for having your own desktop, here's how it's done.

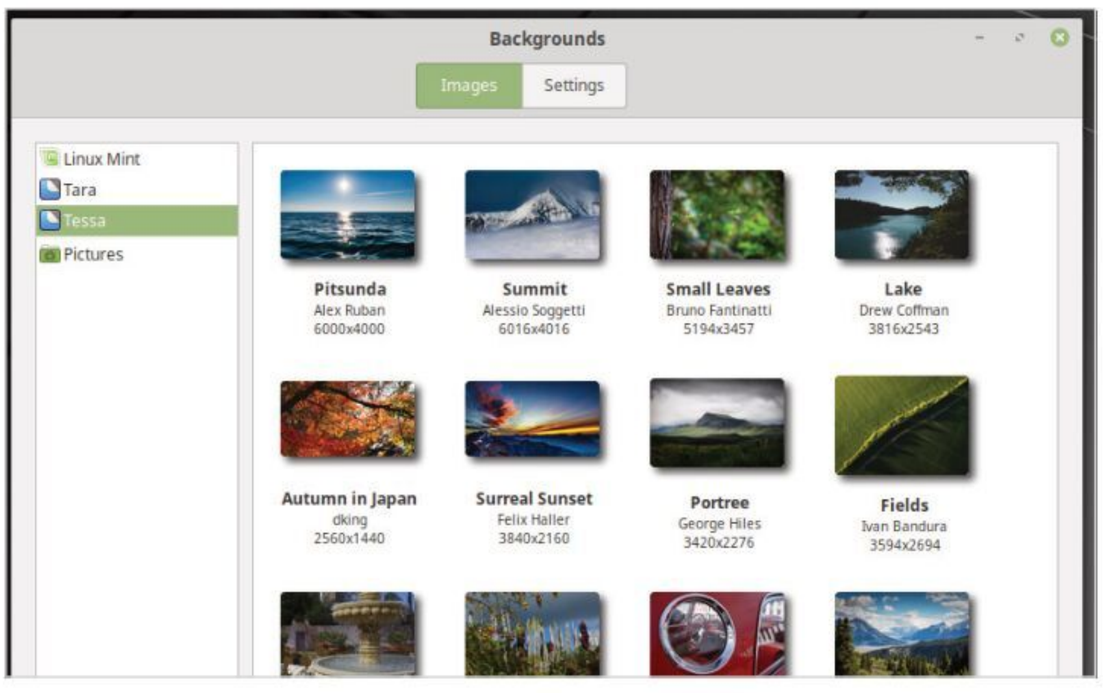
YOUR DESKTOP

Linux is probably one of the customisable operating systems there is. With just a few tweaks, one or two extras installed and some imagination, you can create something incredible.

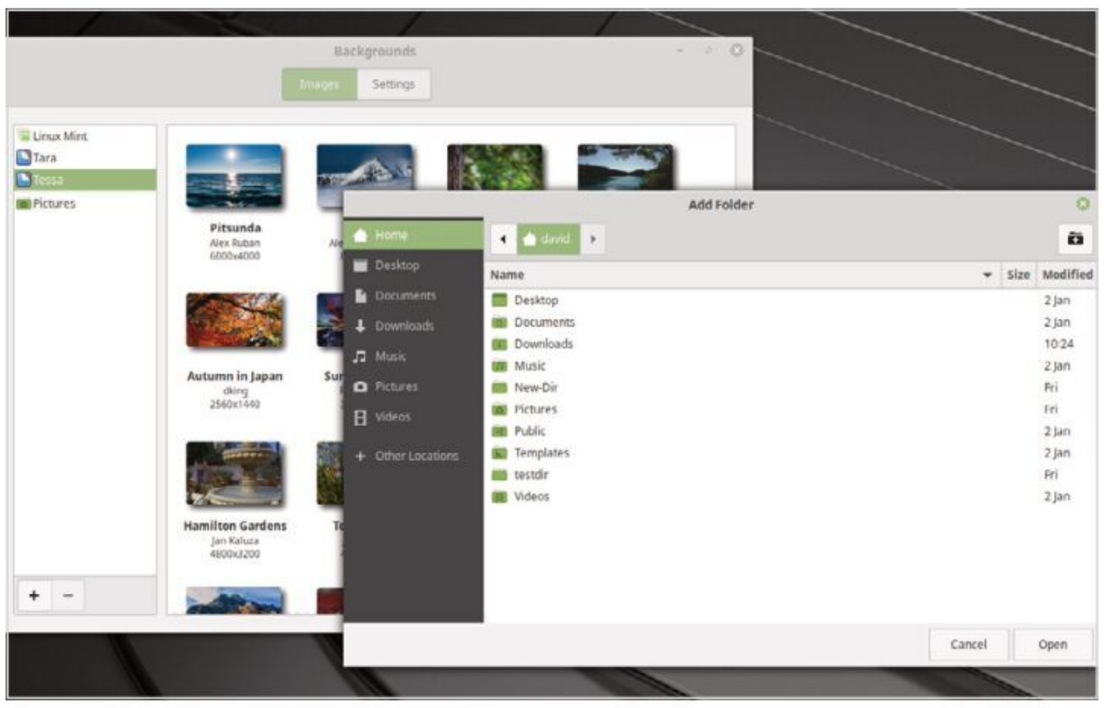
STEP 1 The first aspect of desktop customisation is to change the wallpaper. Right-click the desktop and choose Change Desktop Background. This opens the Backgrounds app in Linux Mint; remember, other distros may present their background, wallpaper selection tools differently.



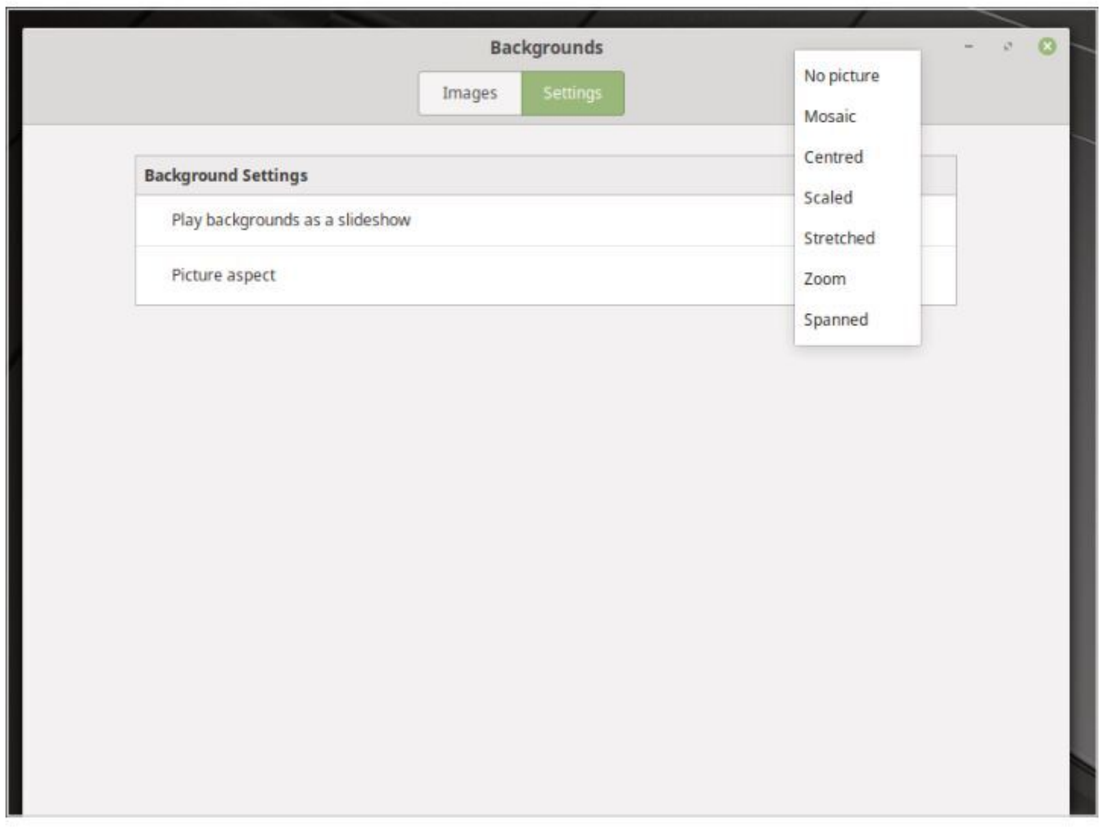
STEP 2 More recent versions of Linux Mint display available backgrounds depending on the version the user is running. You normally get three categories followed by a fourth, Pictures, which is separated from the others. The Pictures option is different because it reads the image content from the Pictures folder in your Home area.



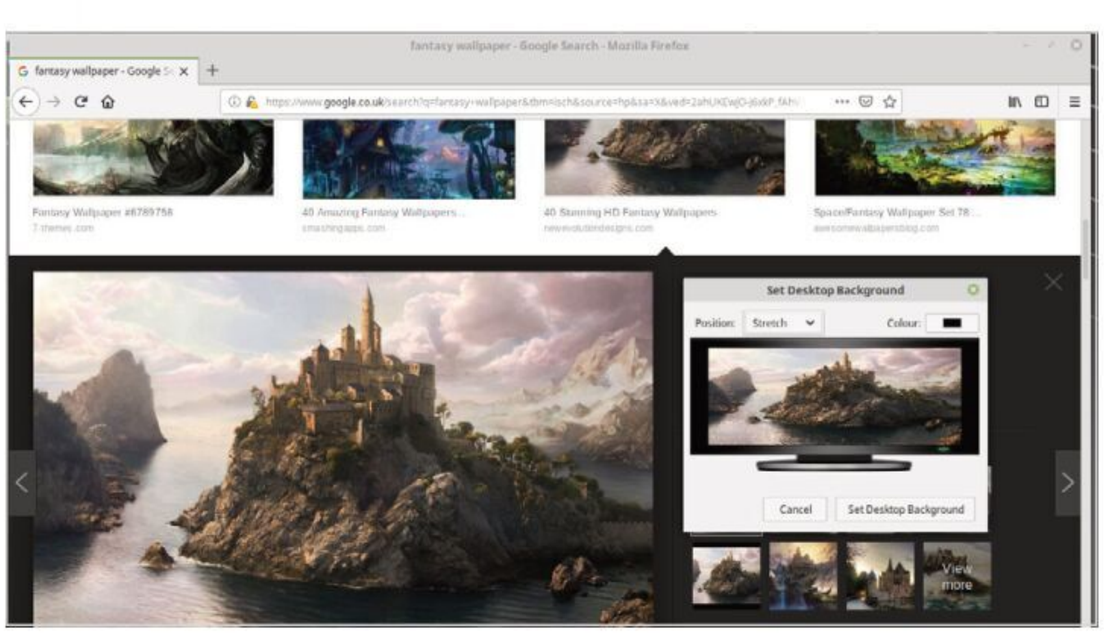
STEP 3 You just need to click the available images, from any of the locations provided to have them install as the desktop wallpaper. Incidentally, if you have images stored in another location on your system or network, you can add them by clicking on the Plus symbol at the bottom of the Backgrounds console, using the file manager to locate them.



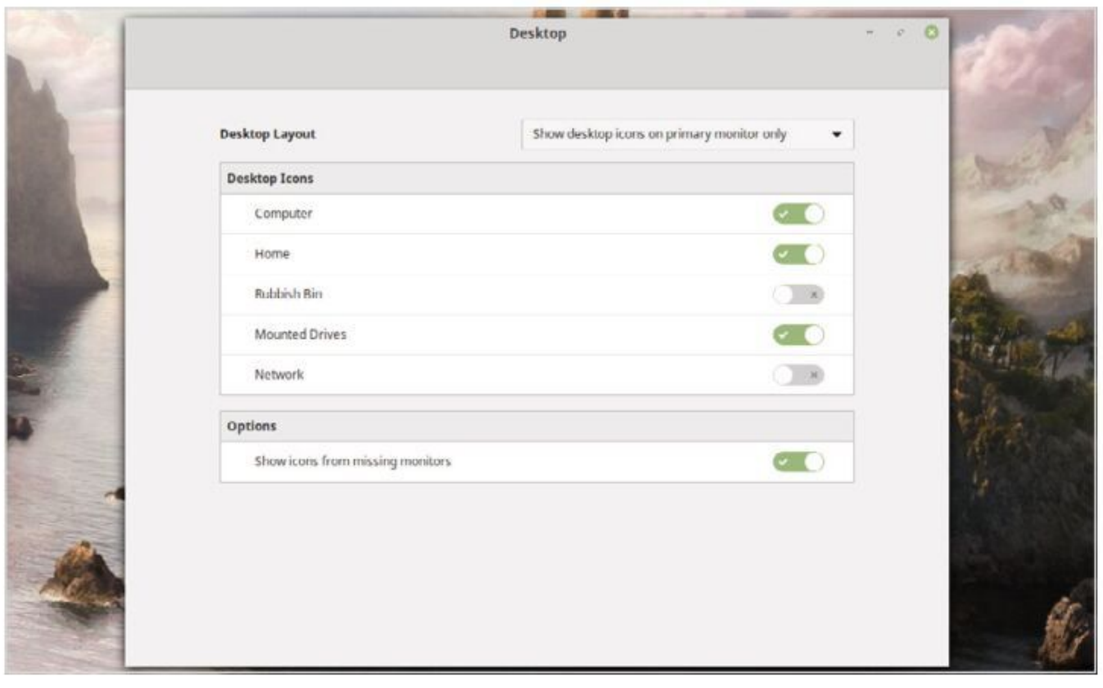
STEP 4 By clicking on the Settings tab you can, instead, play numerous images as a slideshow or change the aspect of the wallpapers to a variety of choices.



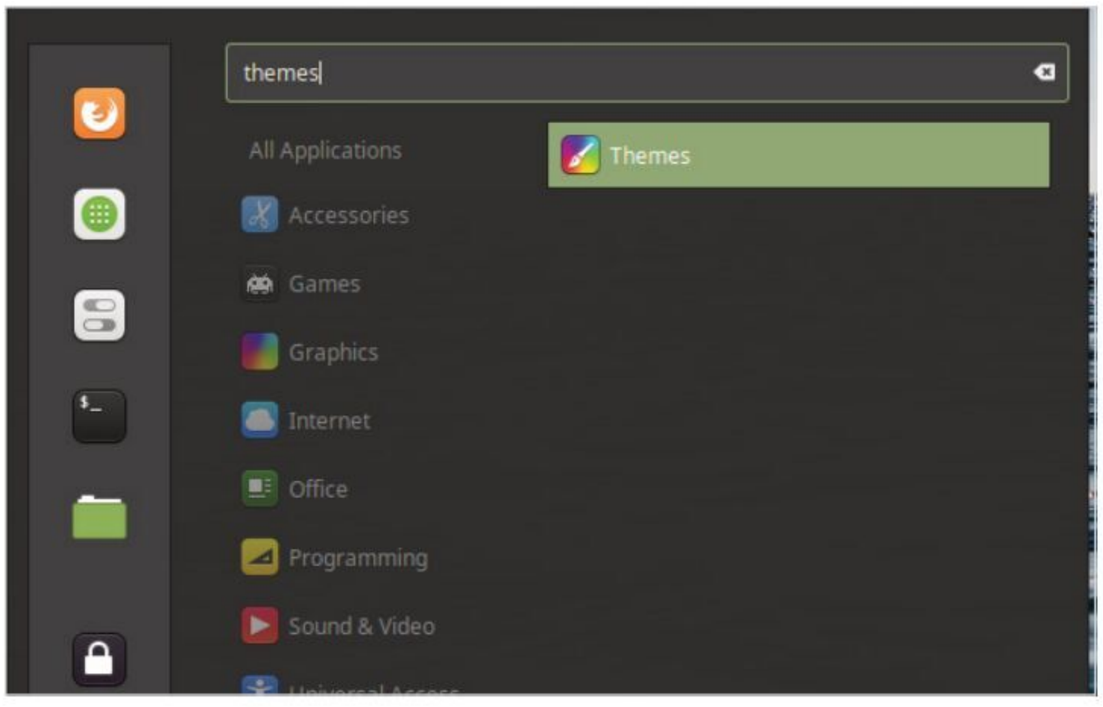
STEP 5 If none of the available wallpapers take your fancy, open a browser and search for the type of background image you prefer. When you've found the image you want as the desktop wallpaper, right-click it and choose Set As Desktop Background from the list of options. When the Set Desktop Background console open, click the Set Desktop Background button.



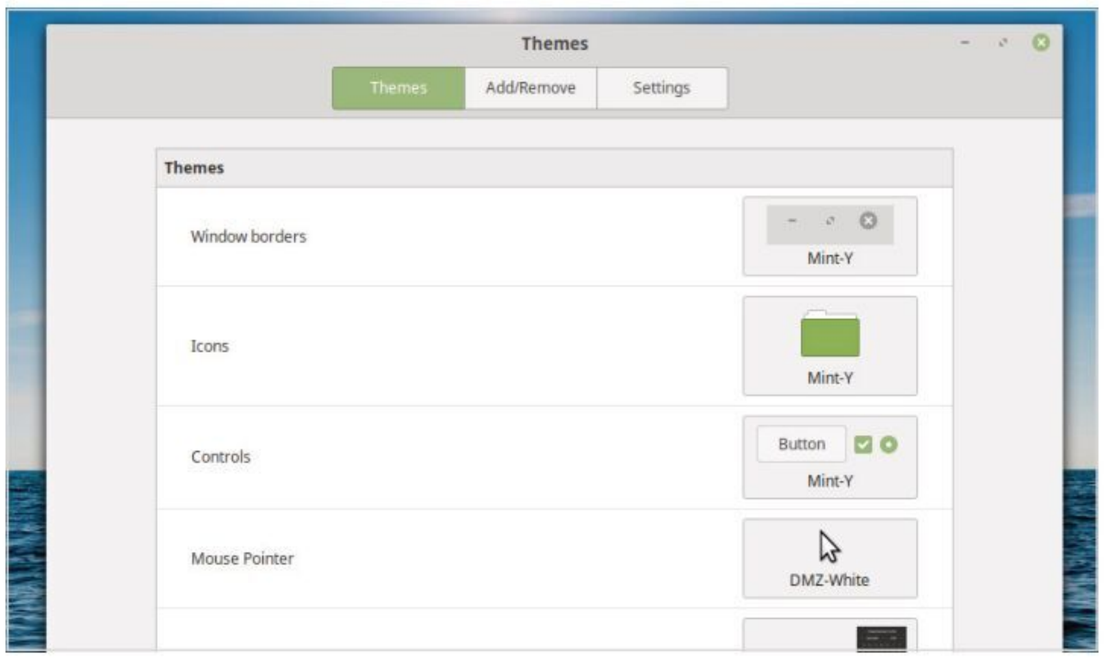
STEP 6 Click the Menu button and search for 'desktop' and click the Desktop Settings result. The Desktop console allows you to pick the layout, desktop icons and options for multi-monitor support you want. You can experiment with the options for the best setup, according to your personal tastes.



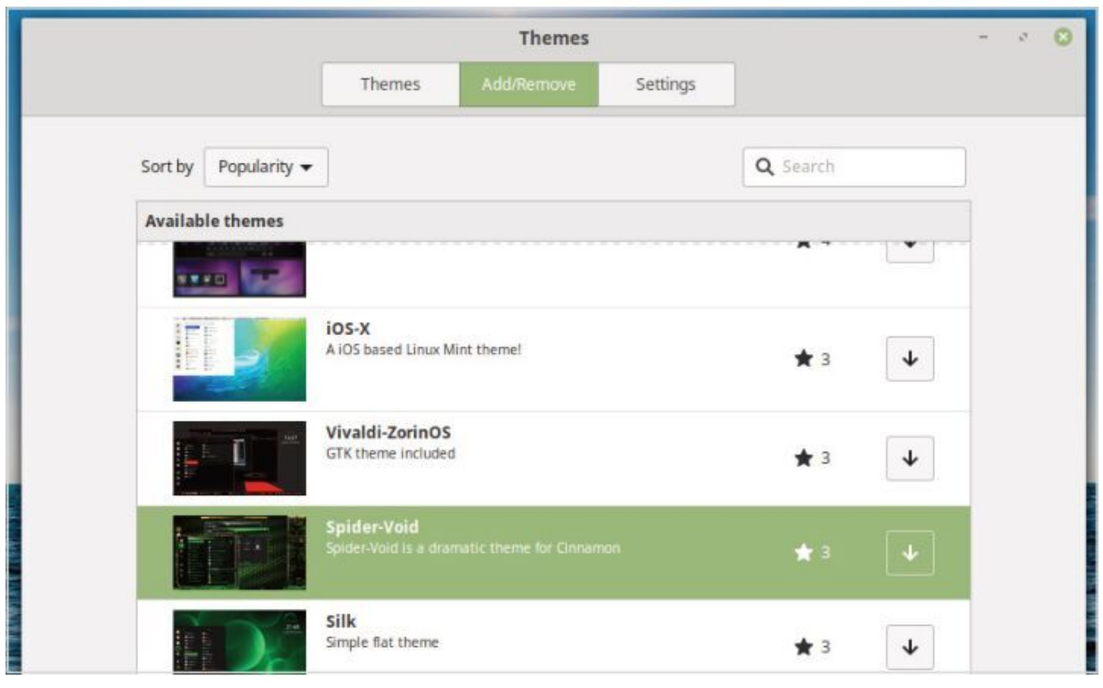
STEP 7 In addition to changing the desktop wallpaper, and how the icons are displayed, you can also alter the overall theme for Linux Mint. From the Mint Menu, search for 'themes' and click the Themes app as it appears in the search results.



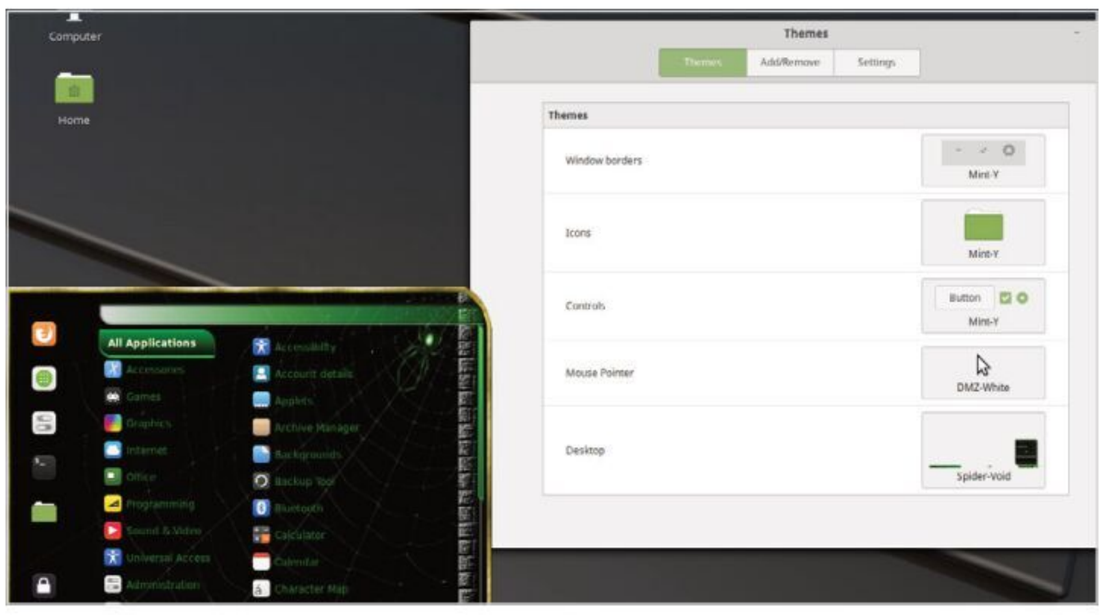
STEP 8 Themes allows you to change the way certain aspects of the Mint desktop look: Window Borders, Icons, Controls, Mouse Pointer and Desktop. In the Settings tab you can extend the options with a few on/off slider buttons, too.



STEP 9 If you click on the Add/remove button in the centre of the three available options, you can choose the default view from a number of preinstalled themes. Click the theme you want, then click the Install (downward-pointing arrow) button to enable it.



STEP 10 Click back on Themes, then Desktop and you can locate the newly installed theme and apply to your desktop. Any installed Themes can also be uninstalled via the Add/Remove button. It's worth spending some time personalising your desktop how you want it and there are some incredible themes available too.



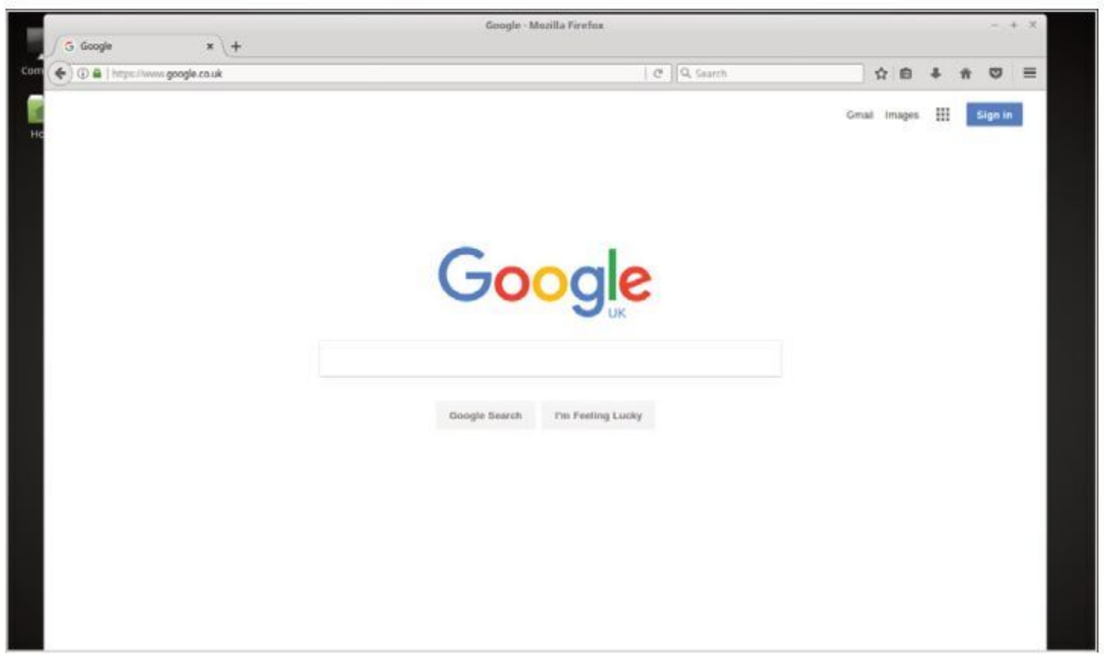
Becoming Anonymous Online

The digital age has led to many great advances in communications; however, it has also brought on a new age of spying and snooping. Most of us are no strangers to the frequent news stories of governments, secret organisations and underground hackers breaking our privacy but how can you combat this?

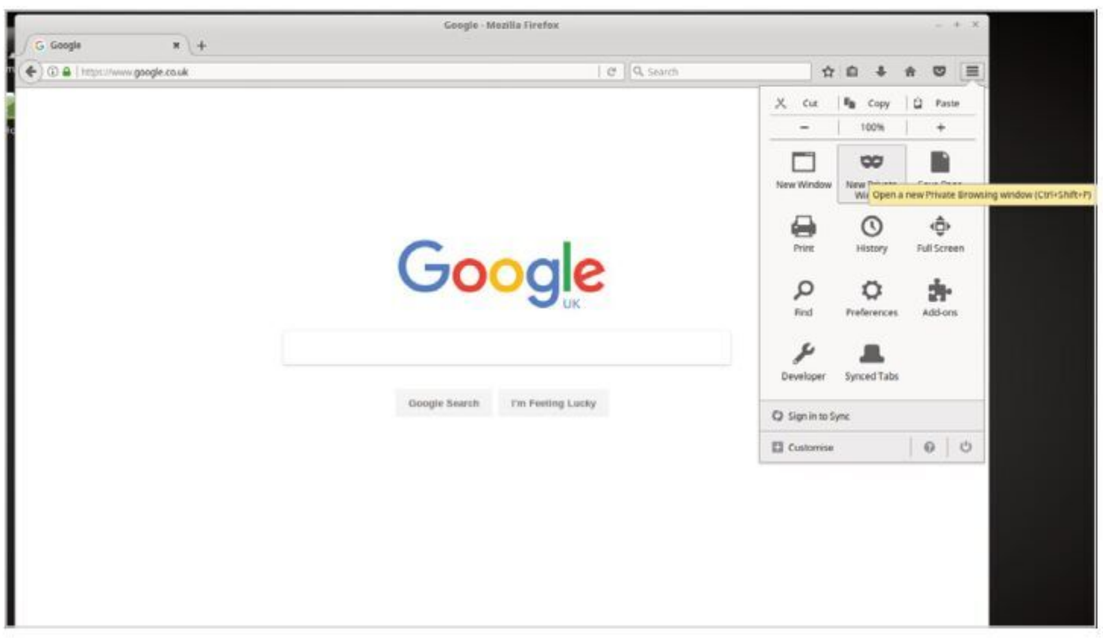
ANONYMITY WITH MINT

While it's virtually impossible to become totally anonymous online, you can take measures to ensure our privacy is at its best.

STEP 1 Starting with the basics, use HTTPS instead of the standard HTTP when browsing. This means that anything that's transmitted over HTTPS is secure (hence the S part at the end) and encrypted.

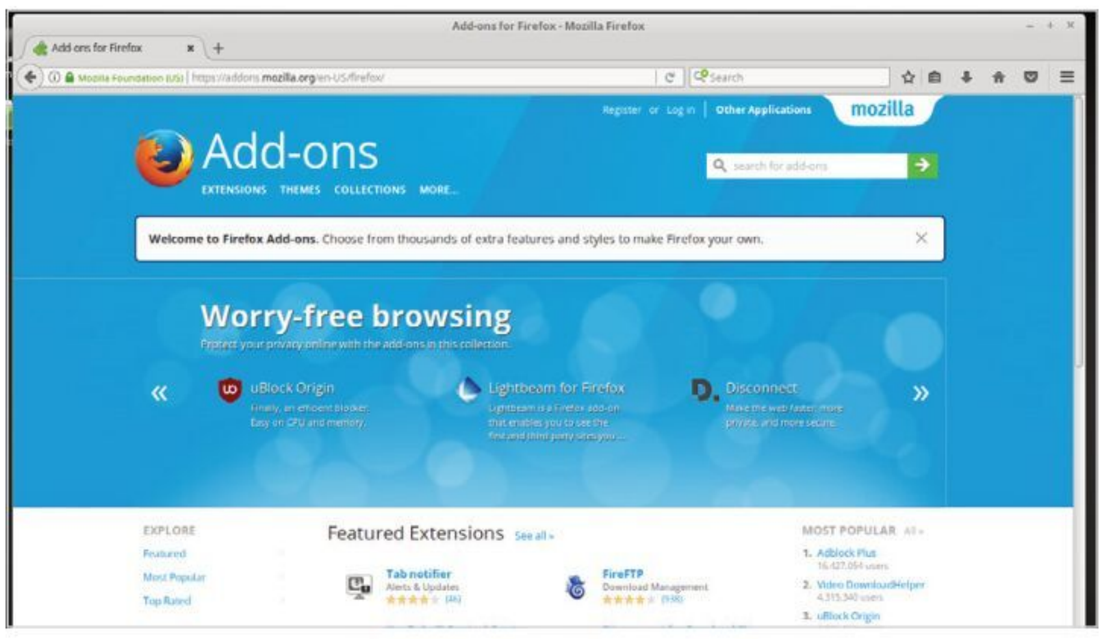


STEP 2 When you're browsing, consider using the Incognito or Private browsing modes available in a browser. This disables your web history and web cache, allowing you to browse without the details being stored for later scrutiny by someone else. However, it doesn't stop any data or search tracking.



STEP 3 Although using Google may seem like the obvious choice for a modern Internet search, the company does track all searches made by an individual. Instead, consider an alternative search engine, such as DuckDuckGo, a search engine that doesn't store personal data or track you.

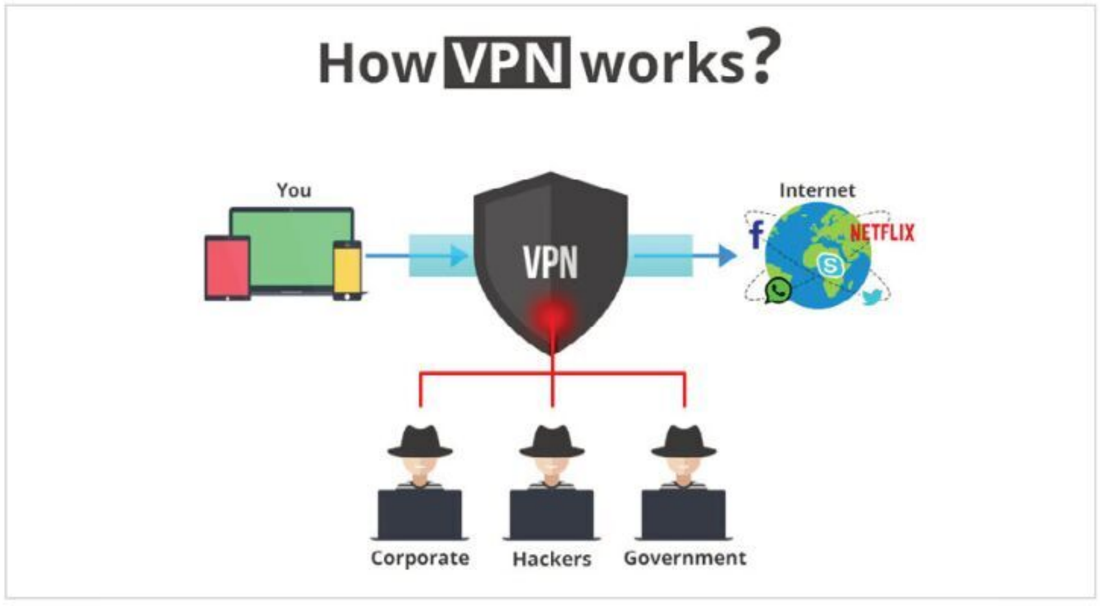
STEP 4 If you're regularly on the Internet then consider installing some of the browser plug-ins that enhance your privacy. For example, for Firefox, use Ghostery, NoScript and Adblock Plus to block trackers, adverts and other data mining techniques.



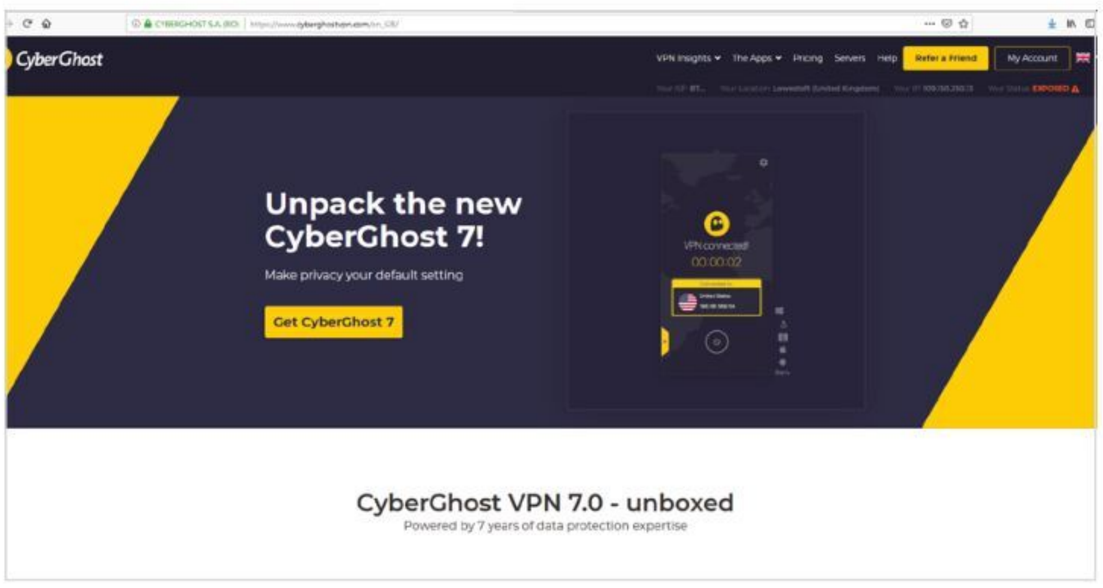
VPNS AND TOR

The previous steps can aid your online privacy but to really become anonymous you need a Virtual Private Network and Tor.

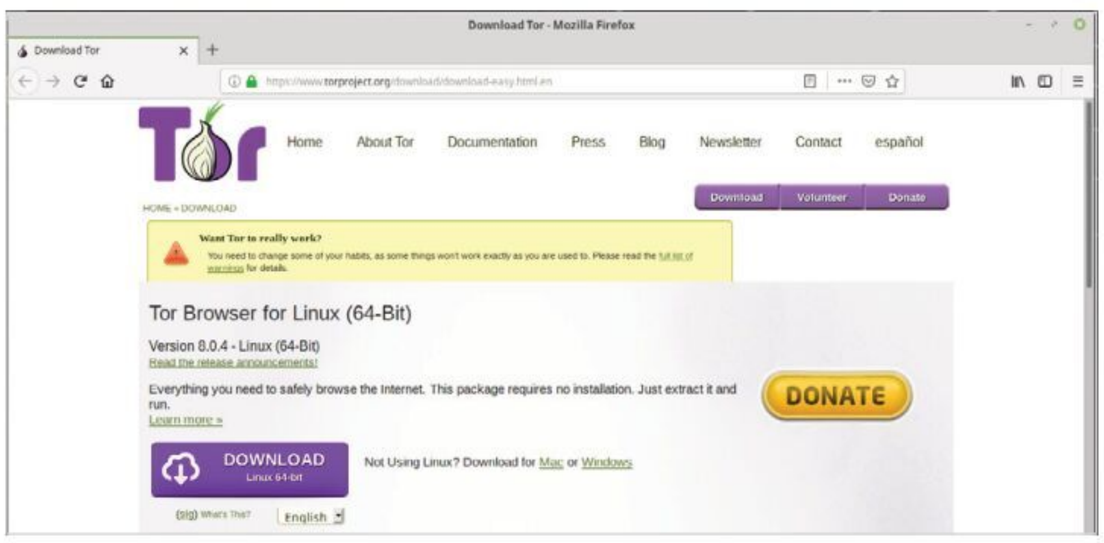
STEP 1 A VPN is a remote server, or cluster of servers, that establishes a connection with your computer. The end result is that your computer's identity on the Internet is hidden behind the VPN remote server; so you could live in the UK but have an IP address (the computer online identity) belonging to Iceland.



STEP 2 Most good VPNs charge a monthly or annual fee but it's worth the expense. We use CyberGhost, **www.cyberghostvpn.com**, which offers VPN connections for Windows, Mac, Mint (as well as other Linux distros), Android and iOS devices. Details for each OS can be found at **www.support.cyberghostvpn.com/hc/en-us/articles/213190329-Read-me-first-**.

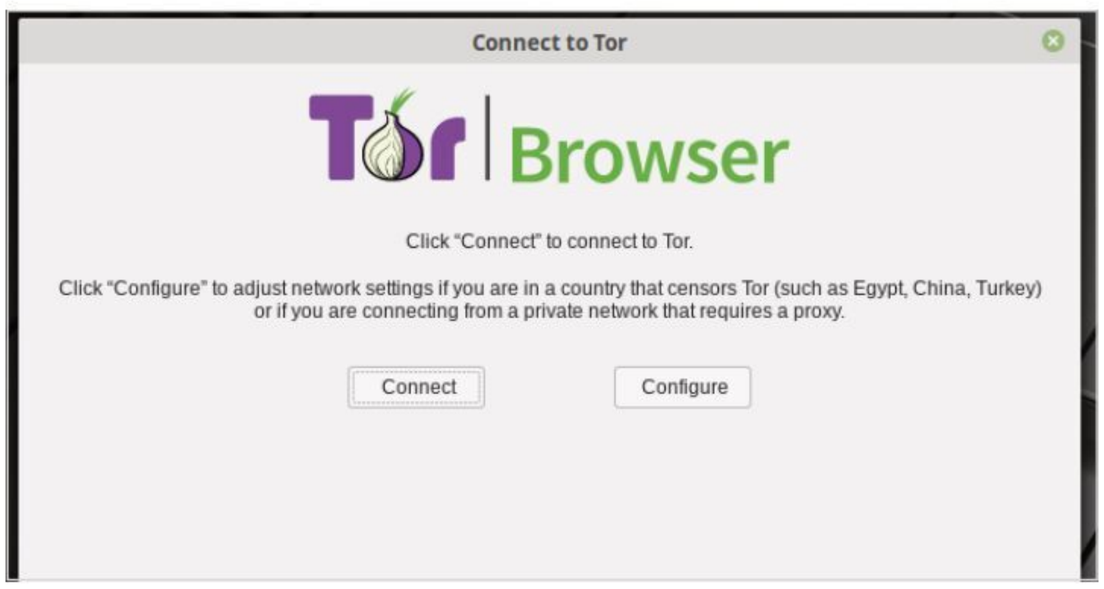


STEP 3 Another option is to use Tor. Tor is an open network that you can attach to that hides your IP address behind countless nodes around the world. It's available for Windows, Mac and Linux computers and is very easy to install and use. Start by navigating to **www.torproject.org/download/download-easy.html.en** and clicking on the Download Linux 64-bit button.

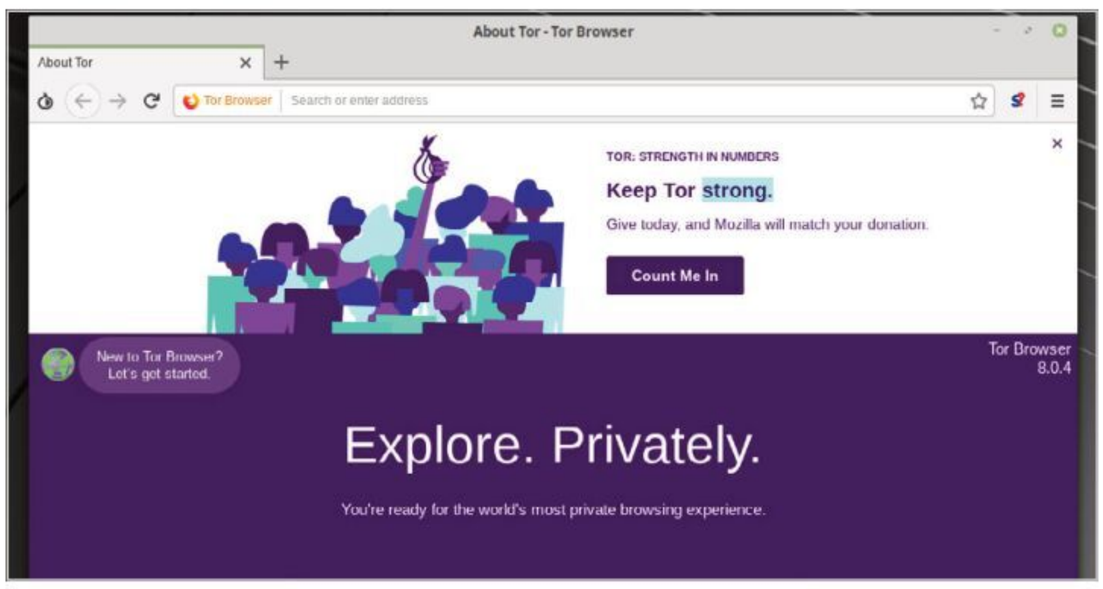


STEP 4 When the download has finished, drop into the Terminal and enter the Downloads folder, `cd Downloads/`. Enter `ls` to check the tar.xz Tor file is present, then enter: `tar -xf tor-browser-linux64-8.0.4_en-US.tar.xz` (Tor is updated regularly, so if your version is different press Tab to autocomplete the `tor-browser-` filename). When the files are unpacked, use `cd tor-browser_en-US/` to enter the new folder.

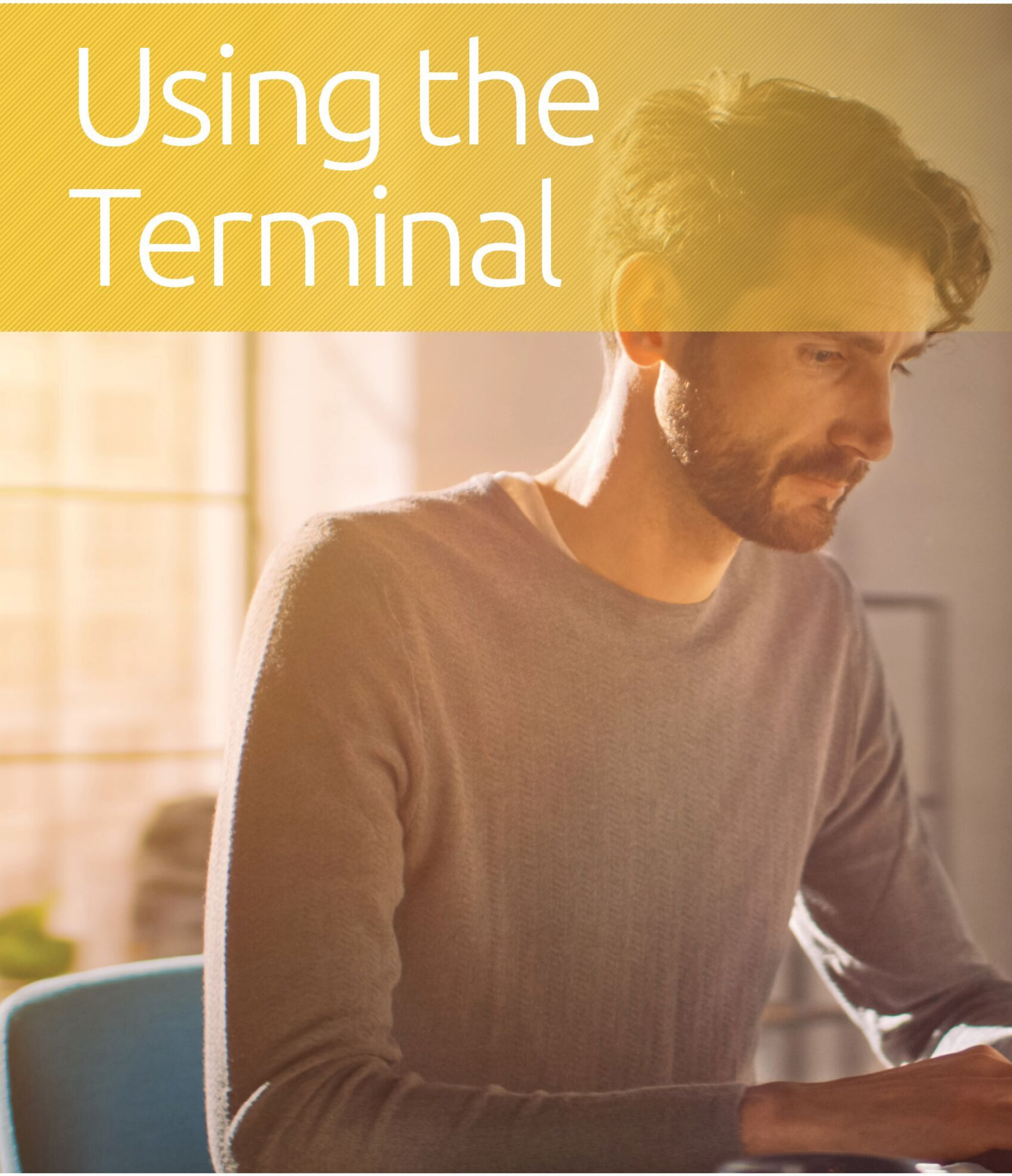
STEP 5 A quick `ls` reveals a couple of entries: a folder called Browser and a file called 'start-tor-browser.desktop'. To start the Tor setup, type `./start-tor-browser.desktop`. This command launches the Tor setup, where you are offered two options: Connect or Configure. For most users, the Connect option will suffice. Click Connect when you're ready.

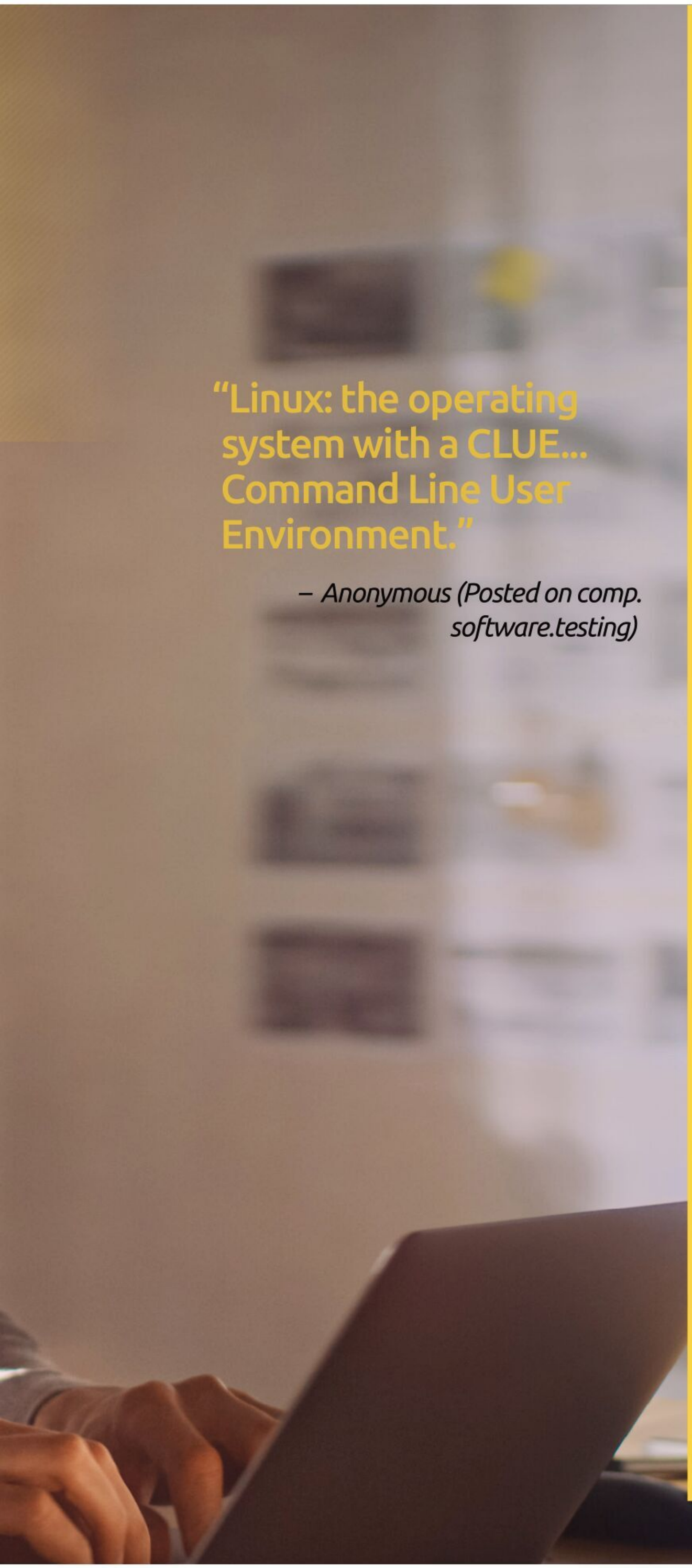


STEP 6 After the connection is established, the Tor Browser launches. This is a customised version of Firefox and from here you can securely browse the Internet without fear of being viewed or tracked. Mixing both a VPN and Tor makes for an extremely secure and private connection to the online world.



Using the Terminal





"Linux: the operating system with a CLUE... Command Line User Environment."

– Anonymous (Posted on comp. software.testing)

The Linux command line, the Terminal, is an incredibly powerful environment. From it, you can bring the OS to its knees, or update it to something spectacular. You can hack into remote computers, look at an animated ASCII camp fire, install new apps and programs, watch Star Wars played out from a server in the Netherlands, and code intricate automated scripts.

In this section you will explore the intricacies of the Terminal, and how it works with Linux. You will learn how to manipulate the Linux file system from the Terminal, and you will discover some odd, but fun things you can do in the Terminal.

The Terminal rules supreme in the world of Linux. Learn how it works, and you can access all that power.

46	Basics of the Terminal
48	Update Mint via the Terminal
50	Install Apps via the Terminal – Part 1
52	Install Apps via the Terminal – Part 2
54	Did You Know…Linux Kernel 0.01
56	Creating a File Using the Terminal
58	Creating and Removing Directories
60	Fun Things to do in the Terminal
62	More Fun Things to do in the Terminal
64	Linux Tips and Tricks
66	Did You Know…Linux and the Big Bang
68	Creating Bash Scripts– Parts 1 - 5
78	Pi x Linux = The Perfect Combination
80	Command Line Quick Reference
82	A-Z of Linux Commands
84	Did You Know…Good enough for NASA

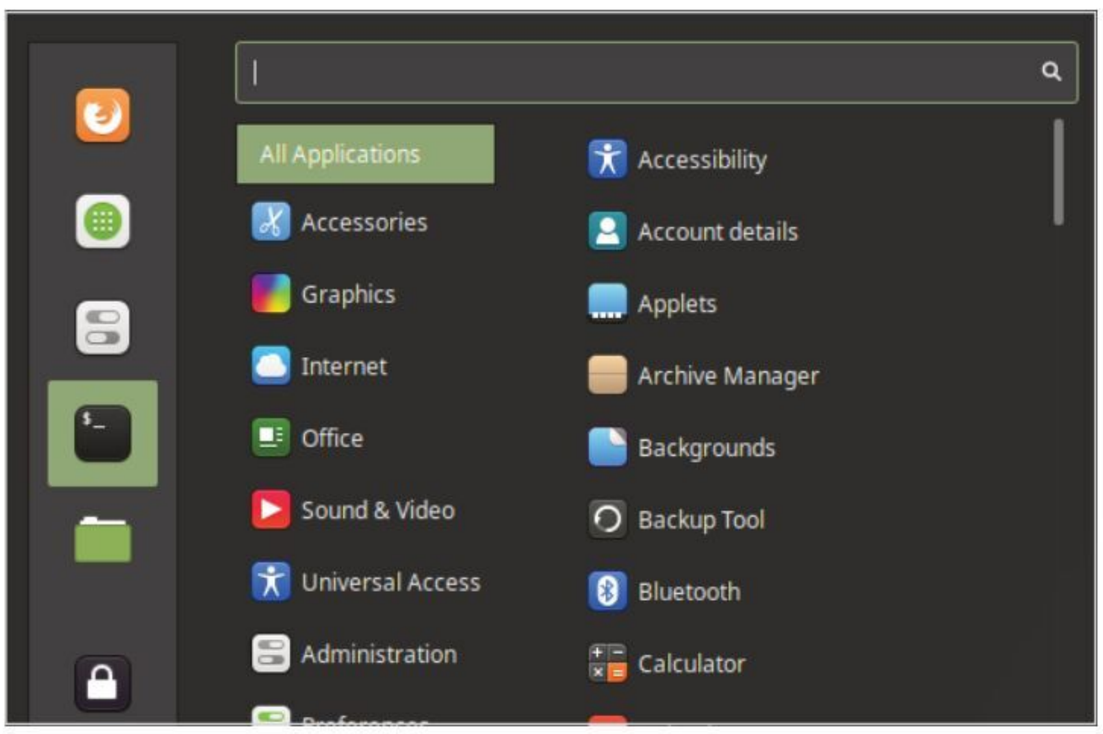
Basics of the Terminal

Most operating systems use two kinds of interface, the GUI, which is the desktop that Windows, macOS and Linux Mint boot into and the command line. While modern operating systems shy away from the command line, Mint uses the Terminal to give the user greater control of the system.

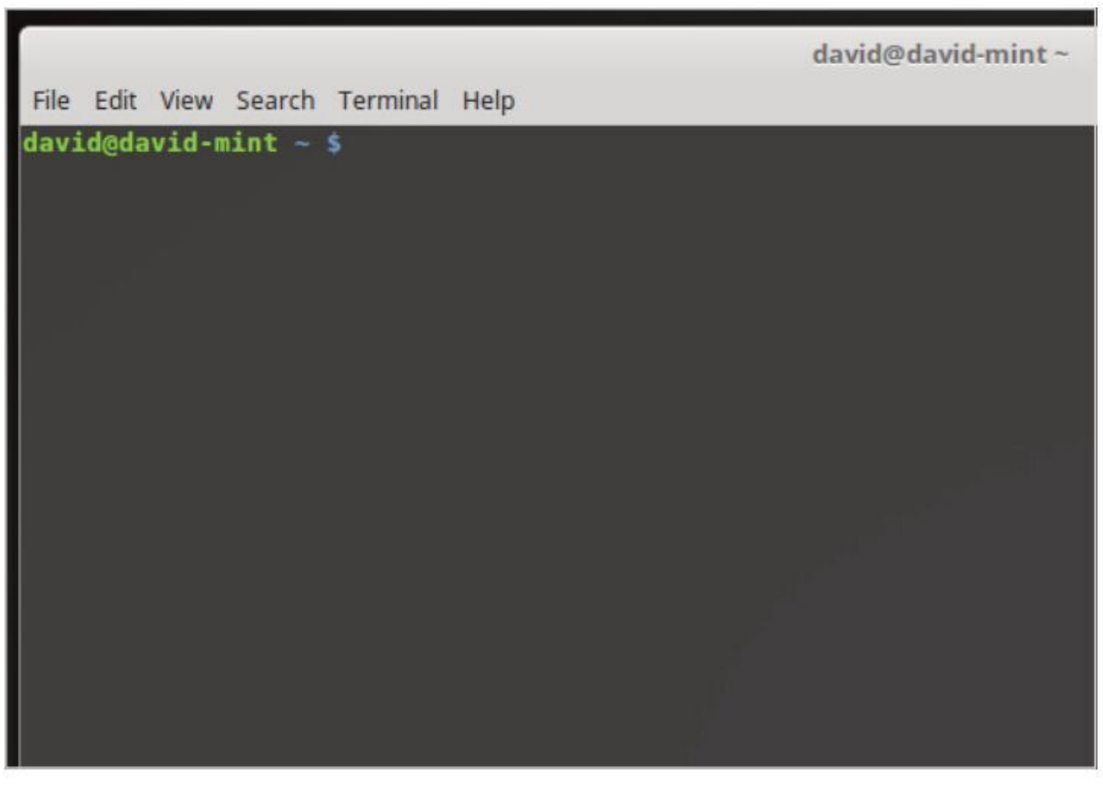
TAKING COMMAND

The command line, or Terminal, is an extremely powerful interface. Everything you can do on the desktop can be done within the Terminal. Let's start by seeing how it works.

STEP 1 The Terminal can be accessed by either clicking on the Terminal icon on the Panel, located between the Firefox and files icons or launched by opening the Menu and selecting it from the left-hand quick launch strip.

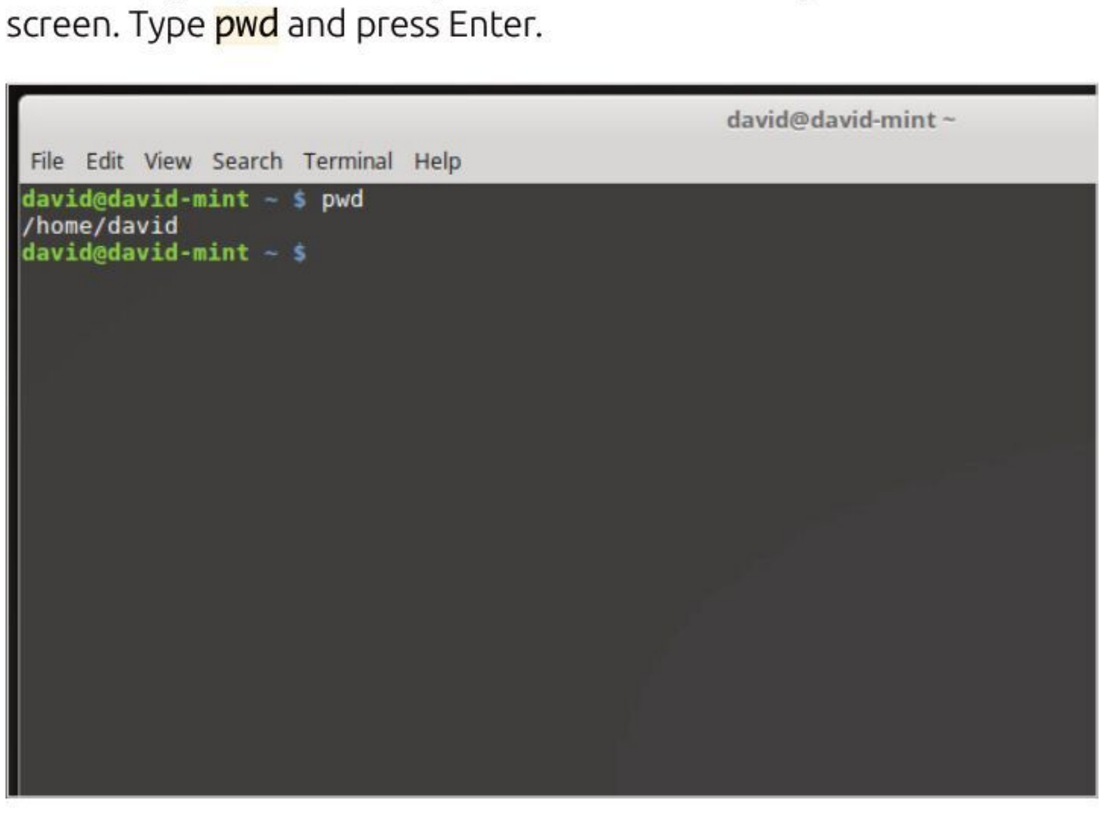


STEP 2 The Terminal gives you access to the Linux Mint Shell, called Bash, which gives you access to the underlying operating system. Everything in Mint, including the desktop and GUI, is a module running from the command line.

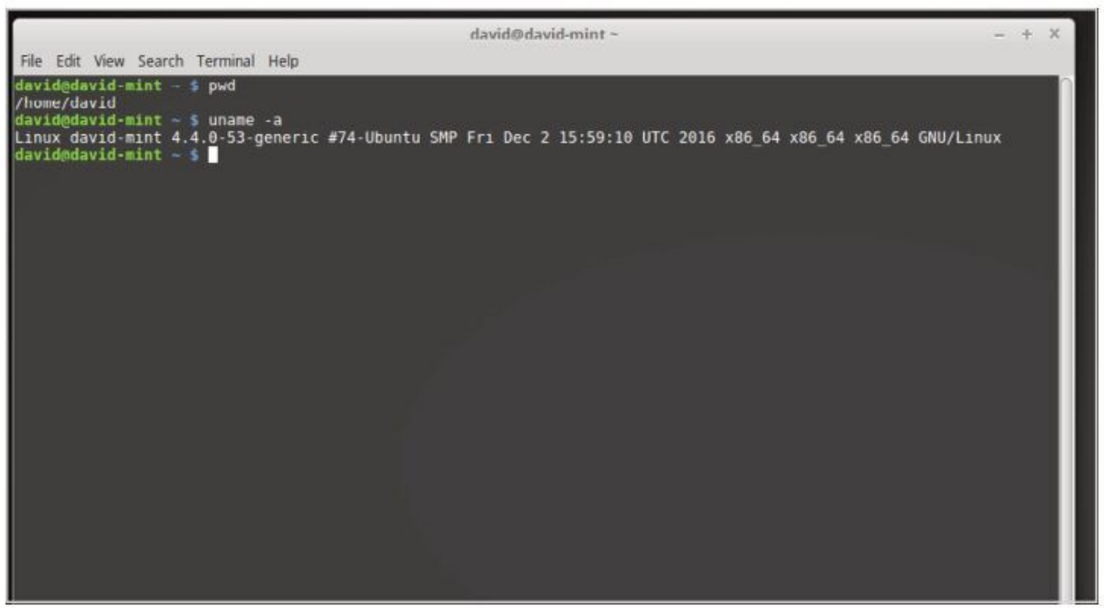


STEP 3 What you currently see in the Terminal is your login name followed by the name of the computer; as you named it during setup when you first installed Mint. The line then ends with the current folder name; at first this is just a tilde ~, which means your Home folder.

STEP 4 The flashing cursor at the very end of the line is where your text-based commands are entered. You can begin to experiment with a simple command, Print Working Directory (pwd), which outputs the current folder you're in to the screen. Type pwd and press Enter.

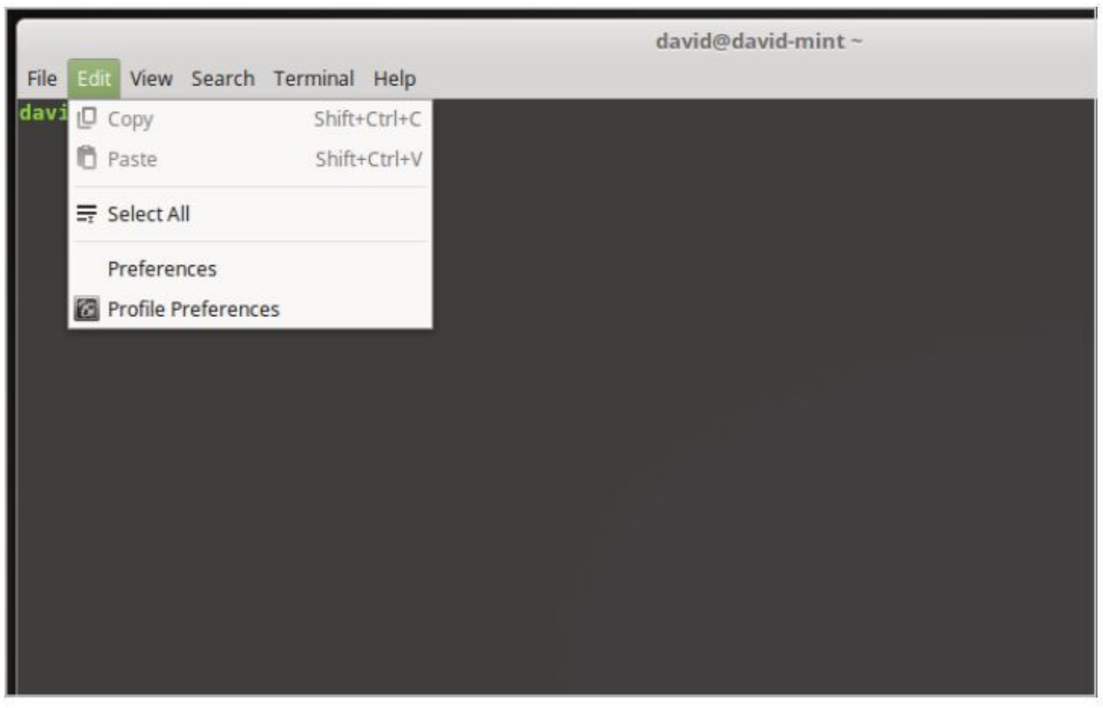


STEP 5 All the commands you enter work in the same manner: you enter the command, include any parameters to extend the use of the command and press Enter to execute the command line you've entered. Type into the Terminal: `uname –a` and press Enter. This displays some system information regarding Mint.

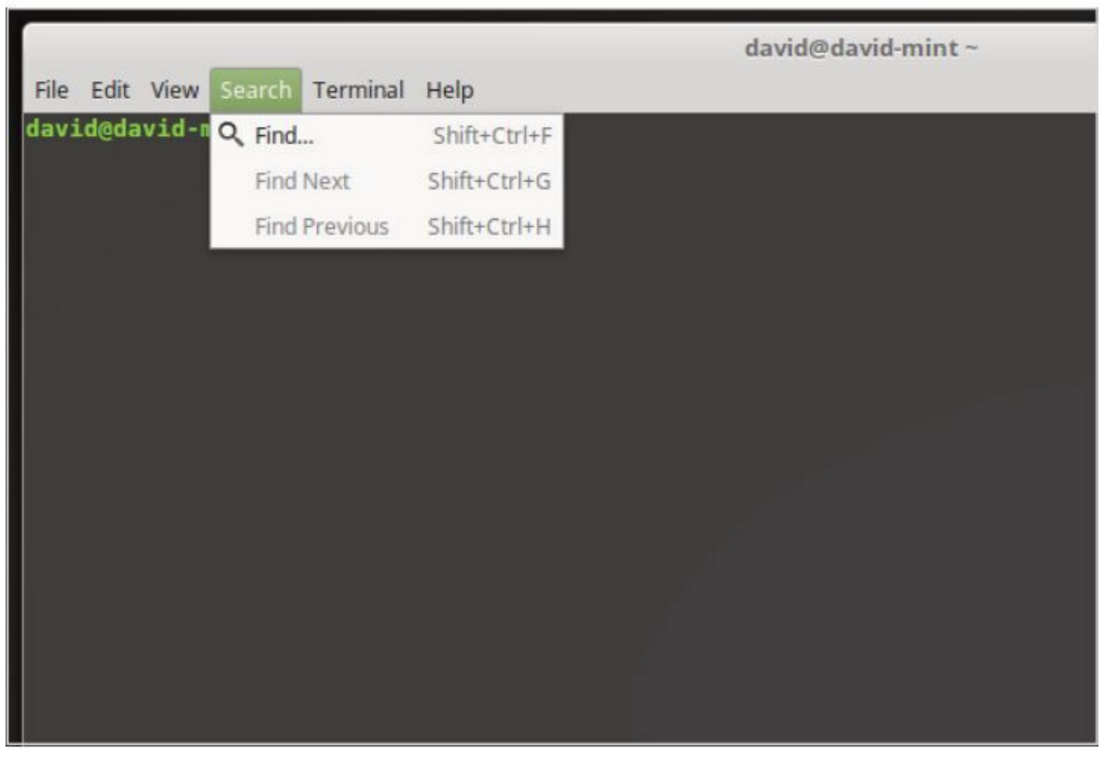


STEP 6 Before we get into entering commands, let's take a moment to see what menus the Terminal has to offer. The File menu option allows you to open a new Terminal, create a new profile, where you can alter the size, colours and behaviour of the Terminal, add a new tab, and close all current active Terminal sessions.

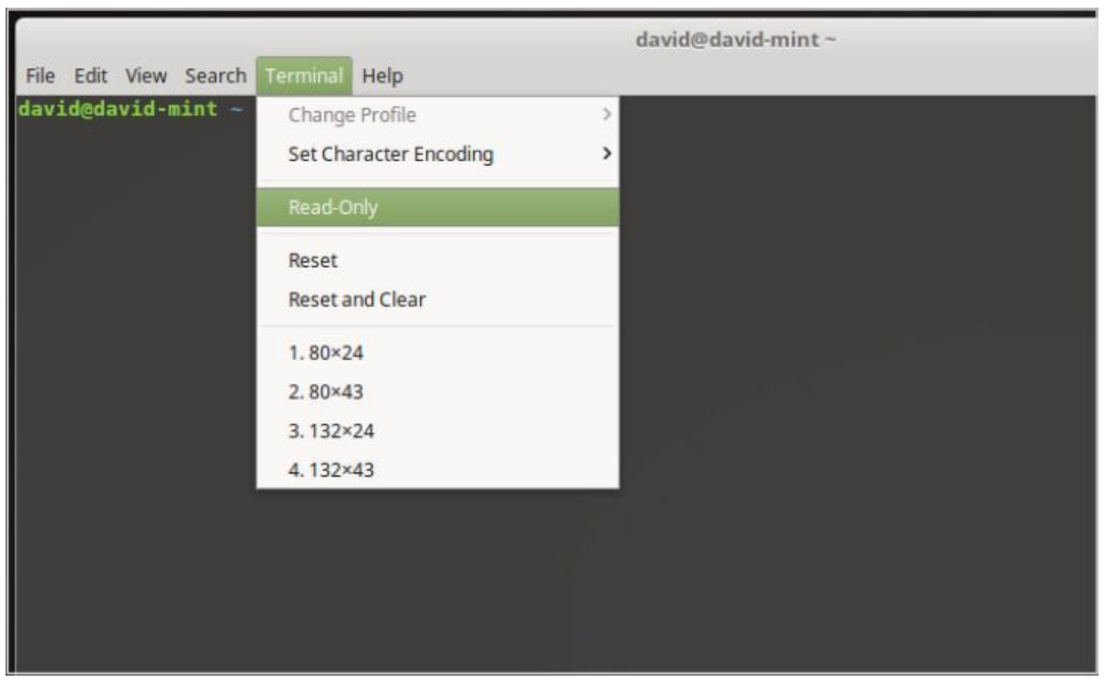
STEP 7 The Edit option lets you copy and paste commands to and from the Terminal and other sources; handy for when you want to copy a very long and complex command from a web page. It also allows you to edit the current profile preferences.



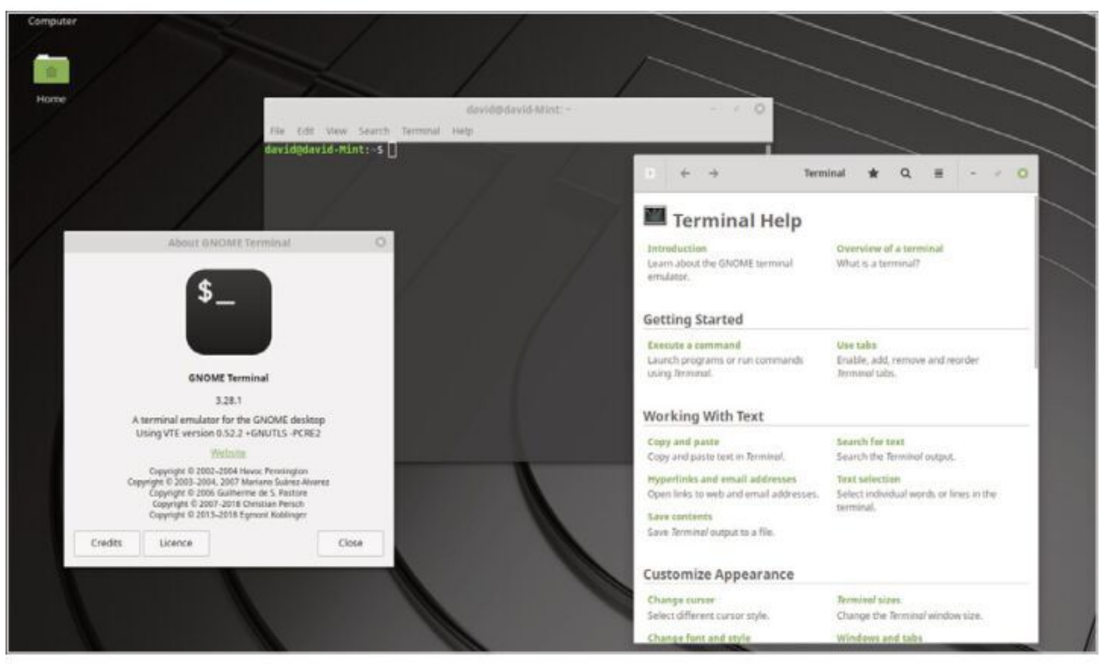
STEP 8 View and Search options let you alter the sizing of the Terminal window and of course search within the Terminal for any particular key words.



STEP 9 The Terminal entry extends the profile editing and sizing and allows you to alter the character encoding. Interestingly, you can also set it to a Read Only mode, which stops you from entering any commands into the Terminal; this is good for when you need to permanently display the Terminal contents.



STEP 10 Finally, the Help option displays the help contents and version number of the Terminal, or to be more precise, GNOME Terminal; we'll simply refer to it as Terminal in future. The Contents are worth having a quick read through, to help familiarise yourself with how the Terminal works.



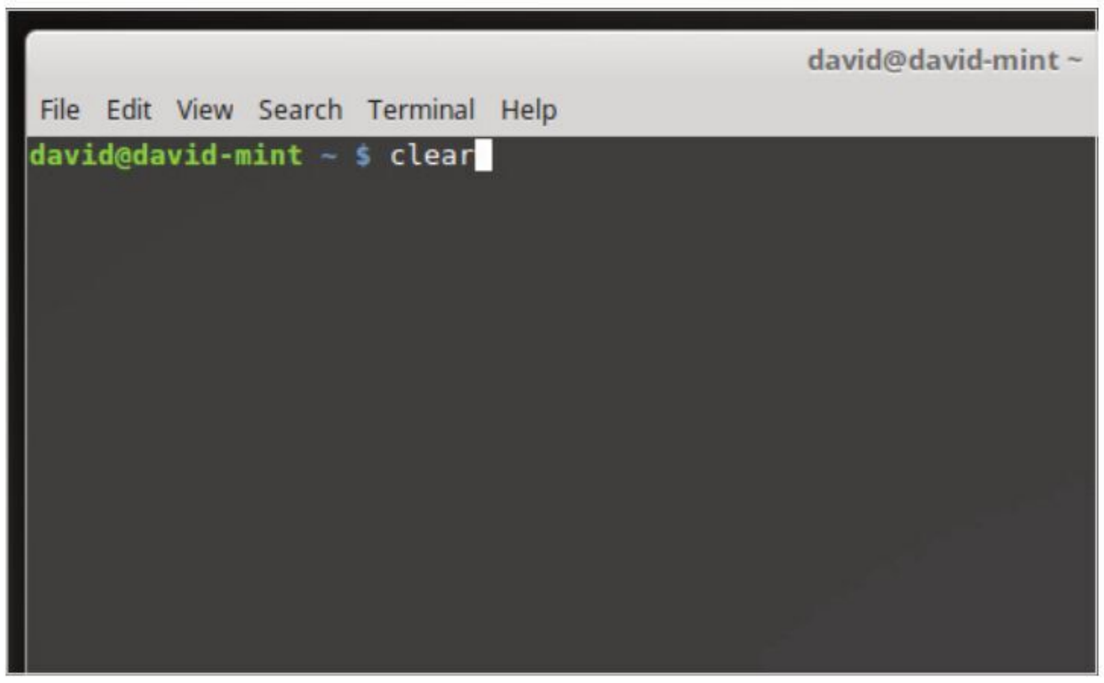
Update Mint via the Terminal

Up to now you've been using the shield icon to launch Mint Update Manager in order to update the system and upgrade the currently installed apps, tools and other elements. However, you can also accomplish a complete system update and upgrade from the Terminal.

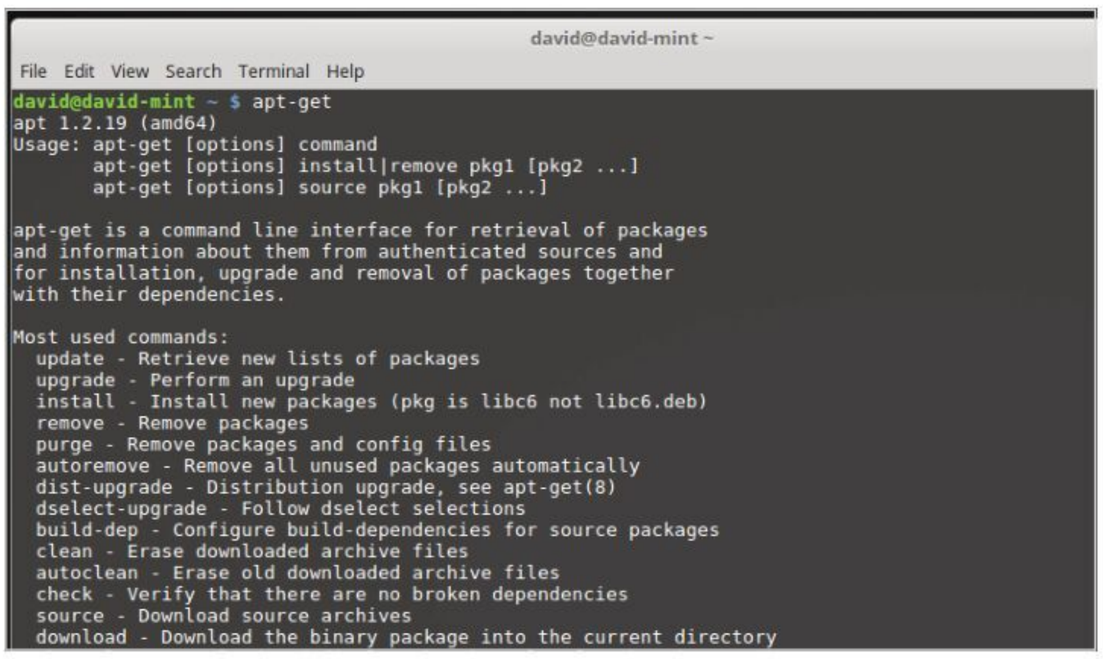
USING APT-GET

To update and upgrade via the Terminal you use the APT (Advanced Packaging Tool) command. It's a powerful command and combines different elements depending on its use.

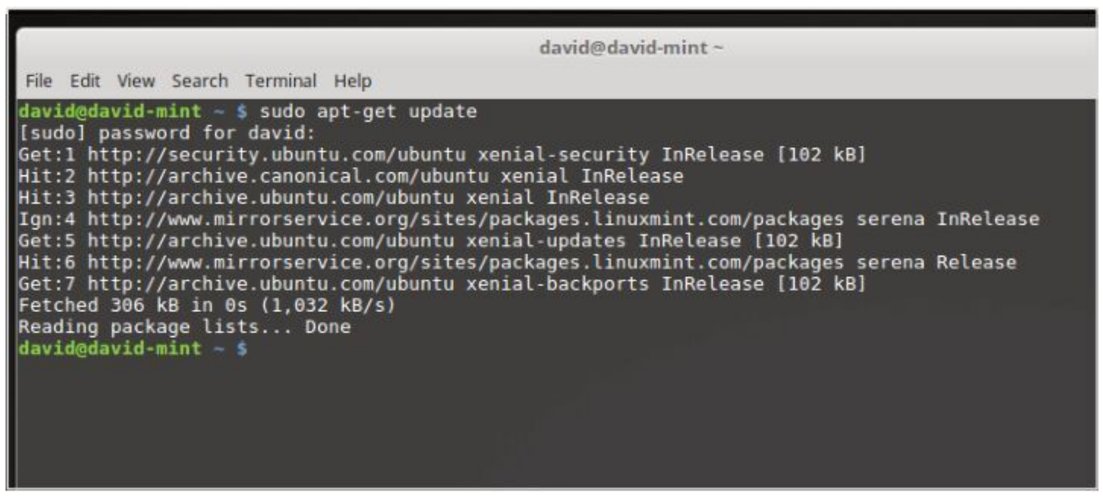
STEP 1 Start by opening a new Terminal or if you already have one opened clear its contents with the clear command. This starts you off with a clean slate on which to work.



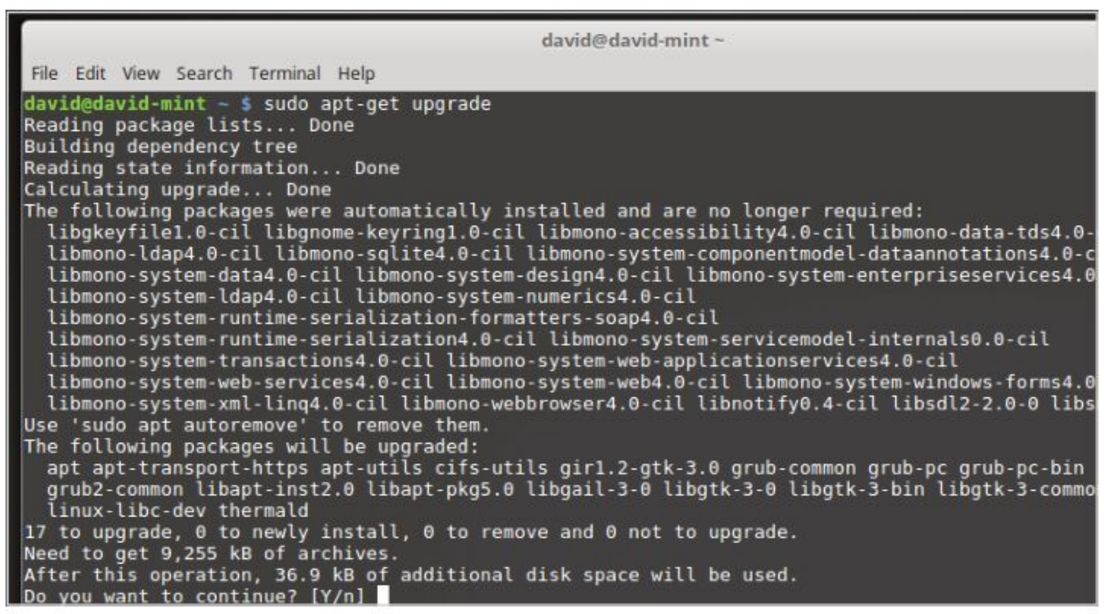
STEP 2 Enter: `apt-get` into the Terminal. This brings up a list of the most used apt-get commands, along with a brief description of what the command does. It's worth having a look at, even if it doesn't make a huge amount of sense at this time.



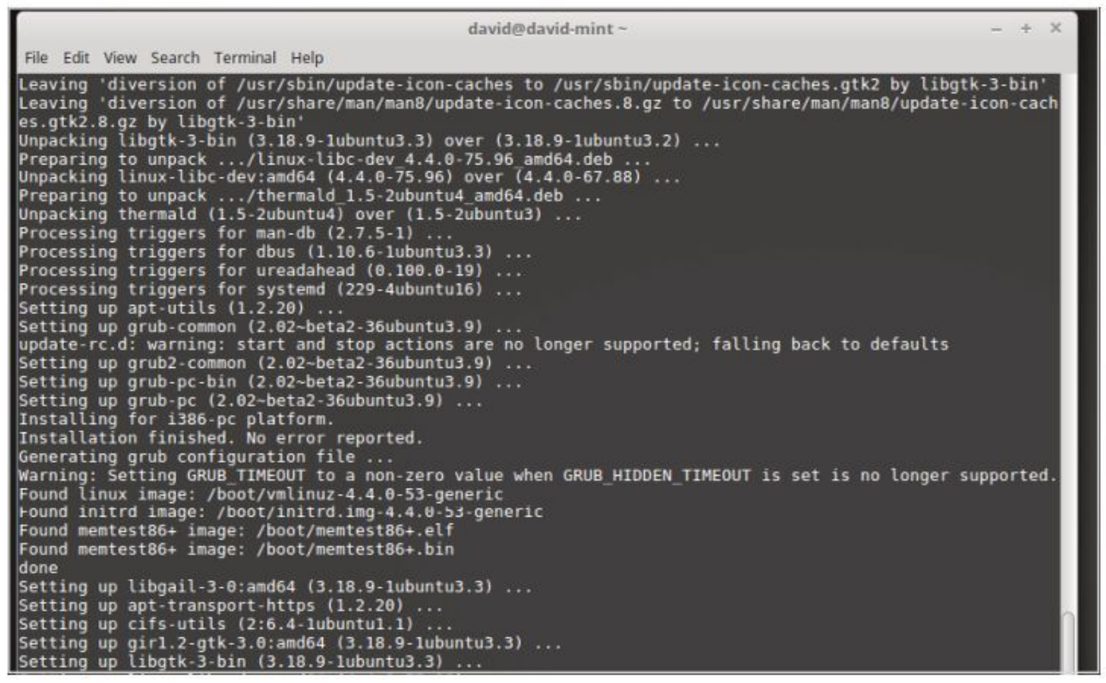
STEP 3 Apt-get is used to update and upgrade the software in Mint, as well as Ubuntu and other Debian-based distros. Using the Update element retrieves new package lists and updates the list of source files. Upgrade downloads and performs an upgrade to the latest versions of those files. To start the entire Upgrade and Update process, enter: `sudo apt-get update`, followed by your password.



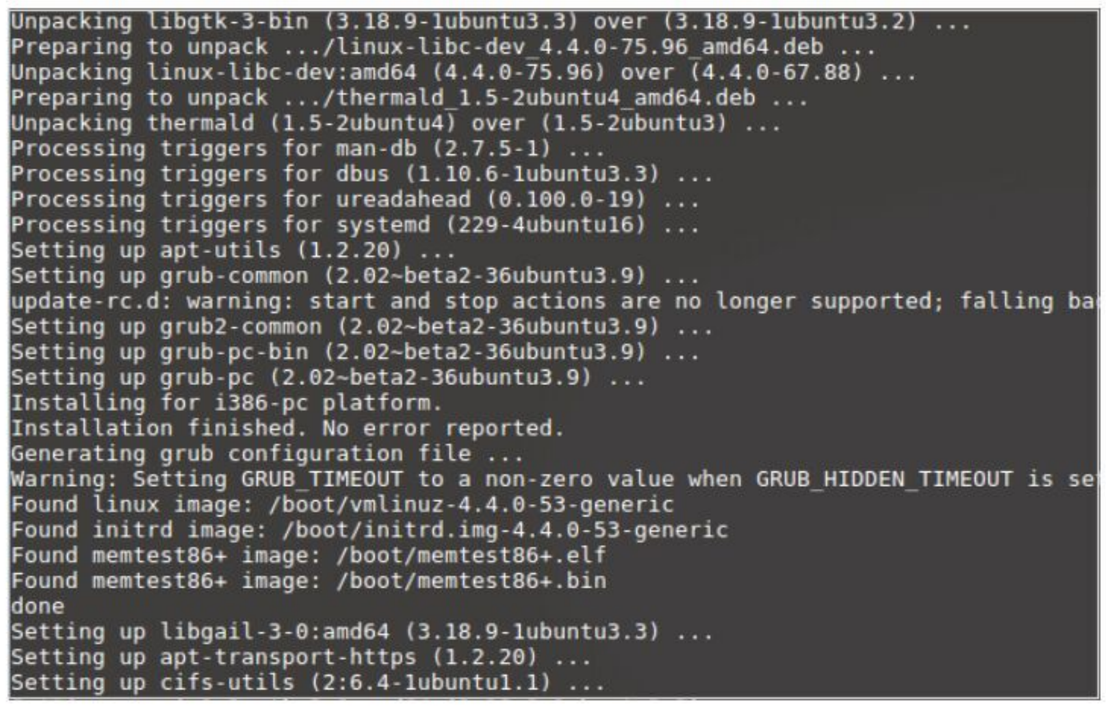
STEP 4 Notice now the addition of the sudo command. The sudo command once meant Super User Do; these days it's more acceptable as Substitute User Do. It means that the administrative user (Super User) uses APT (Advanced Packaging Tool) to Get any Updates. Now try this: `sudo apt-get upgrade`.



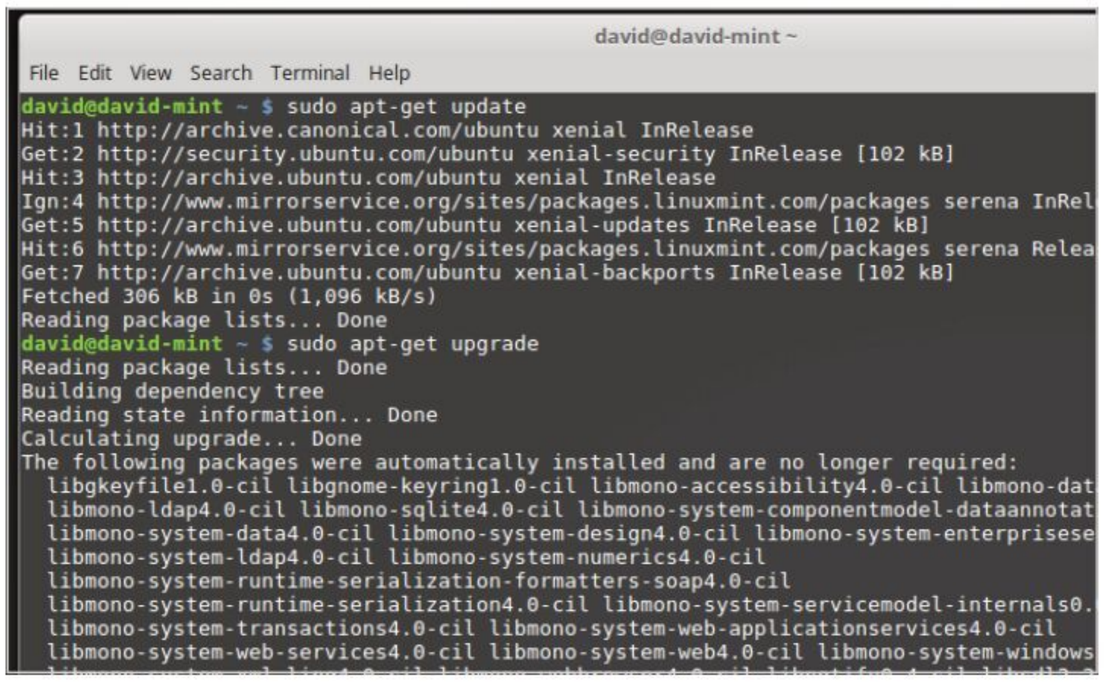
STEP 5 Depending on the state of your updates, if you have any waiting to be installed, you might be asked if you want to apply the results of the `sudo apt-get upgrade` command. You can press y to accept and continue. What's happening here is that apt-get has some updated software to apply to Mint, and you're okaying the action.



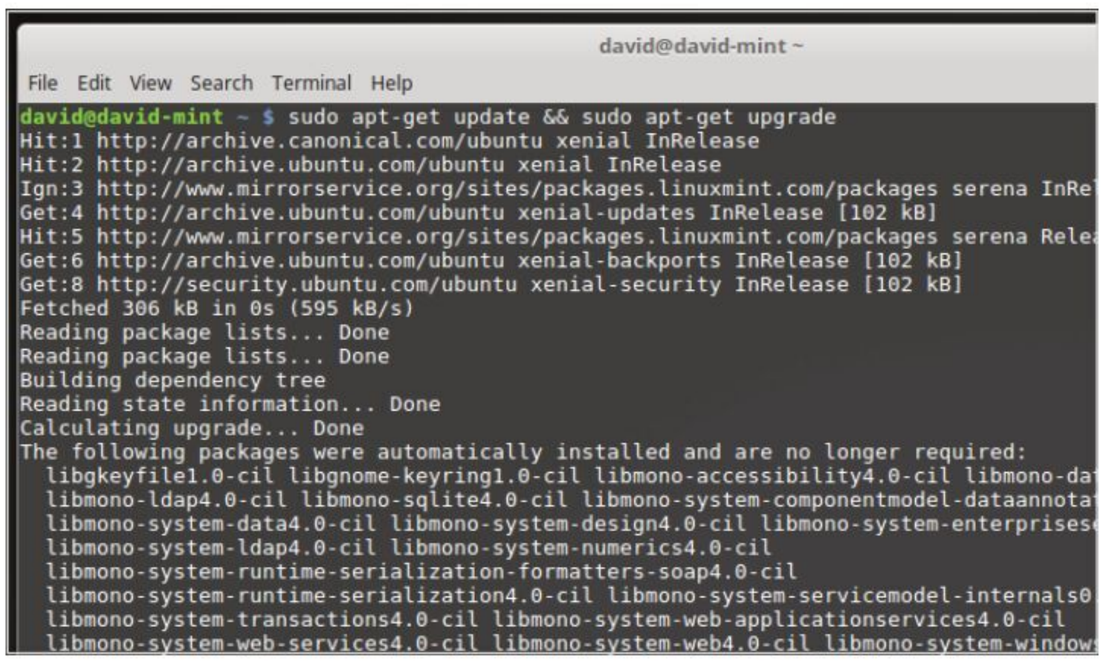
STEP 6 There's likely to be a long list of what seems gibberish now filling your Terminal window but don't worry. The files necessary for the upgrade have been downloaded, prepared, unpacked, processed, installed and set up correctly. There's a lot going on when you perform an upgrade, even with the smallest package.



STEP 7 Essentially, that's it, your system is now up to date according to the available list of packages from the apt-get update command. You can run through the process one more time, just to check if everything went okay. To recap, enter: `sudo apt-get update`, press `Enter`, then type: `sudo apt-get upgrade` and press `Enter`.



STEP 8 Interestingly, Linux Mint, among other distros, offers you the ability to chain several commands together. In this example, therefore, we can use `sudo apt-get update && sudo apt-get upgrade`. The double ampersand is what combines the commands and works perfectly, providing the preceding command went without a hitch. It's recommended to start any session with the update and upgrade combo.



TERMINAL VS UPDATE MANAGER?

Why use the Terminal to update and upgrade over the Update Manager, regardless of the distro you're using? Some users greatly prefer using the Terminal to update their Linux systems and accompanying apps, in the belief that it's better. However, that's not often the case.

Using the Terminal, apt-get upgrade, doesn't handle changing dependencies between versions of packages, so if a package has its dependent files changed from one version to another, then the upgrade is held back.

The Update Manager, or Software Manager (depending on the distro), often phases its updates and marks those packages with changed dependencies for updating. However, and this is where Linux can often get confusing, sometimes it doesn't.

It all boils down to the developer of the package being updated and the way the package is held in the distro's repositories and whether the update is classified as stable or not. In essence, from the point of view of the user, if you update and upgrade using both the Terminal and the Update Manager regularly, then you will be as up to date as possible, and get the essential and necessary stable versions of the packages and core software. If you're looking for cutting edge package updates, then it's best to opt for a rolling release distro instead.

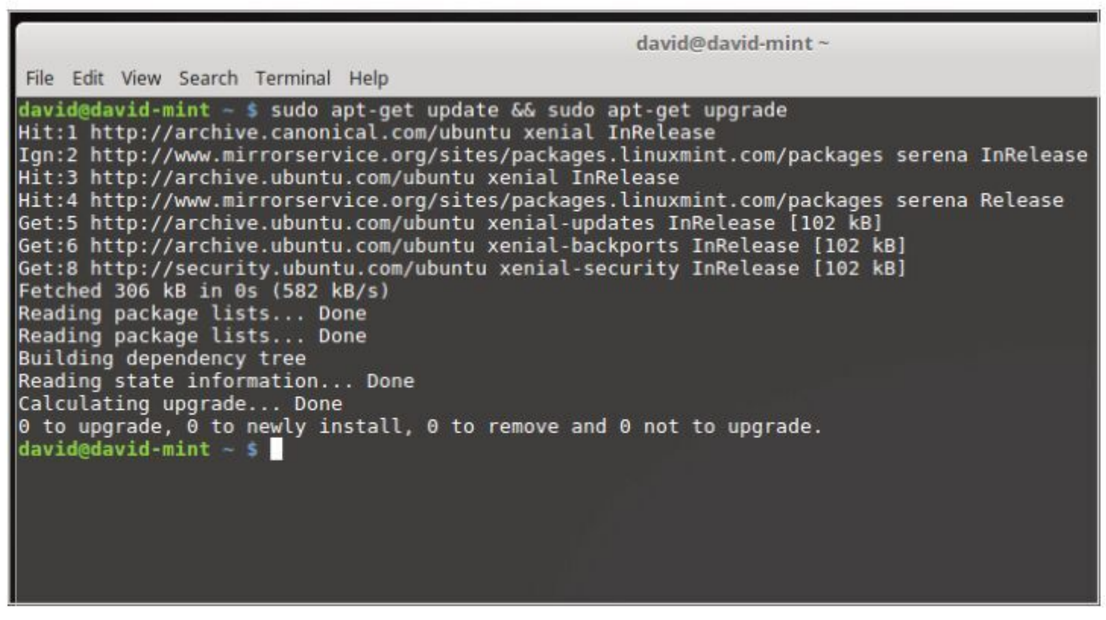
Install Apps via the Terminal – Part 1

There are different ways to install apps and programs on Linux. You can opt for the graphical route, using a Software Manager, or you can use the Terminal. Often, the Terminal provides better control over the software being installed and sometimes, you have no choice in the matter.

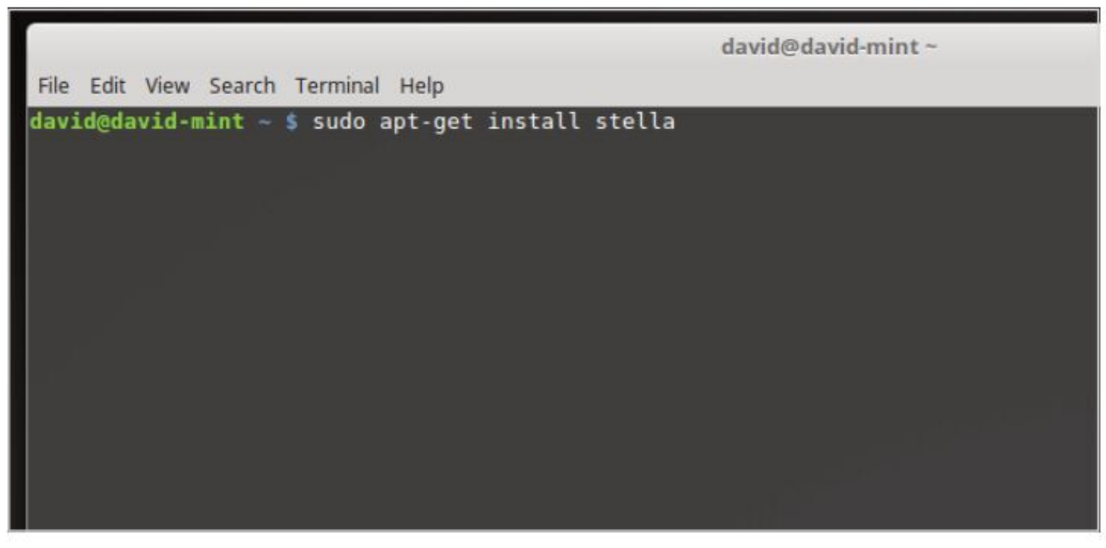
COMMAND LINE INSTALLS

Installing an app with the Terminal may require some nifty keyboard work but you get a better sense of what's being installed and where.

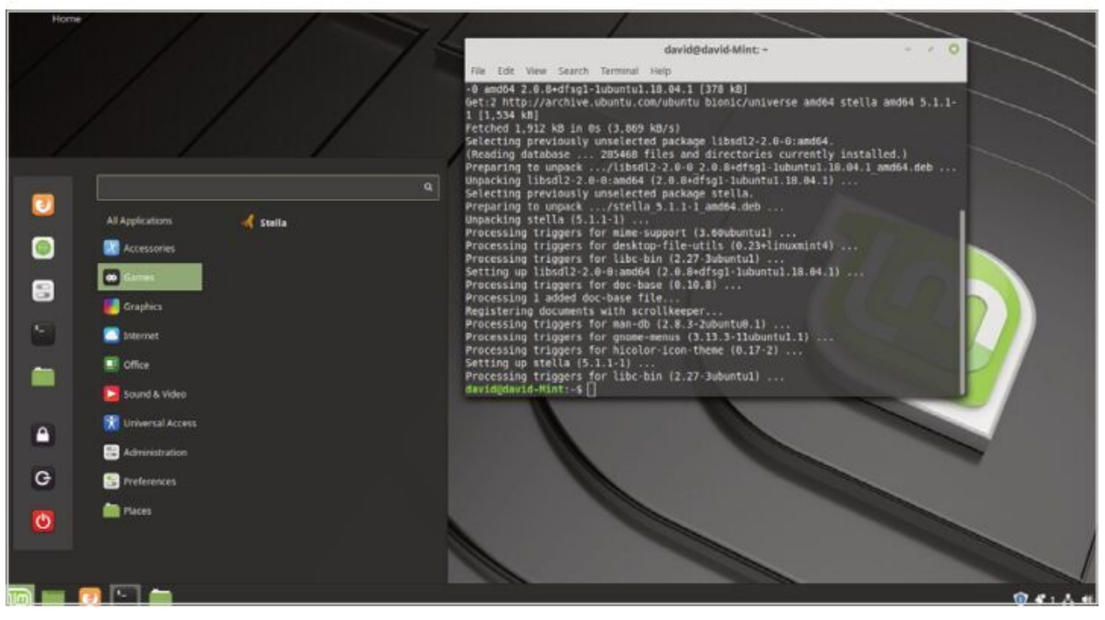
STEP 1 Installing apps form the Terminal is often relatively simple. First though, you need to make sure that the system is up to date. To do this open up the Terminal and enter: `sudo apt-get update && sudo apt-get upgrade`. Enter your password and accept any necessary updates.



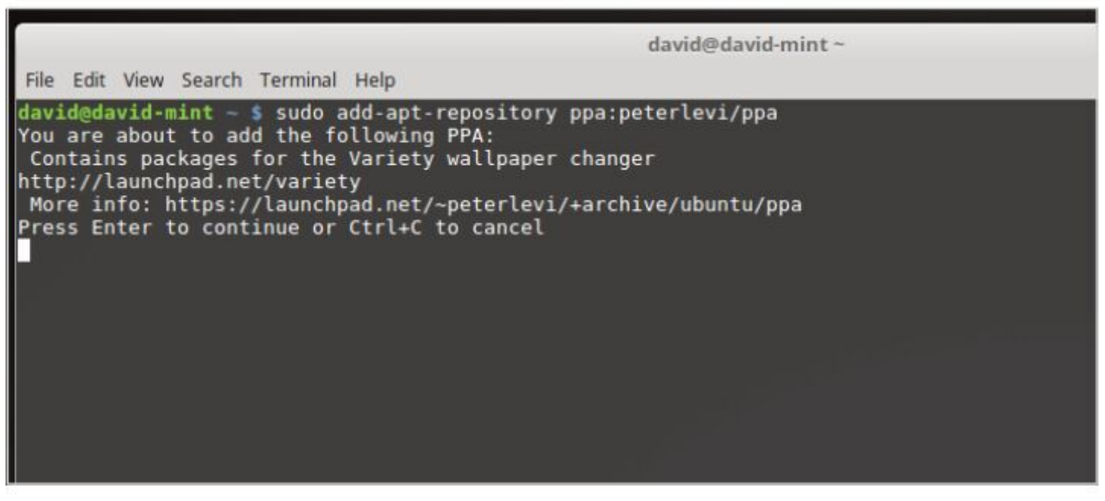
STEP 2 Just as you've seen, sudo apt-get update/upgrade and so on are designed to upgrade the software that's already installed on the system. How do you install more apps though? It just so happens that it's extraordinarily simple. First you need an app to install, so let's use Stella again. Enter: `sudo apt-get install stella`.



STEP 3 You need to enter y to confirm the installation, which takes up around 5.5MB of storage in the system. Once Stella is installed, you can see again that Mint has automatically created the Games category in the Menu as well as the app shortcut.



STEP 4 Sometimes, when installing software, you need to add the app's Repository. The repository, or repo, is simply the remote server location where the software is held, along with all its dependencies (the vital libraries and such it needs to function). Start by typing in this: `sudo add-apt-repository ppa:peterlevi/ppa`. Press Enter when asked to and add the PPA (Personal Package Archive).



STEP 5 This adds the repo for the app Variety Wallpaper Changer, an Ubuntu-based app that works in Mint and changes the wallpaper automatically. Now that the repo is added, enter: `sudo apt-get update`, to update the new information and add the contents of the repo to the package database.

```
File  Edit  View  Search  Terminal  Help
david@david-mint ~ $ sudo add-apt-repository ppa:peterlevi/ppa
You are about to add the following PPA:
 Contains packages for the Variety wallpaper changer
http://launchpad.net/variety
 More info: https://launchpad.net/~peterlevi/+archive/ubuntu/ppa
Press Enter to continue or Ctrl+C to cancel

Executing: /tmp/tmp.ug5iHzLK12/gpg.1.sh --keyserver
hkp://keyserver.ubuntu.com:80
--recv-keys
A546BE4F
gpg: requesting key A546BE4F from hkp server keyserver.ubuntu.com
gpg: key A546BE4F: public key "Launchpad PPA for Peter Levi" imported
gpg: Total number processed: 1
gpg:               imported: 1  (RSA: 1)
david@david-mint ~ $ sudo apt-get update
Get:1 http://ppa.launchpad.net/peterlevi/ppa/ubuntu xenial InRelease [17.5 kB]
Get:2 http://security.ubuntu.com/ubuntu xenial-security InRelease [102 kB]
Hit:3 http://archive.canonical.com/ubuntu xenial InRelease
Hit:4 http://archive.ubuntu.com/ubuntu xenial InRelease
Ign:5 http://www.mirrorservice.org/sites/packages.linuxmint.com/packages serena InRelease
Get:6 http://archive.ubuntu.com/ubuntu xenial-updates InRelease [102 kB]
Hit:7 http://www.mirrorservice.org/sites/packages.linuxmint.com/packages serena Release
Get:8 http://archive.ubuntu.com/ubuntu xenial-backports InRelease [102 kB]
Get:9 http://ppa.launchpad.net/peterlevi/ppa/ubuntu xenial/main Sources [824 B]
Get:10 http://ppa.launchpad.net/peterlevi/ppa/ubuntu xenial/main amd64 Packages [844 B]
Get:11 http://ppa.launchpad.net/peterlevi/ppa/ubuntu xenial/main i386 Packages [844 B]
Get:13 http://ppa.launchpad.net/peterlevi/ppa/ubuntu xenial/main Translation-en [548 B]
Fetched 327 kB in 0s (712 kB/s)
Reading package lists... Done
```

STEP 6 Now to install Variety, enter: `sudo apt-get install variety`. Press y to confirm and accept the installation, and to continue with the install. Once installed, you can type `variety` into the Terminal to run the app.

```
Selecting previously unselected package libgexiv2-2:amd64.
(Reading database ... 365690 files and directories currently installed.)
Preparing to unpack .../libgexiv2-2_0.10.8-1_amd64.deb ...
Unpacking libgexiv2-2:amd64 (0.10.8-1) ...
Selecting previously unselected package gir1.2-gexiv2-0.10:amd64.
Preparing to unpack .../gir1.2-gexiv2-0.10_0.10.8-1_amd64.deb ...
Unpacking gir1.2-gexiv2-0.10:amd64 (0.10.8-1) ...
Selecting previously unselected package variety.
Preparing to unpack .../variety_0.7.0~git201809151602.8eff230~ppa768~ubuntu18.04
.1_all.deb ...
Unpacking variety (0.7.0~git201809151602.8eff230~ppa768~ubuntu18.04.1) ...
Processing triggers for mime-support (3.60ubuntu1) ...
Processing triggers for desktop-file-utils (0.23+linuxmint4) ...
Setting up libgexiv2-2:amd64 (0.10.8-1) ...
Processing triggers for libc-bin (2.27-3ubuntu1) ...
Processing triggers for man-db (2.8.3-2ubuntu0.1) ...
Processing triggers for gnome-menus (3.13.3-11ubuntu1.1) ...
Processing triggers for hicolor-icon-theme (0.17-2) ...
Setting up gir1.2-gexiv2-0.10:amd64 (0.10.8-1) ...
Setting up variety (0.7.0~git201809151602.8eff230~ppa768~ubuntu18.04.1) ...
david@david-Mint:~$
david@david-Mint:~$
david@david-Mint:~$ variety
```

REMOVING APPS

In addition to installing apps, the apt command can also be used to remove any apps and helps keep the system tidy and free up resources.

STEP 1 To uninstall, or remove, the Variety app enter the following: `sudo apt-get remove variety`. Enter y to continue with the uninstall of the app; notice also that you're informed of how much space you're freeing up on the hard drive as a result of removing the app.

```
david@david-mint ~
File  Edit  View  Search  Terminal  Help
david@david-mint ~ $ sudo apt-get remove variety
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer requi
  libboost-python1.58.0 python-pyexiv2 variety-slideshow
Use 'sudo apt autoremove' to remove them.
The following packages will be REMOVED
  variety
0 to upgrade, 0 to newly install, 1 to remove and 0 not to upgrade.
After this operation, 2,442 kB disk space will be freed.
Do you want to continue? [Y/n]
```

STEP 2 While the 'apt-get remove' command uninstalls an app, it doesn't get rid of the extra clutter that comes with an app, such as configuration and library files. To completely remove the clutter, enter: `sudo apt-get purge variety`.

```
david@david-mint ~
File  Edit  View  Search  Terminal  Help
david@david-mint ~ $ sudo apt-get purge variety
Reading package lists... Done
Building dependency tree
Reading state information... Done
Package 'variety' is not installed, so not removed
The following packages were automatically installed and are no longer require
  libboost-python1.58.0 python-pyexiv2 variety-slideshow
Use 'sudo apt autoremove' to remove them.
0 to upgrade, 0 to newly install, 0 to remove and 0 not to upgrade.
david@david-mint ~ $
```

STEP 3 When you remove apps from the system you're be informed that some packages that were automatically installed are no longer required. You already saw in the previous tutorial, that you can tidy things up with the following command: `sudo apt-get autoremove`, followed by pressing y to accept the process.

```
david@david-mint ~
File  Edit  View  Search  Terminal  Help
david@david-mint ~ $ sudo apt-get autoremove
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages will be REMOVED
  libboost-python1.58.0 python-pyexiv2 variety-slideshow
0 to upgrade, 0 to newly install, 3 to remove and 0 not to upgrade.
After this operation, 1,160 kB disk space will be freed.
Do you want to continue? [Y/n]
```

STEP 4 Finally, to tidy up all the non-used packages in the system, and to remove elements that the autoremove command didn't, you can enter: `sudo apt-get autoclean`. These last few steps are vital for keeping your Linux Mint setup in good working order and to trim off the unnecessary excess caused by installations and upgrades.

```
david@david-mint ~
File  Edit  View  Search  Terminal  Help
david@david-mint ~ $ sudo apt-get autoclean
Reading package lists... Done
Building dependency tree
Reading state information... Done
david@david-mint ~ $
```

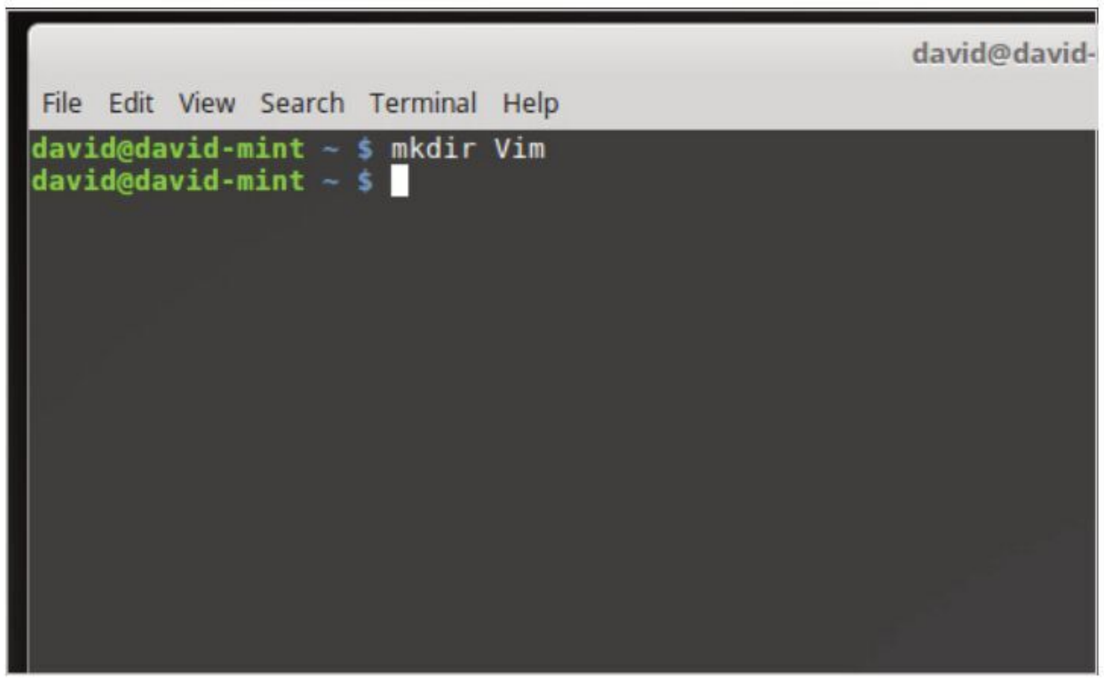
Install Apps via the Terminal – Part 2

Most of the time you'll get to install apps from the Terminal using the standard apt-get command. However, sometimes an app demands a little more work. This means installing an app from its source code, which isn't as scary as it first sounds.

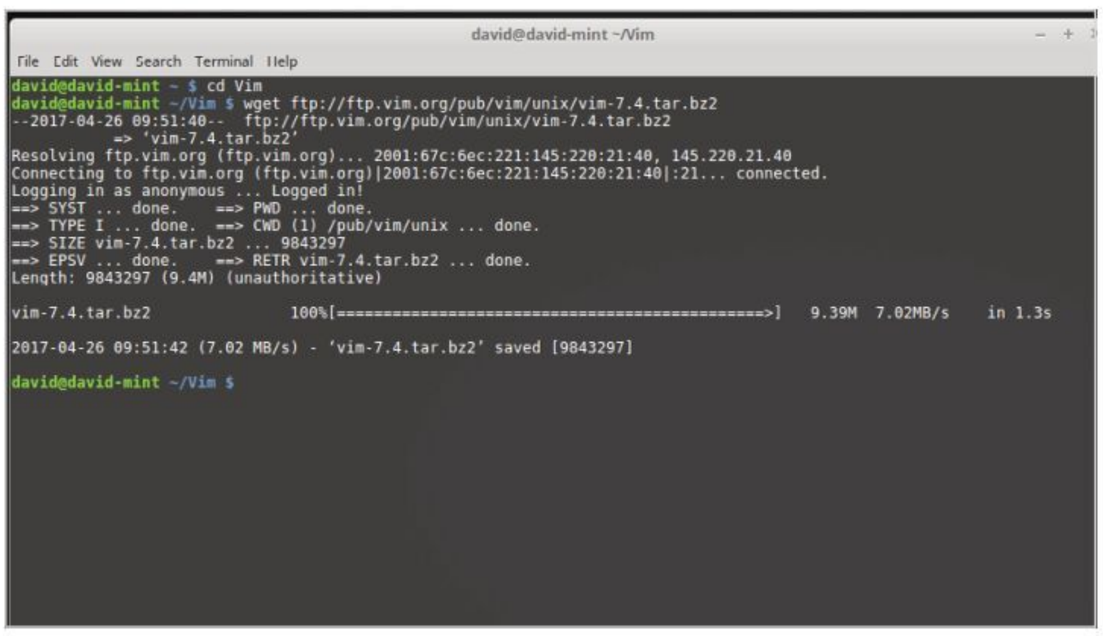
FROM THE SOURCE

The commands you'll need to become familiar with here are Configure, Make and Install. You'll find a lot of apps use installing from source, so it's certainly a skill worth investing time in.

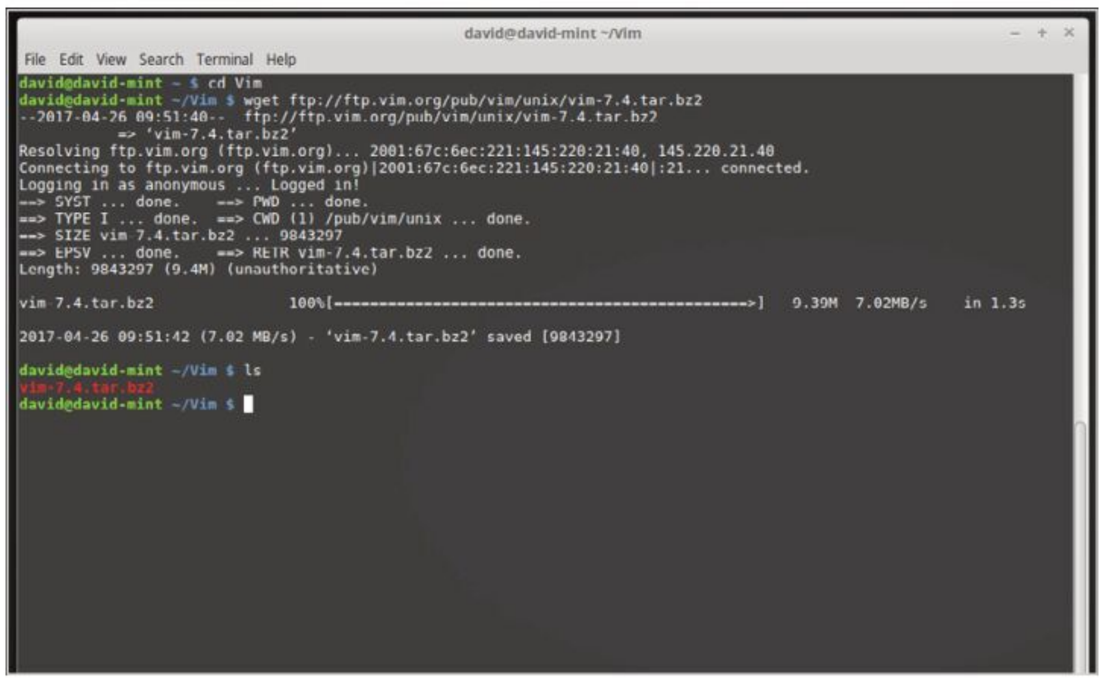
STEP 1 Source code files for Linux usually come in the form of .TAR.GZ or .TAR.BZ2. Both are compressed files holding all the core files needed to 'make' the app. Start off this tutorial by creating a new folder in Home: `mkdir Vim`.



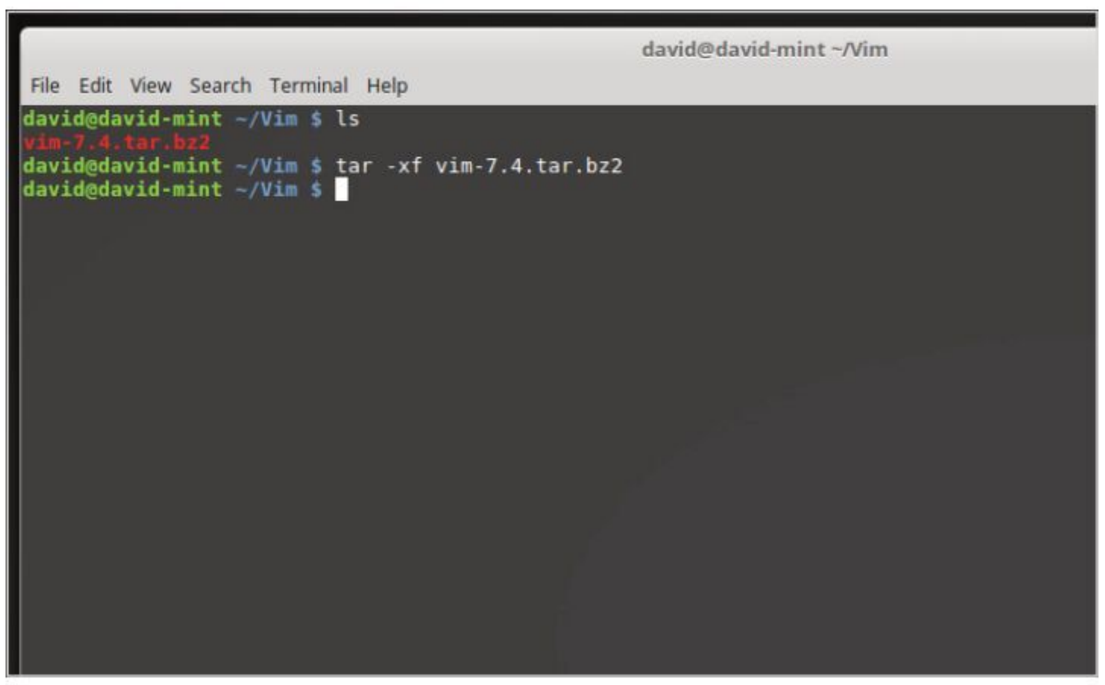
STEP 2 Vim, by the way, is an advanced text editor which we'll use as an example to install. Enter the new folder, cd Vim, then from within the new Vim folder, enter the following command into the Terminal: `wget ftp://ftp.vim.org/pub/vim/unix/vim-7.4.tar.bz2`.



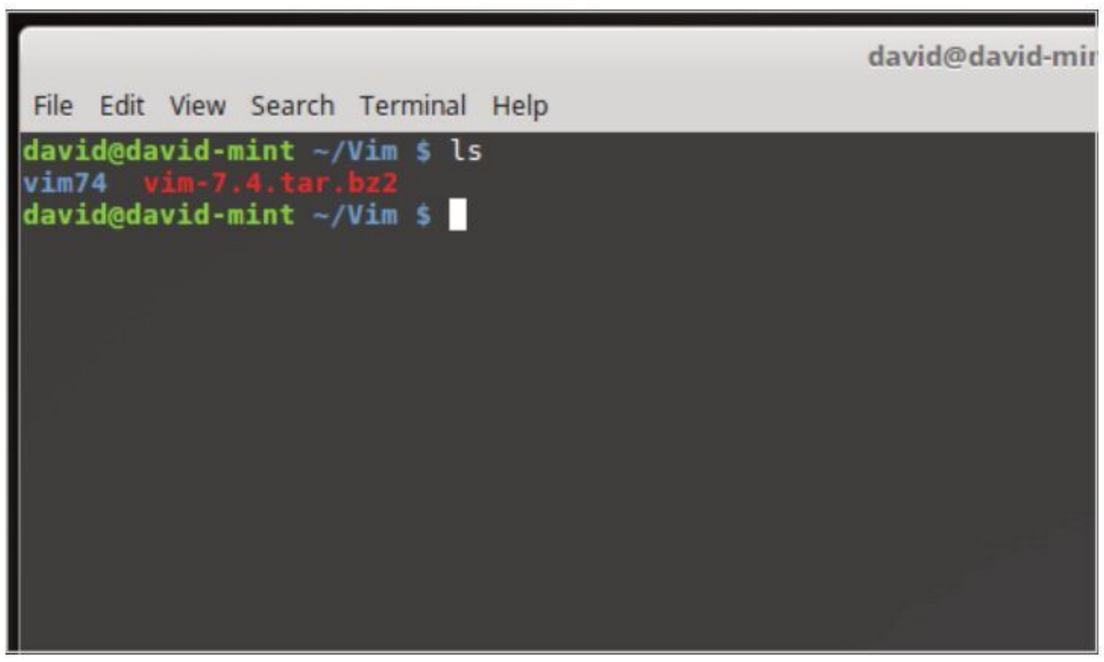
STEP 3 The `wget` command retrieves content from the internet, in this case the .BZ2 file for Vim. To check the file was downloaded successfully, enter: ls. According to Mint's file system colour key, the compressed file should be displayed in red.



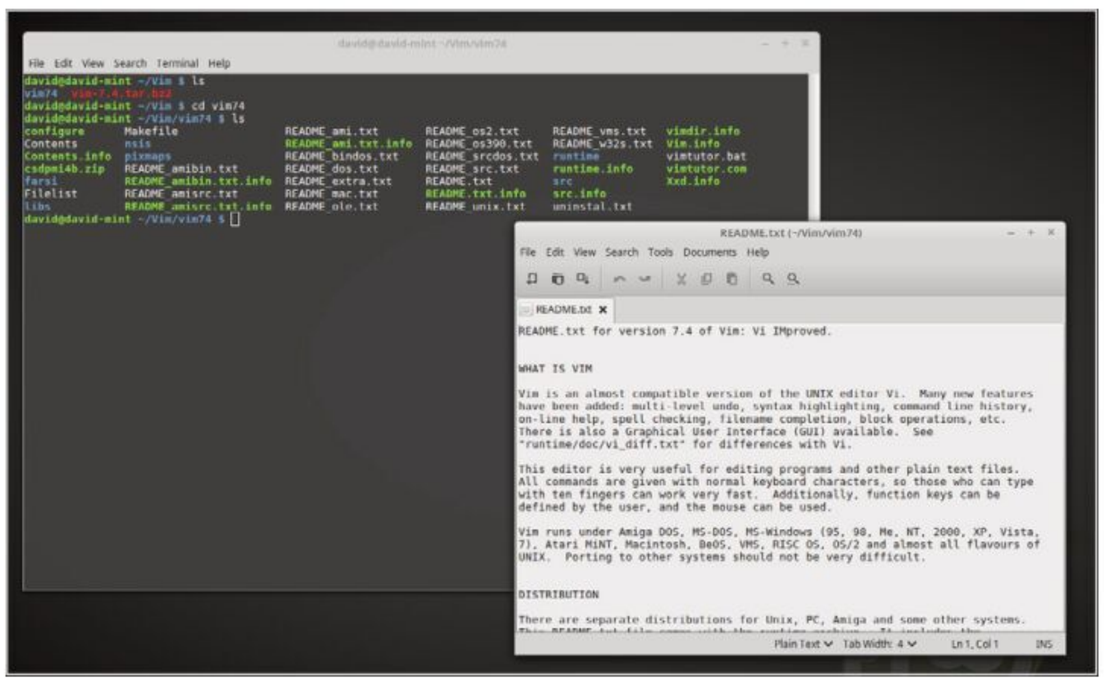
STEP 4 We need to uncompress the contents of the file now, so enter: `tar -xf vim-7.4.tar.bz2` into the Terminal. Note: you can type in `tar -xf v` and press the `Tab` key to auto-fill the remaining file name.



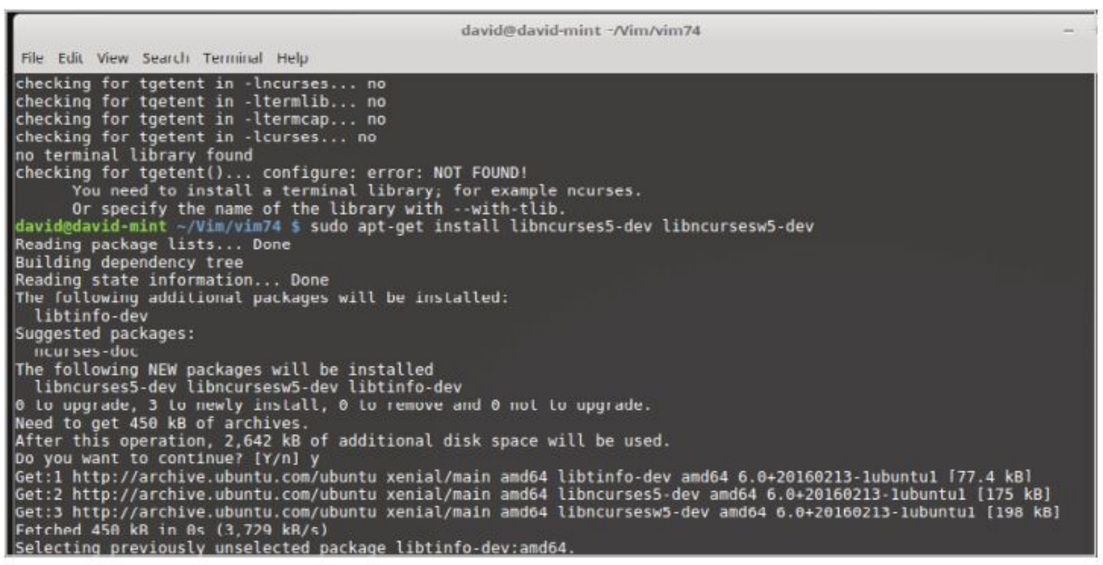
STEP 5 If you enter `ls` again, you'll notice that a new folder has been created: `vim74`; in light blue text representing a folder in Mint. It's always handy to create root folders for the main app, then as you upgrade apps through this method the individual versions will each have their own folder.



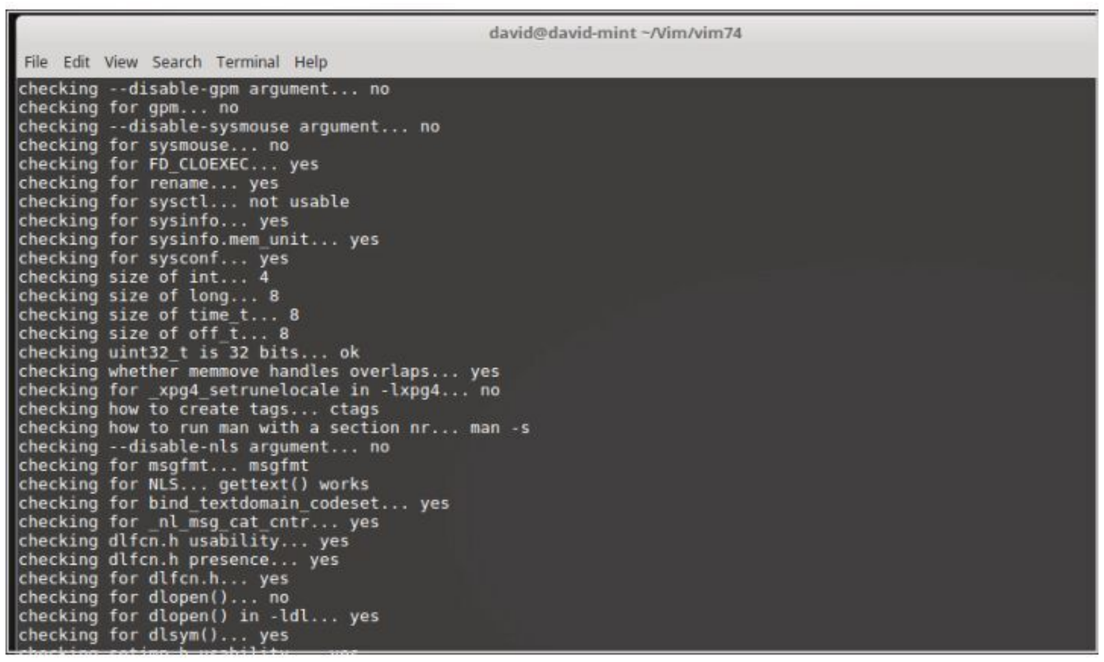
STEP 6 Type in `cd vim74` to enter the folder, and `ls` again to view its contents. There will likely be a fair number of files present; most are the app's core files, while others will be labelled **README** or **INSTALL**. It's always wise to read these files first as they provide valuable information regarding the installation.



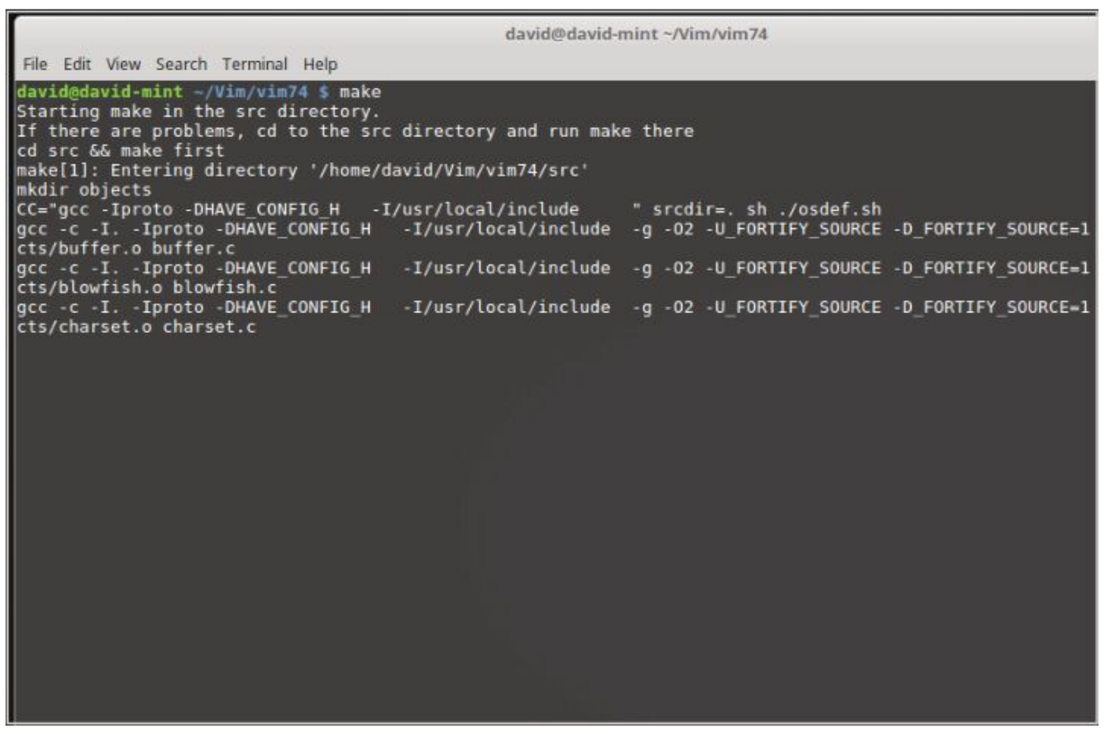
STEP 7 The first part of the installation requires you to enter `./configure`. The ./configure command will check your system for any missing dependencies associated with the app. If you received an error regarding a C Compiler, then enter: `sudo apt-get install build-essential`. The third-party app Ncurses was recorded as missing. We need to install that with: `sudo apt-get install libncurses5-dev libncursesw5-dev`.



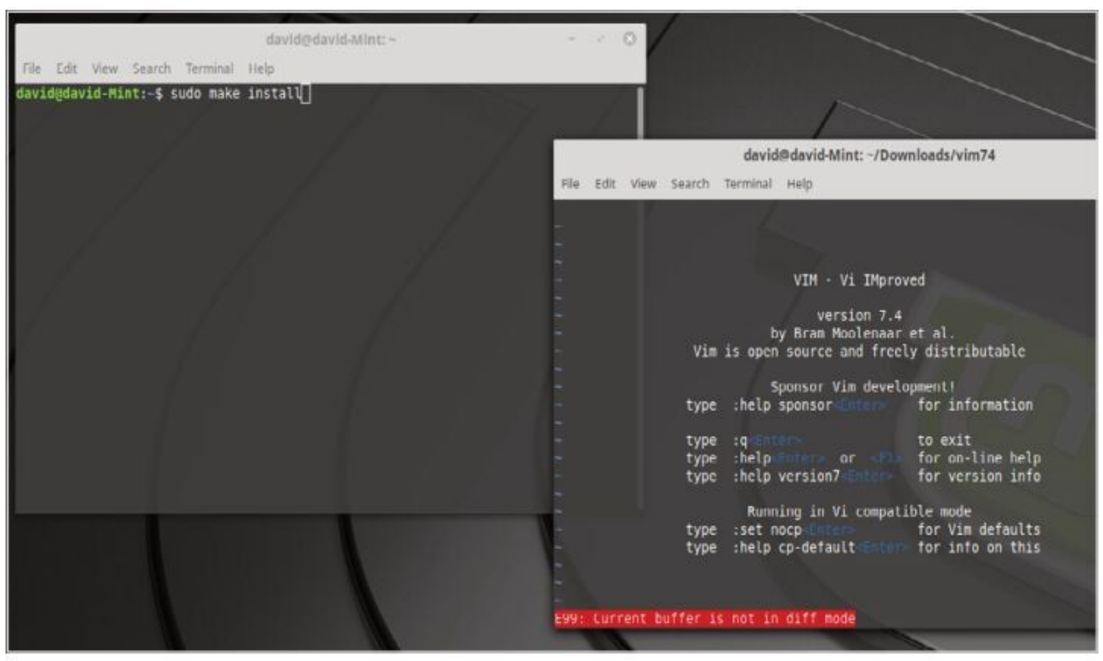
STEP 8 You may need to keep installing new dependencies, depending on the app. After each new dependency is installed, re-run `./configure` and when it doesn't report back with an error you can continue to the next stage of the installation. Note: you may need to search online for some error messages.



STEP 9 With a successful `./configure`, the system will create a **Makefile**. This needs to be 'made' by entering: `make` into the Terminal. This may take a while, depending on the size of the app.



STEP 10 Finally, you need to enter: `sudo make install` into the Terminal. This will install the app, and make it ready for use in the system. When complete you can execute the app, in this case by entering vim into the Terminal or searching for it via the Dash.



```
Kernel command line: block2mtd.block2mtd=/dev/hda2,131072,rootfs root=/dev/mtdbl
ock0 rootfstype=jffs2 init=/etc/preinit  noinitrd console=tty0 console=ttyS0,384
00n8 reboot=bios
Found and enabled local APIC!
Enabling fast FPU save and restore... done.
Enabling unmasked SIMD FPU exception support... done.
Initializing CPU#0
PID hash table entries: 32 (order: 5, 128 bytes)
Detected 1991.657 MHz processor.
Console: colour VGA+ 80x25
console [tty0] enabled
console [ttyS0] enabled
Dentry cache hash table entries: 1024 (order: 0, 4096 bytes)
Inode-cache hash table entries: 1024 (order: 0, 4096 bytes)
Memory: 5112k/8128k available (1497k kernel code, 2624k reserved, 597k data, 196
k init, 0k highmem)
virtual kernel memory layout:
    fixmap  : 0xfffb9000 - 0xfffff000   ( 280 kB)
    vmalloc : 0xc1000000 - 0xfffb7000   (1007 MB)
    lowmem  : 0xc0000000 - 0xc07f0000   (   7 MB)
      .init : 0xc0313000 - 0xc0344000   ( 196 kB)
      .data : 0xc027653c - 0xc030bcfc   ( 597 kB)
      .text : 0xc0100000 - 0xc027653c   (1497 kB)
Checking if this processor honours the WP bit even in supervisor mode...Ok.
Calibrating delay using timer specific routine.. 4047.64 BogoMIPS (lpj=20238210)
```

Did You Know...

Linux Kernel **0.01**

```
/*
 *  console.c
 *
 * This module implements the console io functions
 *  'void con_init(void)'
 *  'void con_write(struct tty_queue * queue)'
 * Hopefully this will be a rather complete VT102 implementation.
 *
 */

/*
 *  NOTE!!! We sometimes disable and enable interrupts for a short while
 * (to put a word in video IO), but this will work even for keyboard
 * interrupts. We know interrupts aren't enabled when getting a keyboard
 * interrupt, as we use trap-gates. Hopefully all is well.
 */

#include <linux/sched.h>
#include <linux/tty.h>
#include <asm/io.h>
#include <asm/system.h>

#define SCREEN_START 0xb8000
#define SCREEN_END   0xc0000
#define LINES 25
#define COLUMNS 80
#define NPAR 16

extern void keyboard_interrupt(void);

static unsigned long origin=SCREEN_START;
static unsigned long scr_end=SCREEN_START+LINES*COLUMNS*2;
static unsigned long pos;
static unsigned long x,y;
static unsigned long top=0,bottom=LINES;
static unsigned long lines=LINES,columns=COLUMNS;
static unsigned long state=0;
static unsigned long npar,par[NPAR];
static unsigned long ques=0;
static unsigned char attr=0x07;
```

DID YOU KNOW... that the first Linux kernel, as programmed by Linus Torvalds, only occupied 65KB of memory? And that since version 0.01, the Linux kernel now boasts over 18 million lines of code. Not bad for a university project that now runs the New York Stock Exchange, and powers the US Department of Defence's fleet of nuclear submarines.

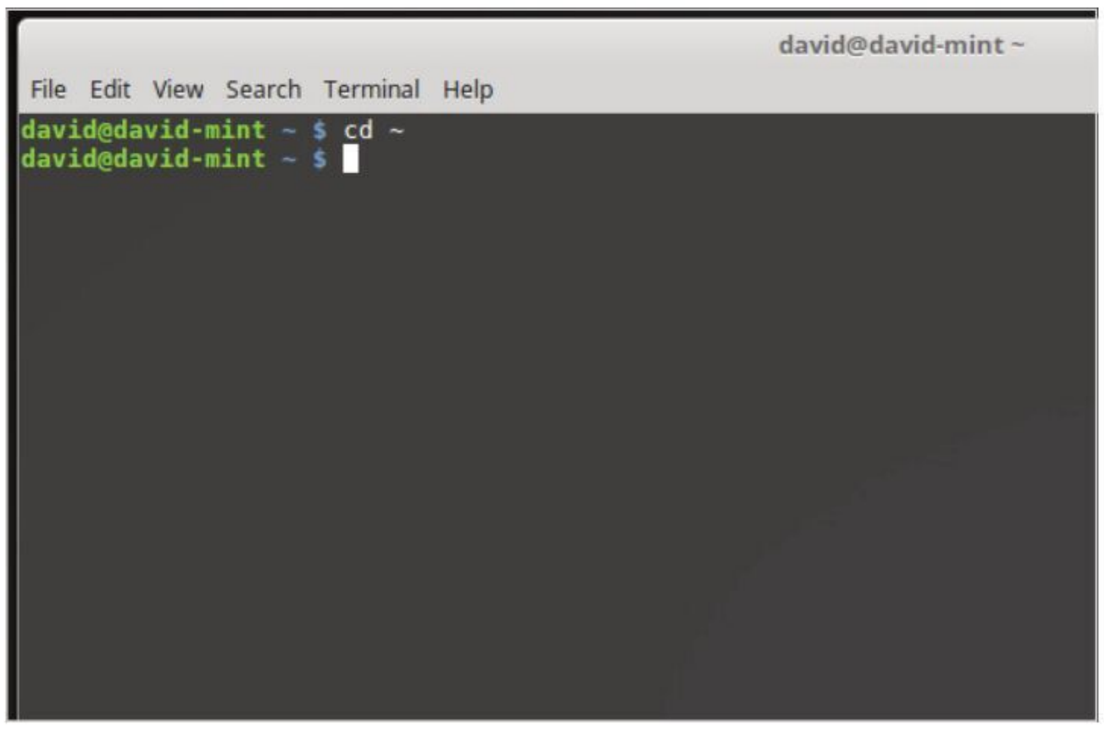
Creating a File Using the Terminal

Using the Terminal, you're able to create folders, files and even execute Linux Mint apps. In truth, if you didn't have the GUI at hand, you could still accomplish the same from the Terminal.

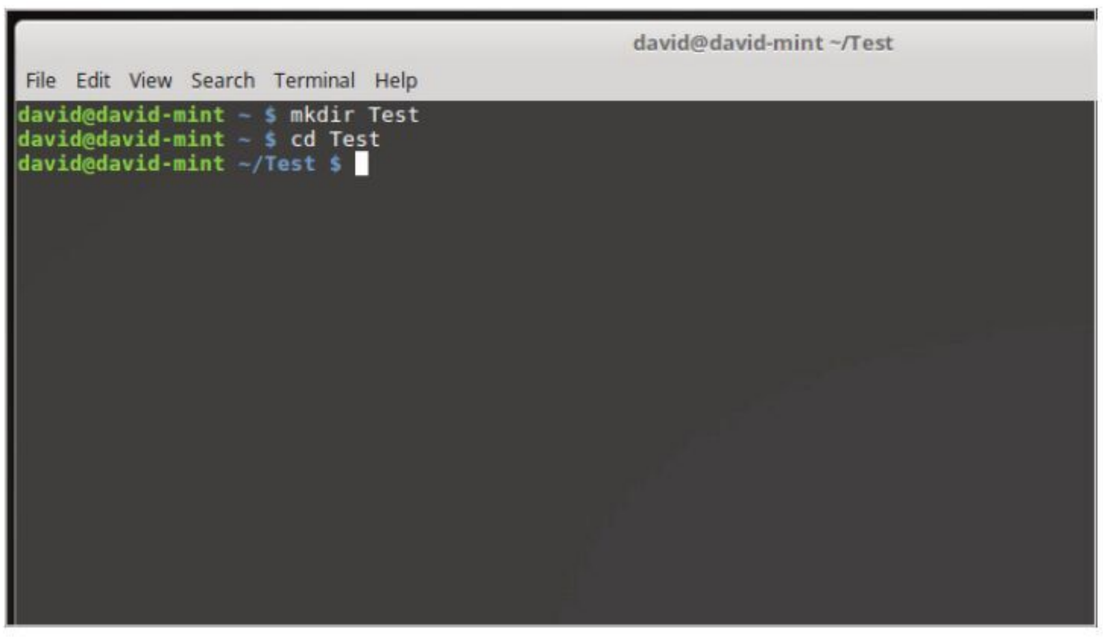
MORE TERMINAL WORK

Creating content using the Terminal isn't quite as strenuous as it may first appear. Yes, the Terminal can look a daunting place for the newcomer, but once mastered it's really quite intuitive.

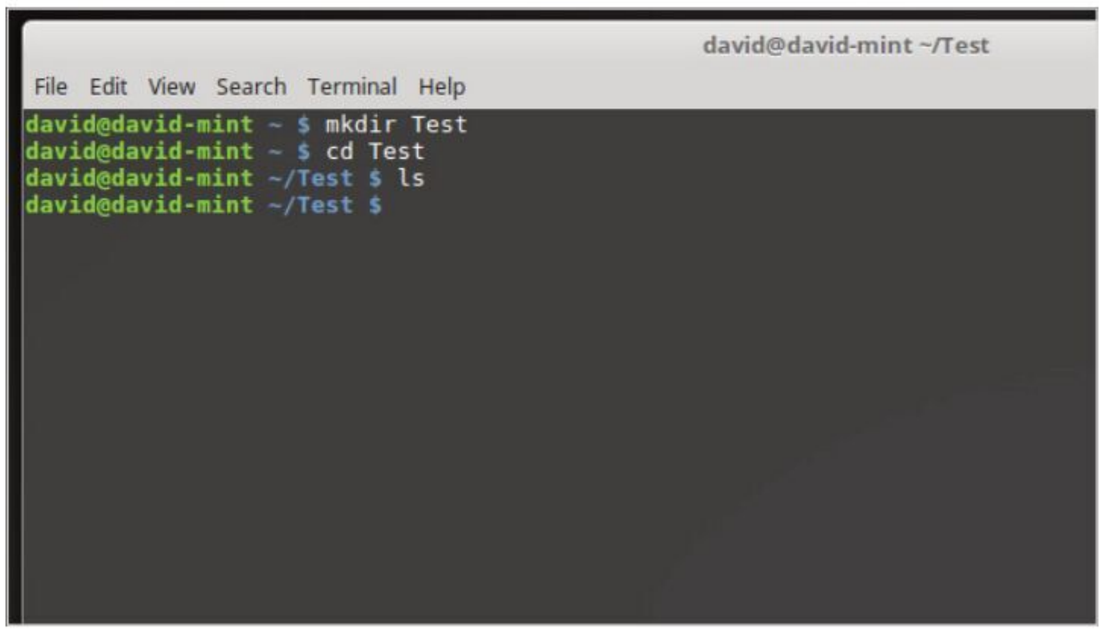
STEP 1 Open up the Terminal, and make sure you're in the Home folder. If not use the **cd ~** command to return you to the Home folder from wherever you're currently located.



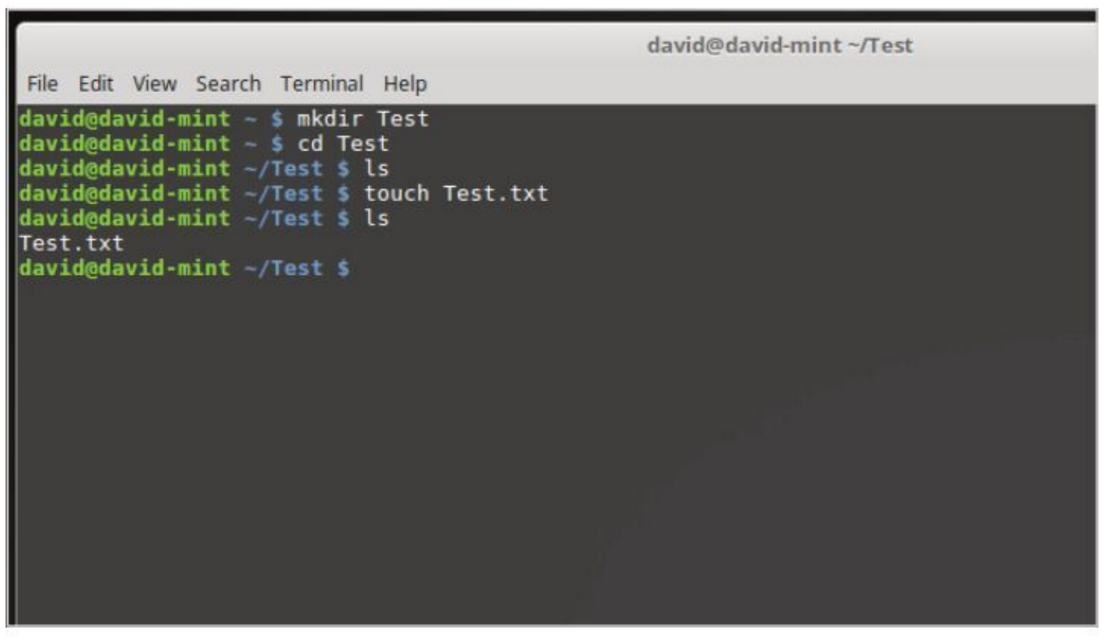
STEP 2 Let's start by creating a new folder within Home, and call it **Test**. The command you'll need is `mkdir` Test. Press **Enter** to create the folder when you've typed in the command, then `cd Test` and press **Enter**. This will **C**hange **D**irectory (hence **CD**) to the newly created Test folder.



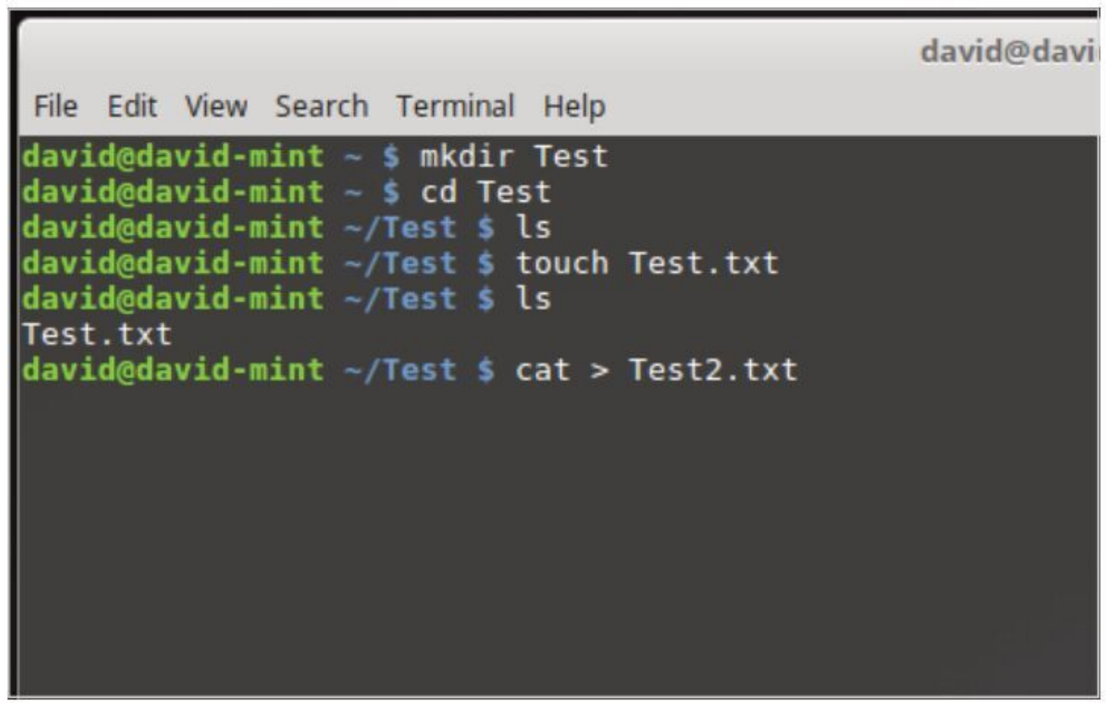
STEP 3 Your command line should change to the Test folder and if you enter `ls` (List folder contents) there'll be nothing within the folder; as you've just created it. The `mkdir` command is fairly self explanatory: Make Directory followed by the name of your choosing.



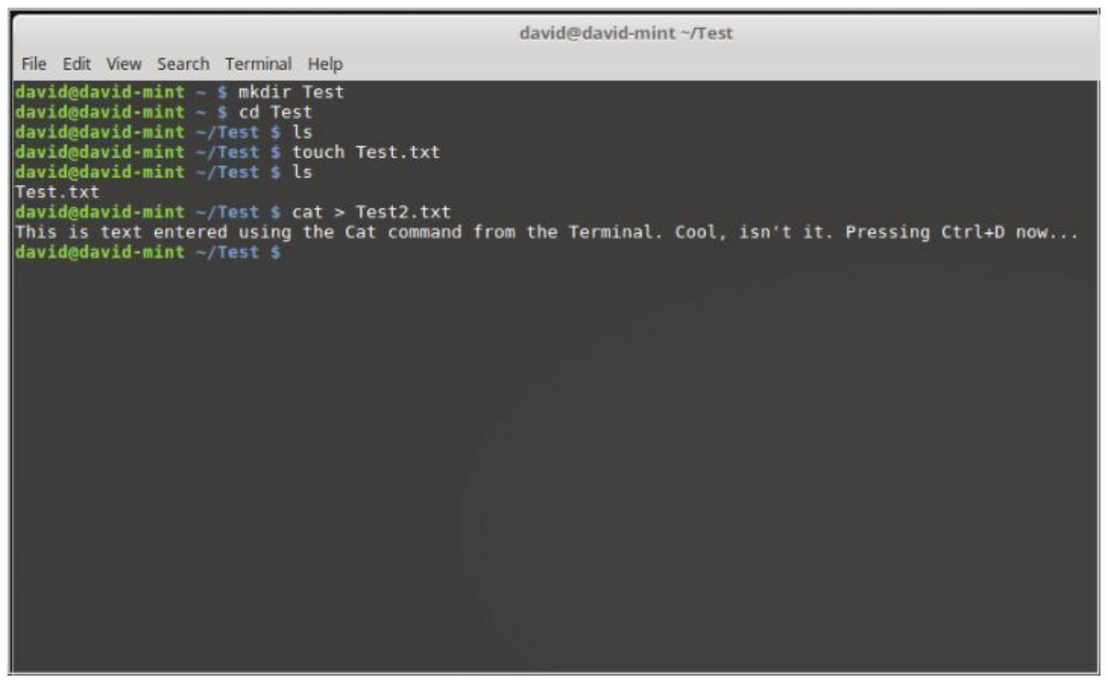
STEP 4 To create an empty text file, called **Test.txt**, enter in the Terminal: `touch Test.txt`. You can then use `ls` to view the new file in the folder. Touch is a standard Linux command that allows the creation of files without the need to open a text editor, save the file, then close the editor.



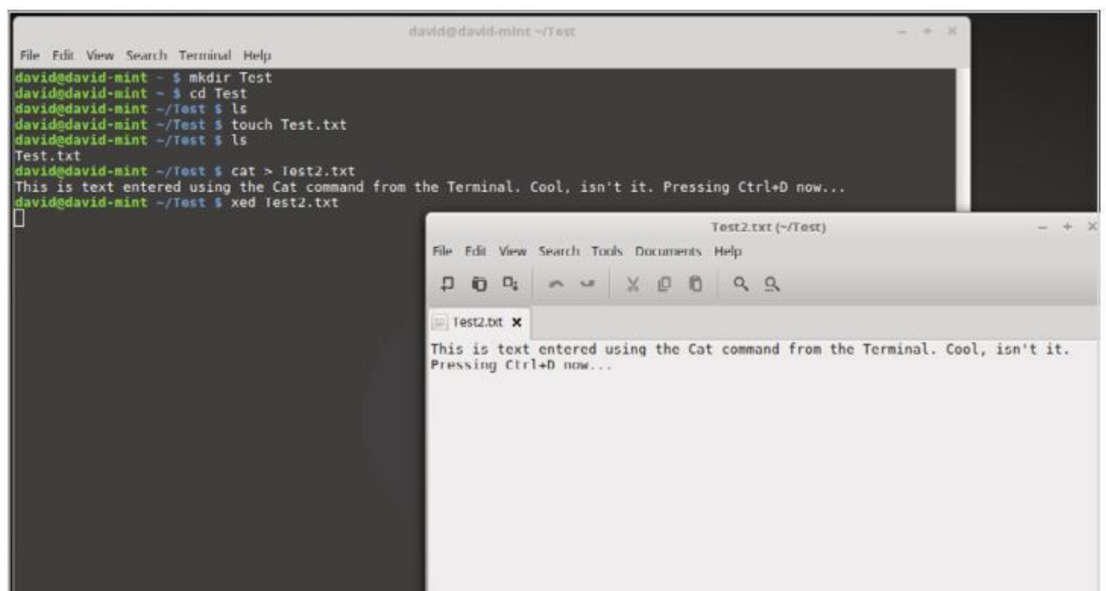
STEP 5 Let's say you now wanted to create a text file, we'll call it Test2.txt, complete with some content. To do so, enter: `cat > Test2.txt`. This will create the file **Test2.txt** and put the Terminal into an editing mode.



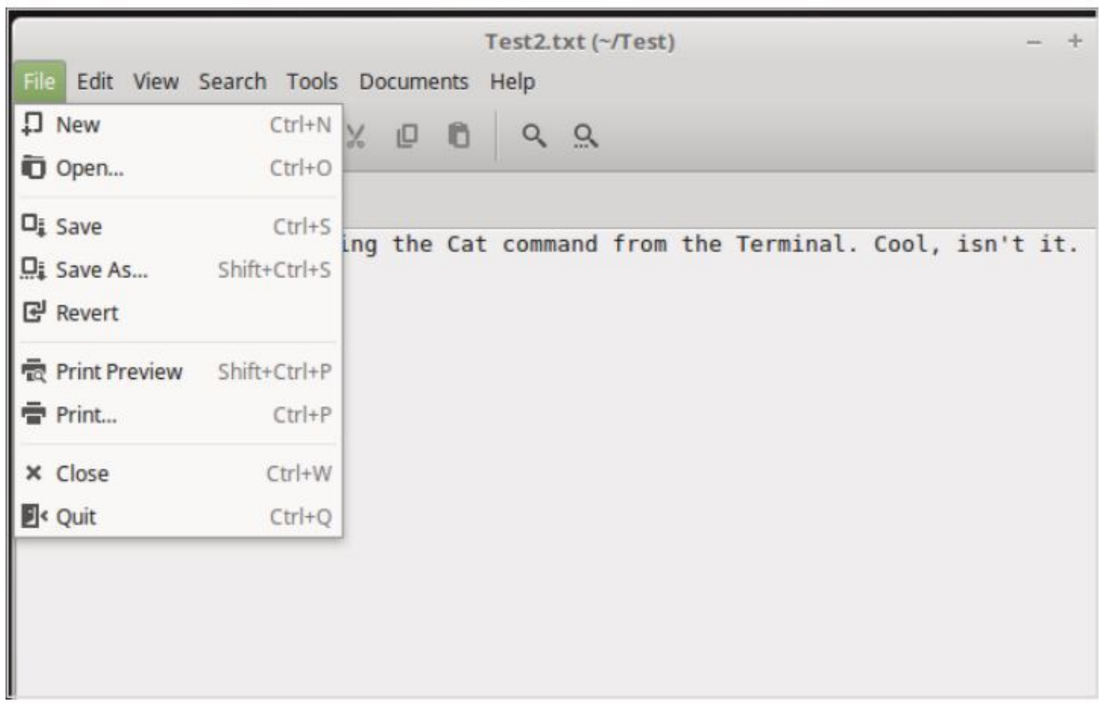
STEP 6 You'll notice that the cursor is flashing below the `cat > Test2.txt` command, without the usual prompt. This editing mode will allow you enter the text that the file will contain. Enter some text, then press **Ctrl+D** to exit and write the contents to the file.



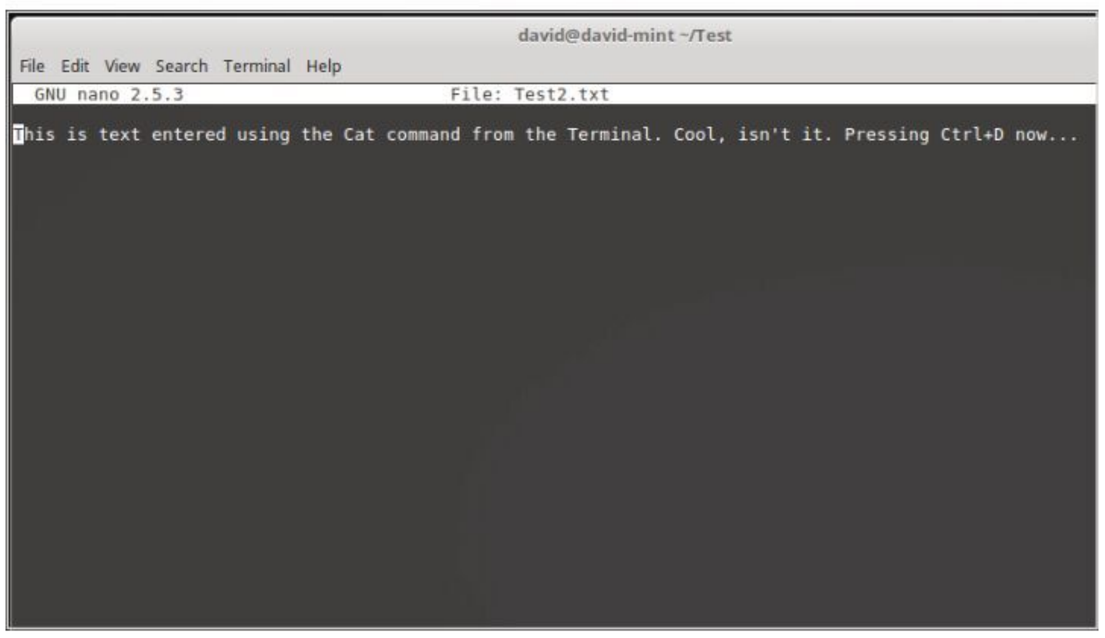
STEP 7 Of course you don't always have to use the Terminal to enter text into a file. Mint comes with a text editor called **Xed**, which is similar to Windows' Notepad. To view the previously created file in Xed, type into the Terminal: `xed Test2.txt`, and press **Enter**.



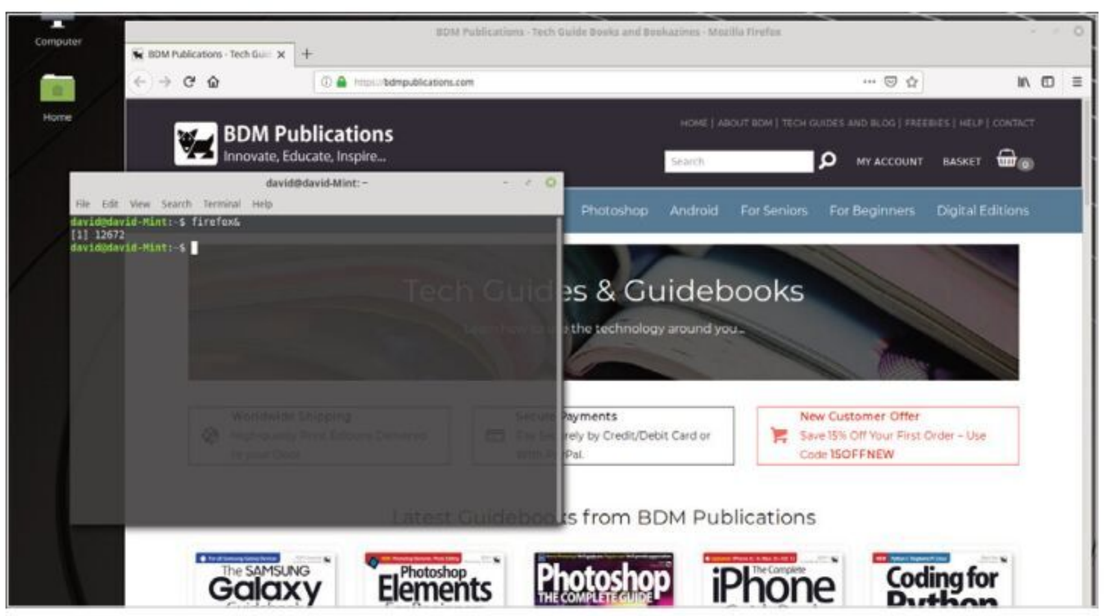
STEP 8 Xed is a GUI app, and you can enter text and save the file, or any file, accordingly by using the app's main window, and the **File > Save**, or **File > Save As** functions from its top menu bar options.



STEP 9 If, however, you prefer to remain working in the Terminal to edit/save/create files, you can use **Nano**. Nano is a simple Terminal-based text editor. To try it with the example, enter: `nano Test2.txt`. There's a menu along the bottom of the screen. To exit and save any content in Nano, press **Ctrl+X** and follow the on-screen instructions.



STEP 10 We've used the Terminal to launch a Mint app, Xed, but any app can be launched from within the Terminal. For example, try: **firefox**, and press **Enter**. Close Firefox to return to the Terminal. Providing you know the name of the app, it can run from the Terminal. Additionally, entering `firefox&` opens Firefox, AND lets you still use the Terminal.



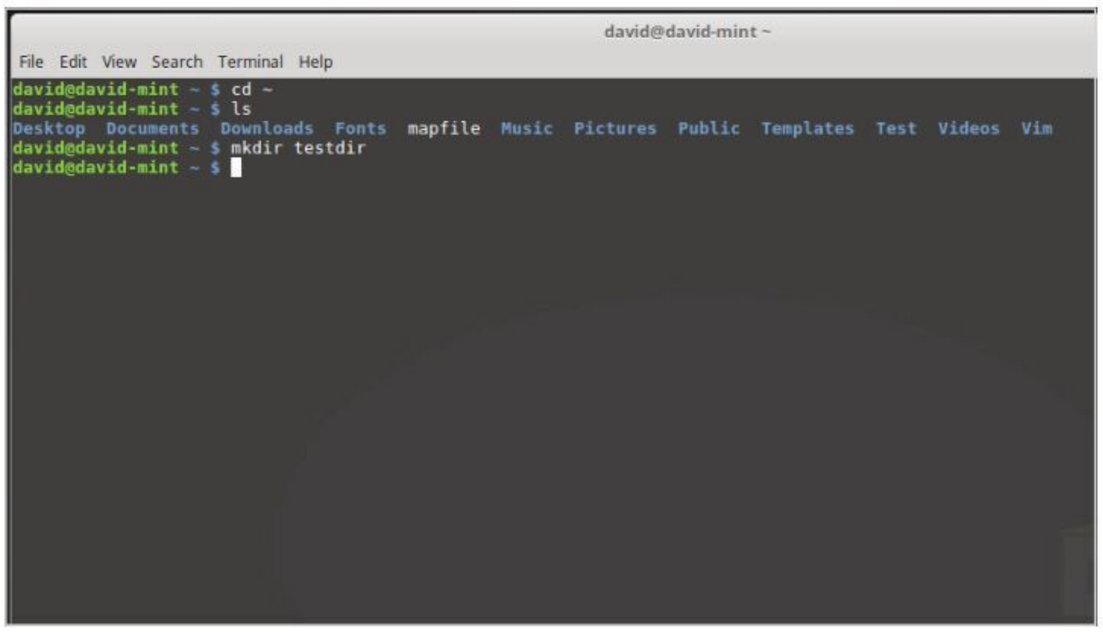
Creating and Removing Directories

As with creating files in the Terminal, you can also create and delete directories, or folders if you prefer. Directories form the structure of your file system, without logical directories the filing system would be in utter chaos.

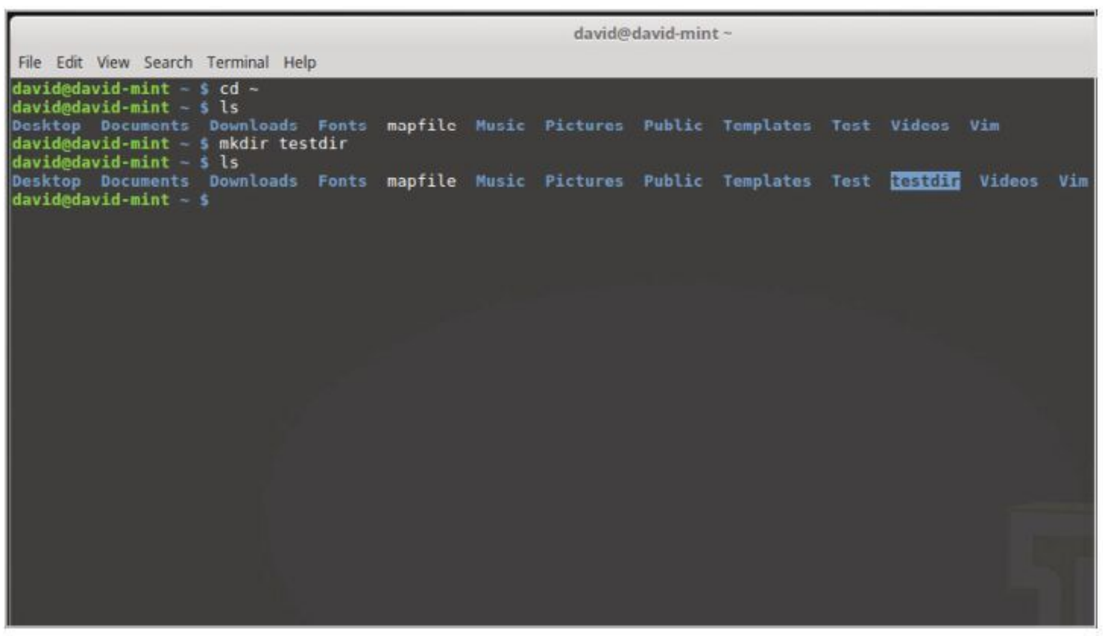
MANAGING FOLDERS

Learning how to create and delete folders in the Terminal is an important Mint, and indeed Linux overall, skill to master. Here's the basics for you to try out.

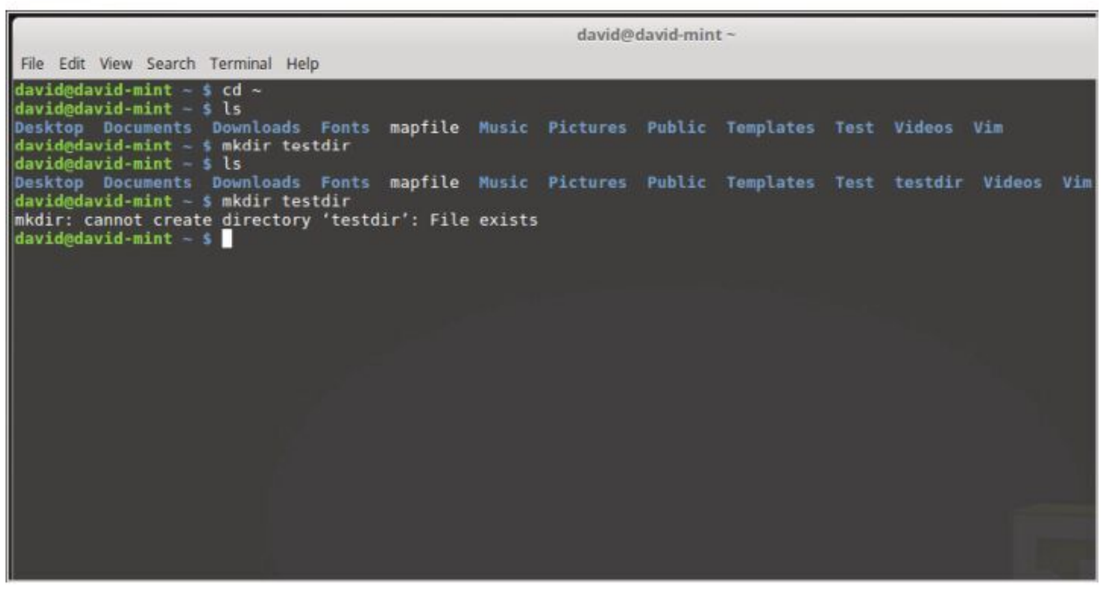
STEP 1 With the Terminal open enter cd ~ to make sure you're in your own Home directory. Now enter ls to view the current folders you have housed in the Home directory. You'll notice that folders are labelled in Mint in cyan (light blue). Let's start by creating a new directory. Enter: `mkdir testdir`.



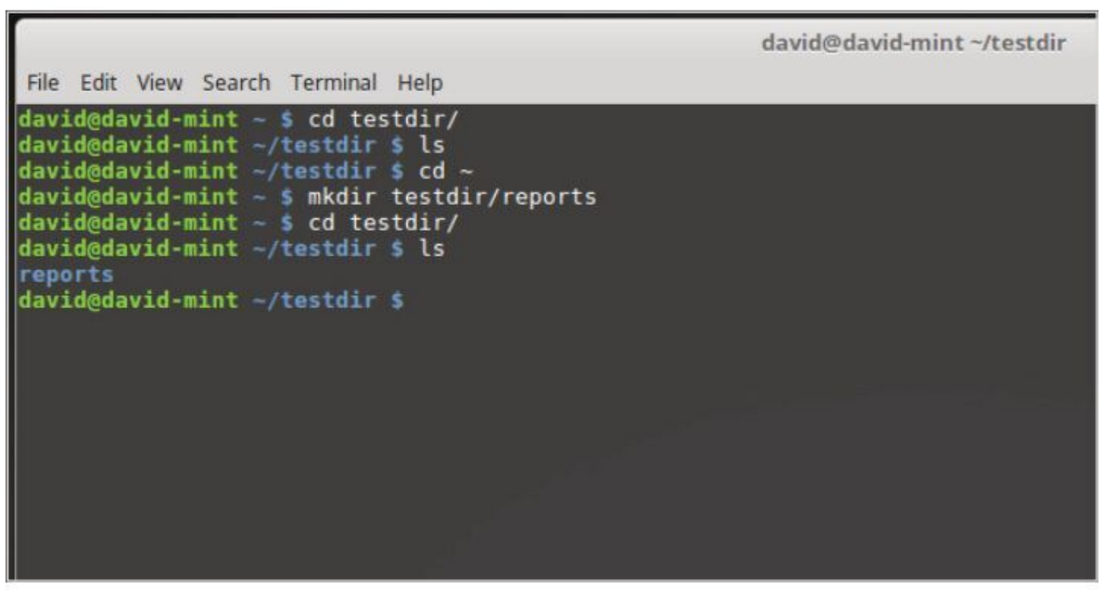
STEP 2 If you now enter ls again, you'll see that the new directory, testdir, has been created alongside the other directories in the Home area. Obviously the command `mkdir` is what creates the directory, and no doubt you've already guessed it stands for Make Directory.



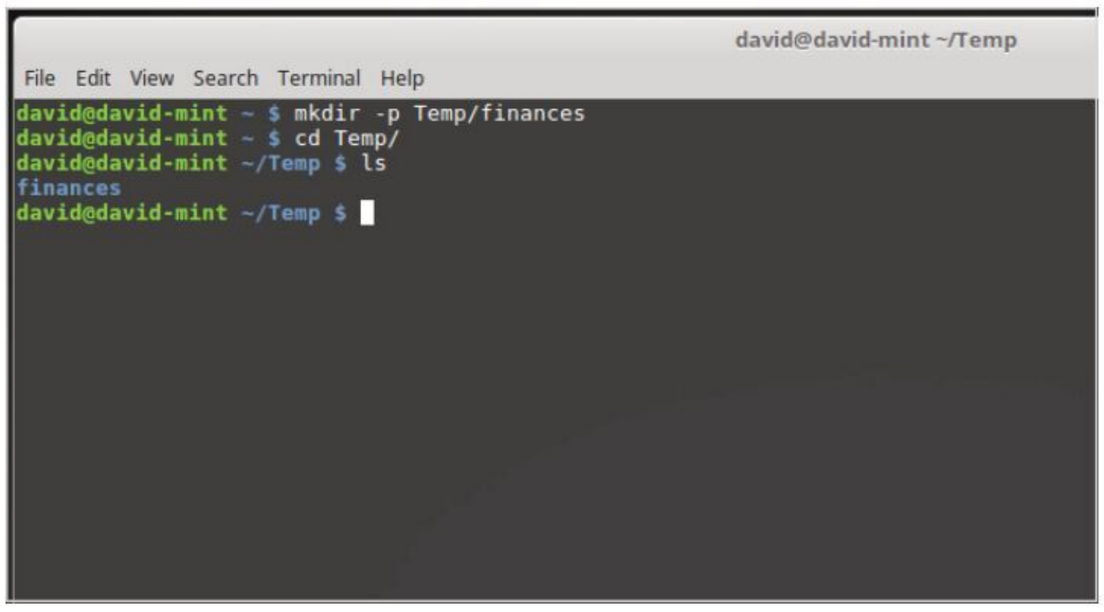
STEP 3 If you were to enter the command again, `mkdir testdir`, you'll receive a message stating: `mkdir: cannot create directory 'testdir': File exists`. It goes without saying then, that you're only able to have one uniquely named directory within the current directory. However, as Linux is case-sensitive, you can have Testdir, TestDir, testDir and so on.



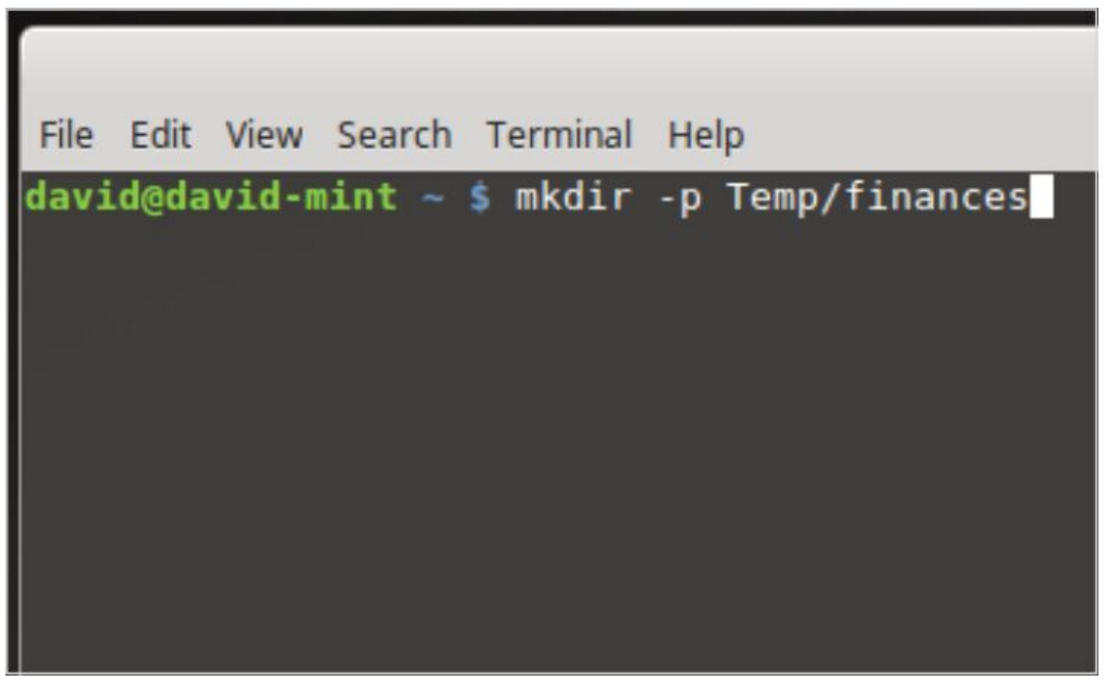
STEP 4 You can create directories within directories you've already created. For example, enter the testdir directory with `cd testdir/` followed by ls to list the folder structure. Naturally there's nothing present, as you've just created the directory. Now drop back to Home with **cd ~**, and enter `mkdir testdir/reports`. Go back to the testdir, `cd testdir/`, and `ls` again.



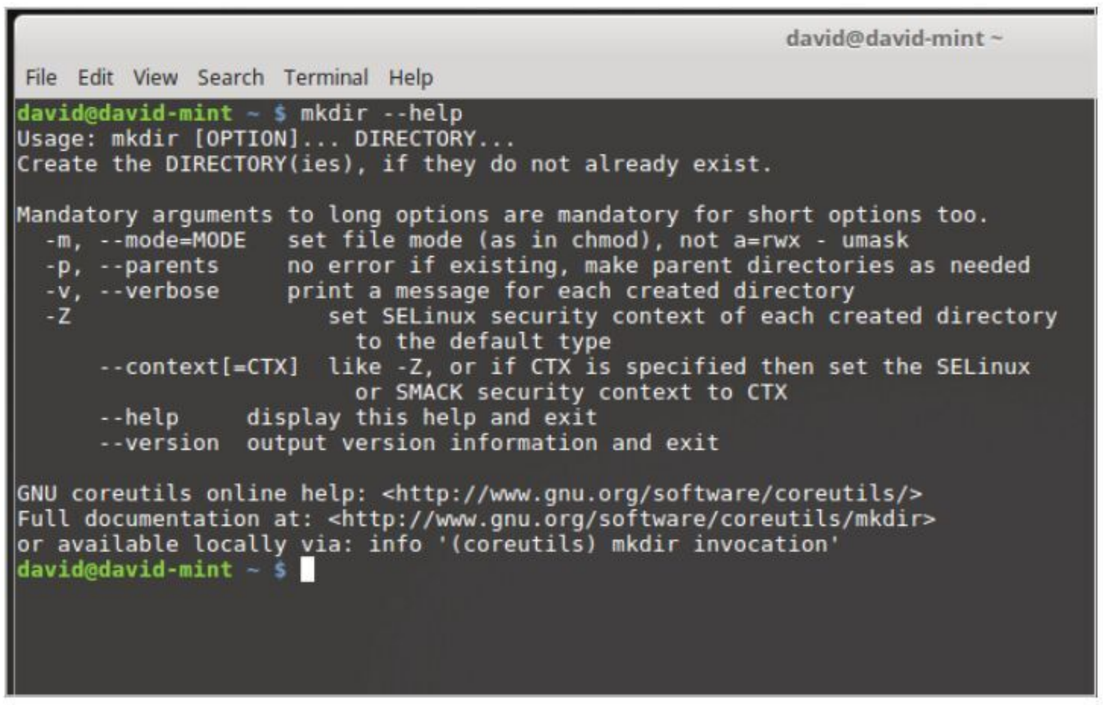
STEP 5 The command to create directories is quite logical, therefore. You'll create the directory, and any sub-directories within. However, what if you want to create a directory and a sub-directory in a single command? Make sure you're at Home (**cd ~**) and enter: `mkdir -p Temp/finances`. Now, **cd Temp/**, and **ls** to list the new directory.



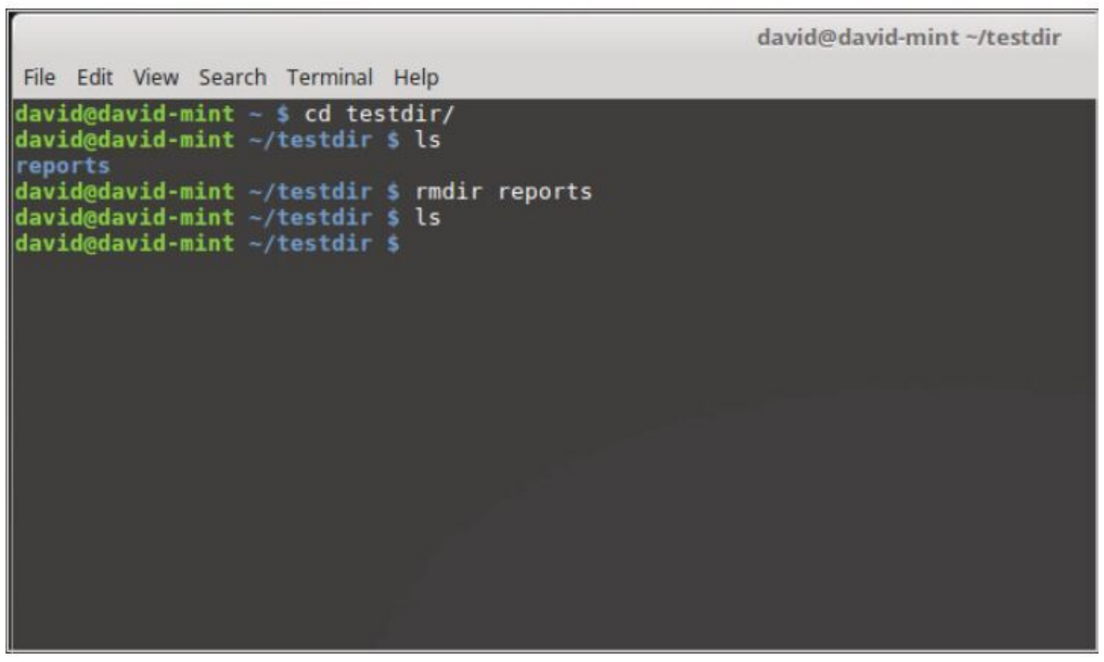
STEP 6 The **-p** option is what enables the mkdir command to create the sub-directory as well as the parent directory. In Linux, commands always follow the same structure: Command, Option, and Argument. In the previous step example, command (**mkdir**), option (**-p**), and argument (**Temp/finances**).



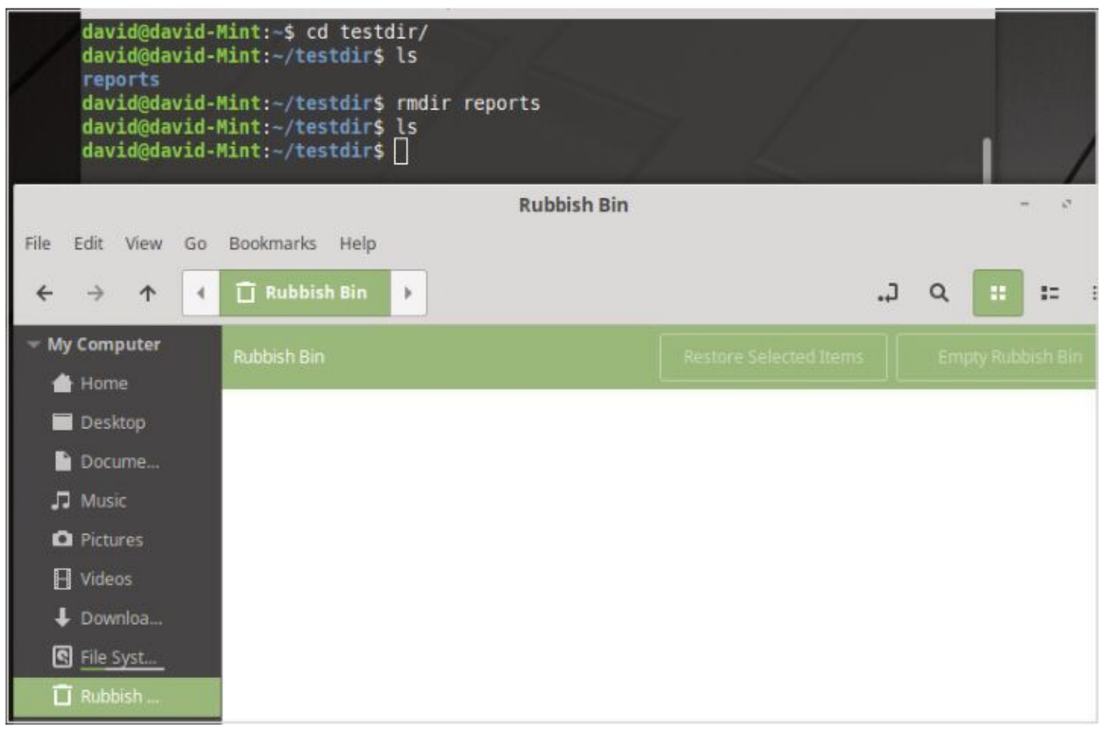
STEP 7 If you want to drill down into the various options available for the mkdir command, you can enter `mkdir --help` into the Terminal. This will provide a quick help guide detailing the options and how the command structure works.



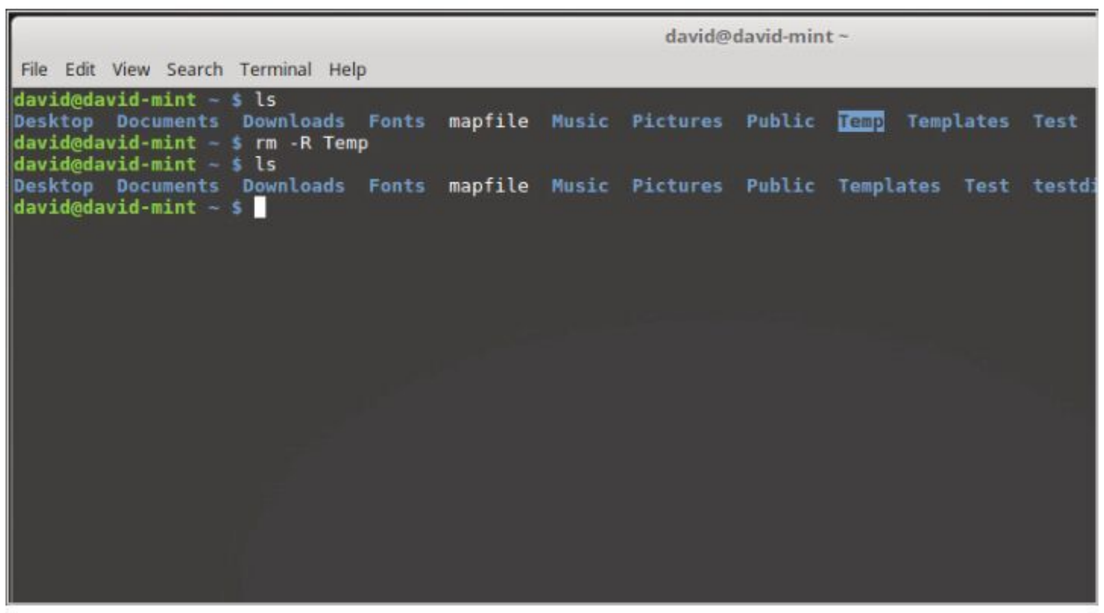
STEP 8 Now that we've created some directories, let's see about removing them. Start by entering the testdir directory and listing its contents: **cd testdir/**, then **ls**. The previously created reports sub-directory is present. One way to remove it is to enter: **rmdir reports**, then **ls** again to confirm it's not there.



STEP 9 A quick warning: removing a directory in the Terminal doesn't place it in the Mint Rubbish Bin, via Nemo file manager. The same goes for any files, too. If you remove a directory from the Terminal command then it's gone for good.



STEP 10 Rmdir will only remove empty directories, to remove directories containing sub-directories, or even files, you'll need to use the rm command with the -R option. For example, on the Temp/finance directories, use `rm -R Temp`. A quick ls reveals that the parent folder and all of its contents are removed. Careful when using this command.



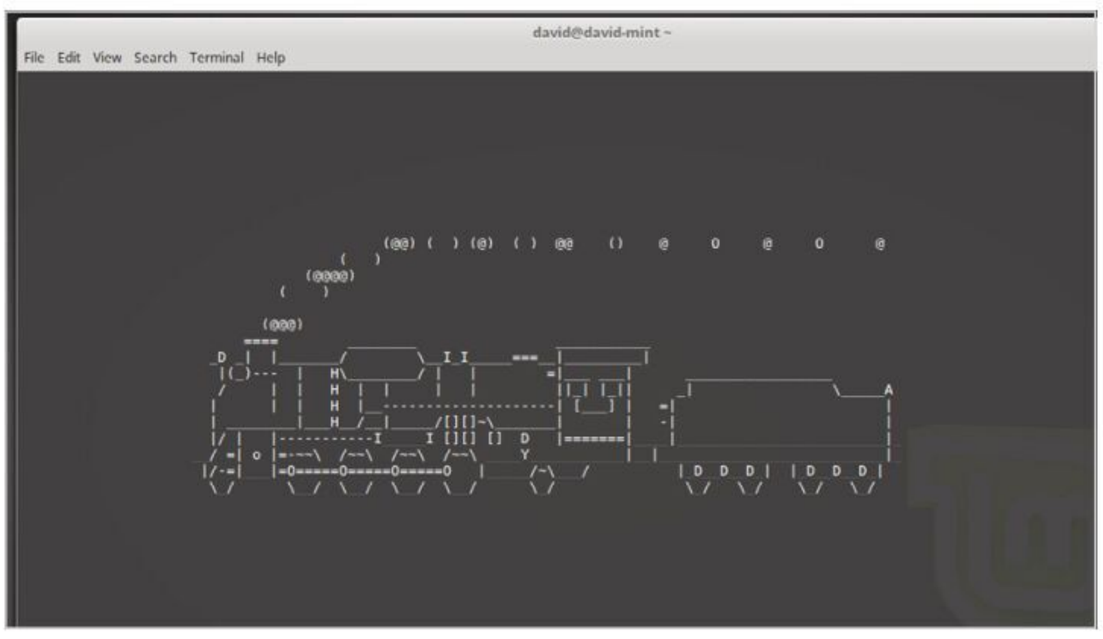
Fun Things to do in the Terminal

Despite the seriousness of an operating system, the Linux community are certainly no strangers to a bit of fun. The developers over the years have created and inserted all manner of fun and odd elements into the Terminal.

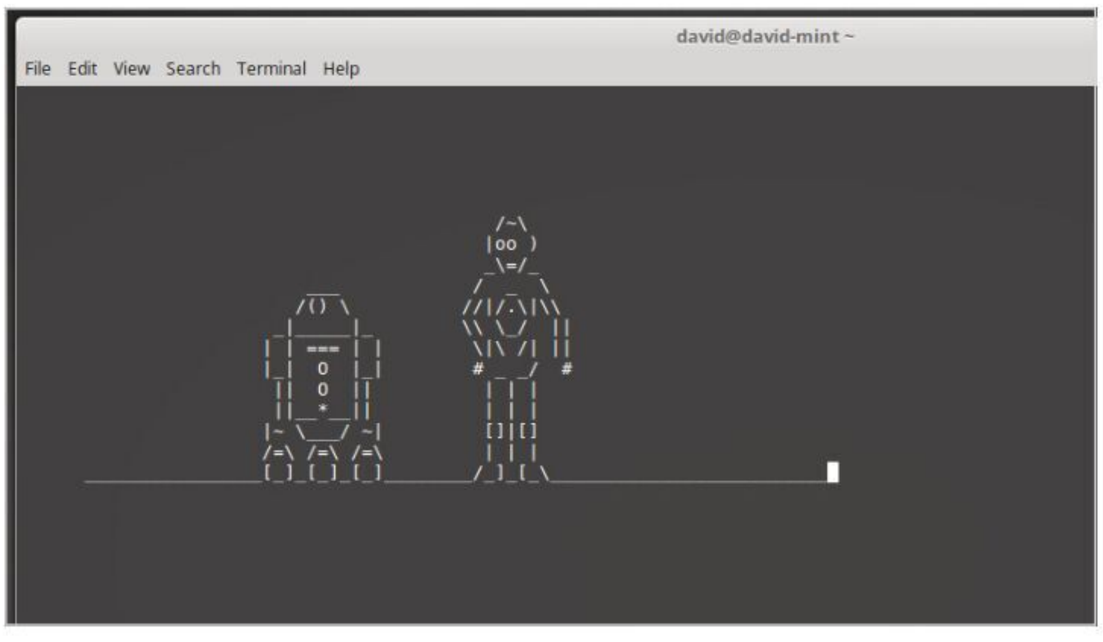
TERMINAL FUN

You'll be working exclusively in the Terminal for these next two sections, so start warming up your fingers. After all, all work and no play… as the saying goes.

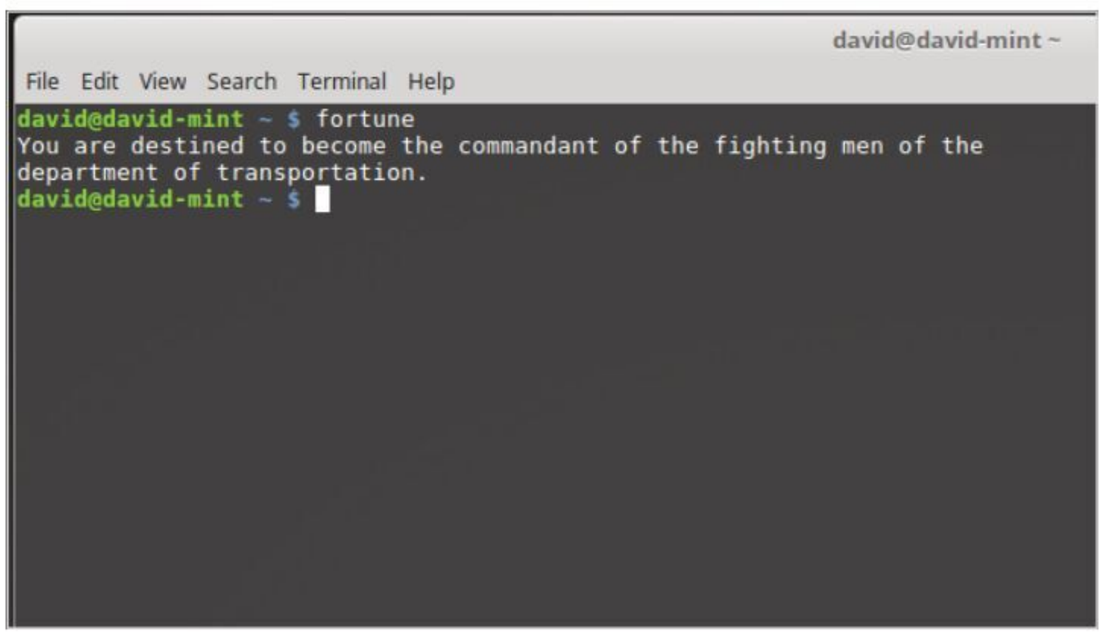
STEP 1 The first command we're going to use is **sl**, it's not installed by default so enter: `sudo apt-get install sl`. The command can be run with **sl** and when executed will display a Steam Locomotive travelling across the screen (hence 'sl'). Entering `LS`, note the upper case, also works.



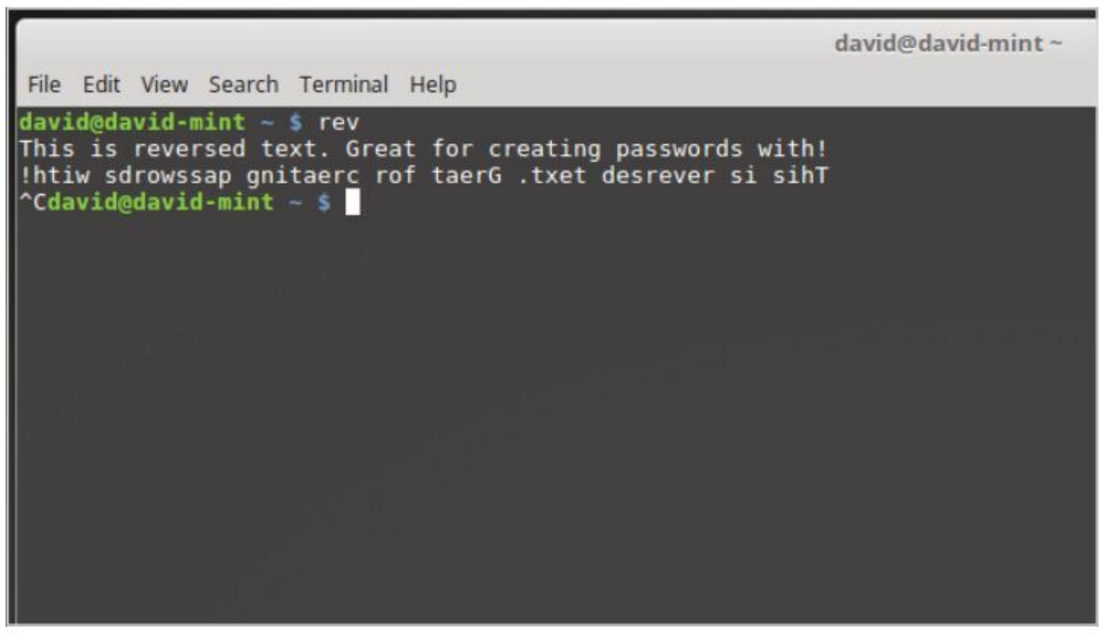
STEP 2 Fans of Star Wars even get a fix when it comes to the Terminal. By linking to a remote server via the telnet command, you can watch Episode IV: A New Hope being played out, albeit in ASCII. To view this spectacle, enter: `telnet towel.blinkenlights.nl`.



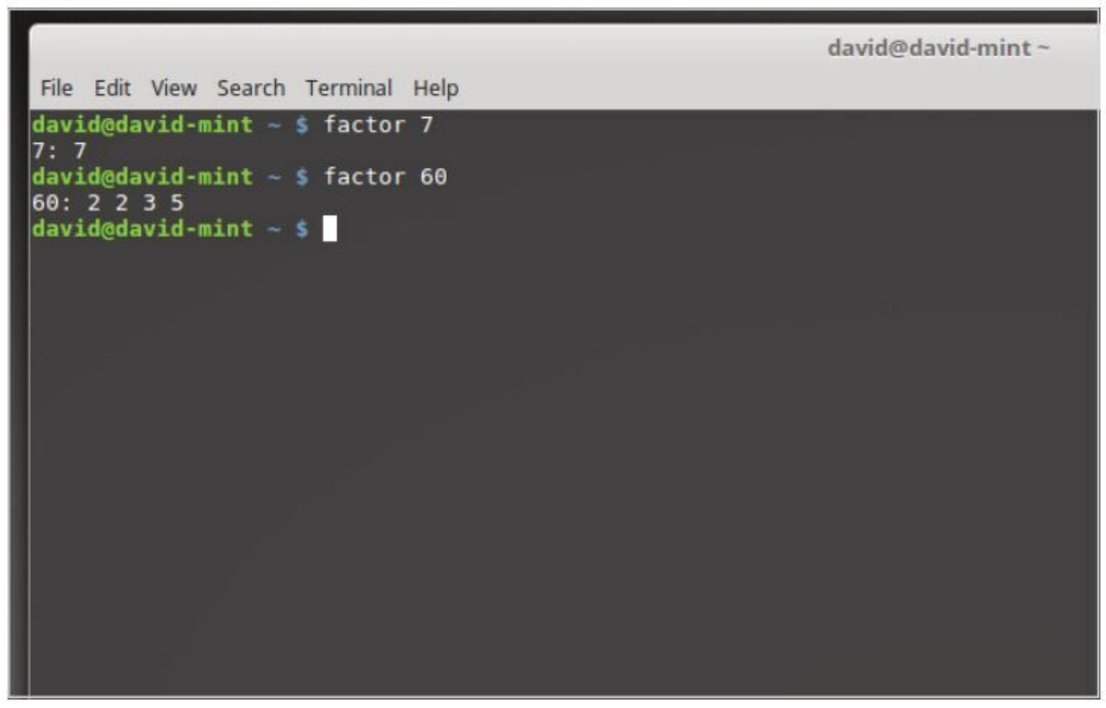
STEP 3 If you've ever fancied having the computer read a random fortune out to you, then you're in luck. Most distros require you to install the fortune app, however Linux Mint differs somewhat by having it already pre-loaded. All you need to do is enter the command `fortune` into the Terminal, and enjoy.



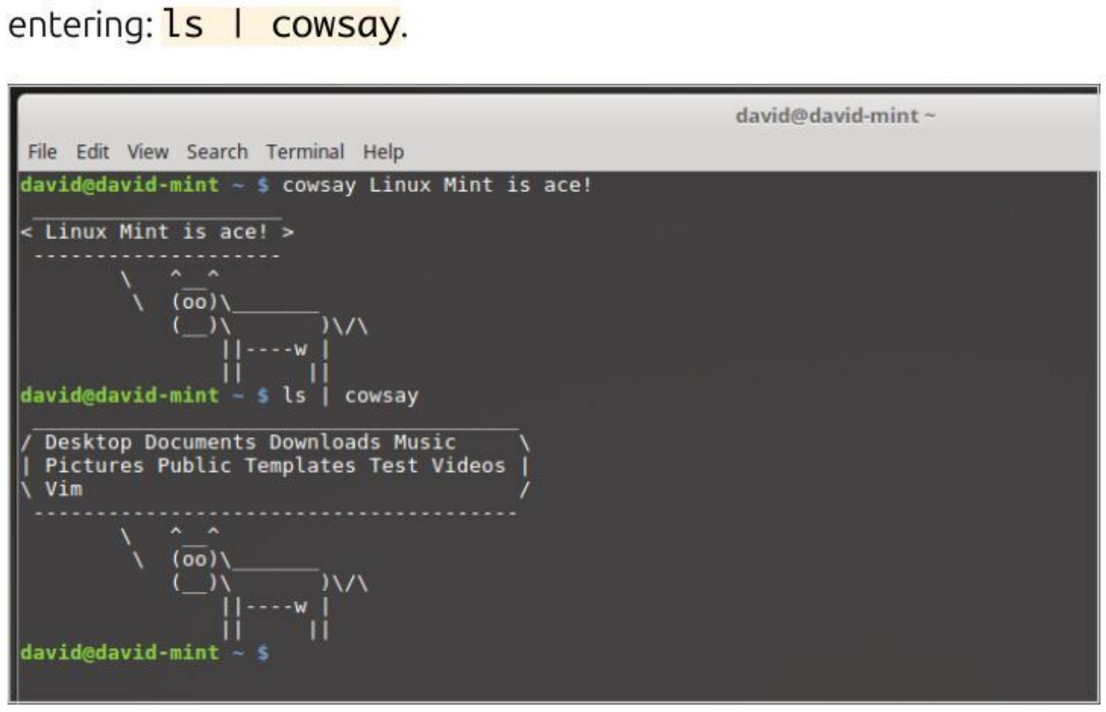
STEP 4 The **rev** command is certainly interesting, and at first what seems a quite useless addition to the OS. However, it can be used to create some seemingly unbreakable passwords. Enter: `rev`, now type some text, when you press **Enter** next, everything you typed in will be reversed. Press **Ctrl+C** to exit.



STEP 5 If you're stuck trying to work out all the possible factors for any particular number, simply enter `factor` followed by the number. For example, `factor 7` doesn't offer much output, whereas `factor 60` displays more.



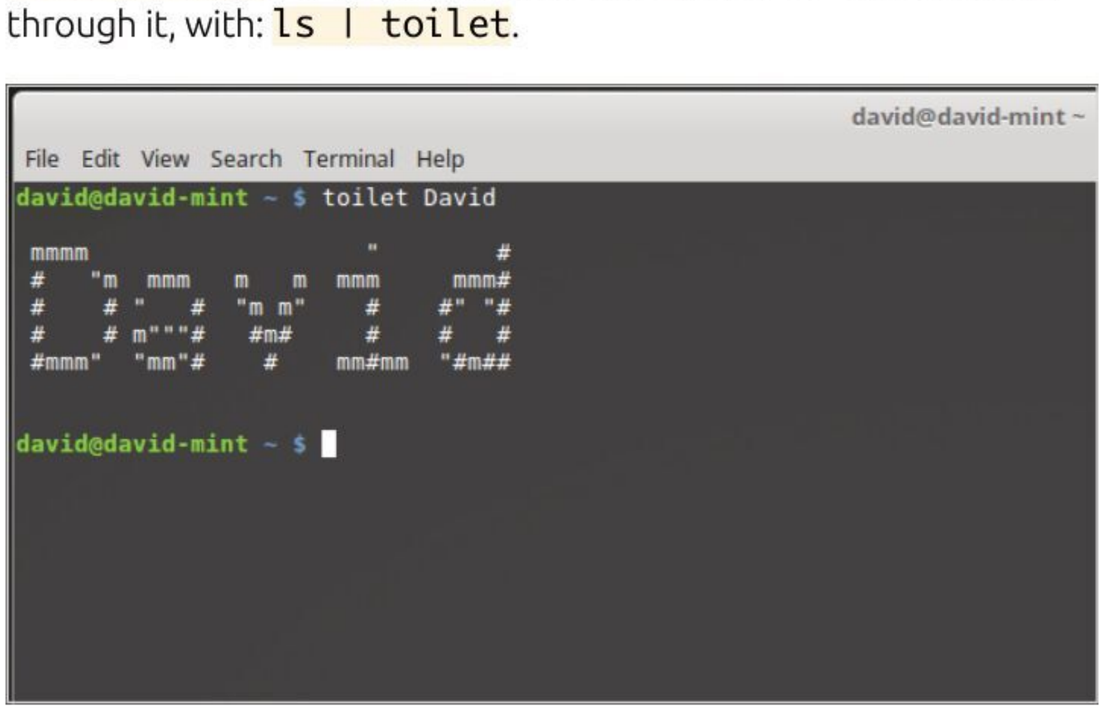
STEP 6 There's a fine line between the rather cool and really-quite-weird. Having an ASCII cow repeat text to you could potentially fall in the latter. Enter `cowsay` followed by any text you want, such as: `cowsay Linux Mint is ace!`. In fact, you can even output the **ls** command through the cow, by entering: `ls | cowsay`.



STEP 7 To further the cow element, there's even a graphical, i.e. non-Terminal, cow available. Install it with: `sudo apt-get install xcowsay`, then when it's installed enter something similar to cowsay, such as: `xcowsay BDM Publications`.

STEP 8 If you really want to expand the whole cow thing, for whatever reason, then pipe the fortune command through it, with: `fortune | cowsay`; and for the graphical cow equivalent: `fortune | xcowsay`. Plus, there's always cowthink. Try: `cowthink ...This book is awesome`.

STEP 9 The command `toilet` doesn't inspire much confidence, we'll admit. However, it's not as bad as it first sounds. Start by installing it with: `sudo apt-get install toilet`. Then when installed, type something along the lines of: `toilet David`. Or perhaps list the contents of the current folder through it, with: `ls | toilet`.



STEP 10 Expanding the toilet command, you can actually generate some decent looking graphics through it. For example, try this: `toilet -f mono12 -F metal David`. You can enter `toilet --help`, for a list of the command line arguments to expand further.

More Fun Things to do in the Terminal

If the previous list of fun, and quite bizarre, things to do in the Terminal has you wanting more, you're in luck. We've put together another batch of some useful, and some not so useful, commands for you to try out.

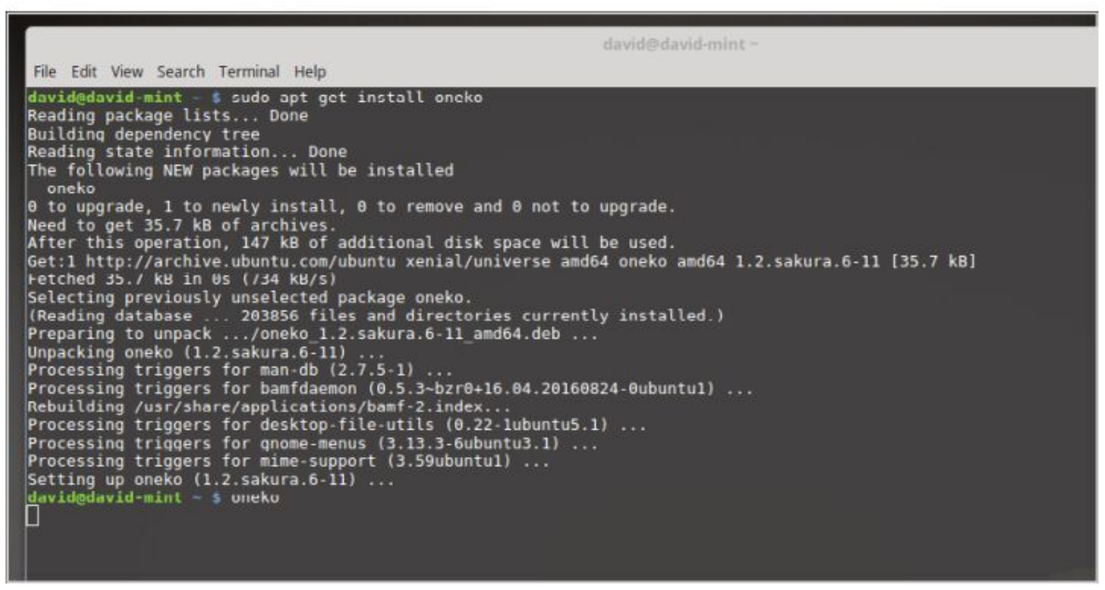
MORE FUN, YAY

Since the Terminal session is already open, and your keyboard digits are nicely warmed up, here are another two pages of Terminal nonsense.

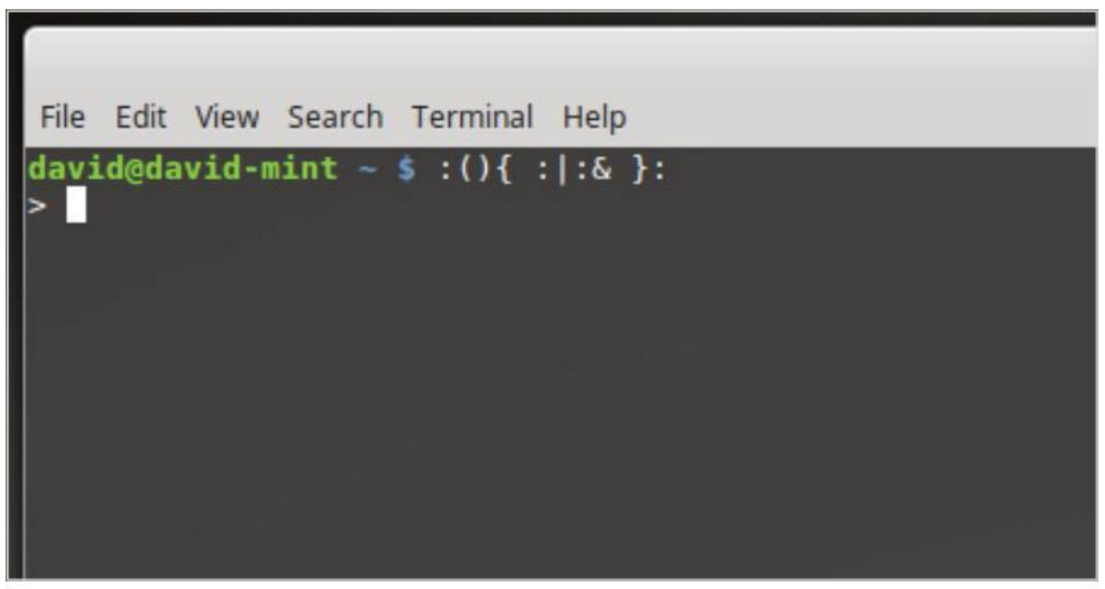
STEP 1 Remember the old ZX Spectrum days of computing, when you could type in 10 print "Hello", 20 goto 10 and Hello would list down the screen? Well, in Linux Mint you can do the same. Simply enter `yes` followed by some text, i.e. `yes Linux is ace`. It'll keep going until you press Ctrl+C.

STEP 2 The Matrix was one of the most graphically copied films ever released; there's even a version of the Matrix code available for Linux Mint. Install it with: `sudo apt-get install cmatrix`. When it's done enter: `cmatrix` and follow the white rabbit, Neo. Unlike the real Matrix though, you can press `Ctrl+C` to exit.

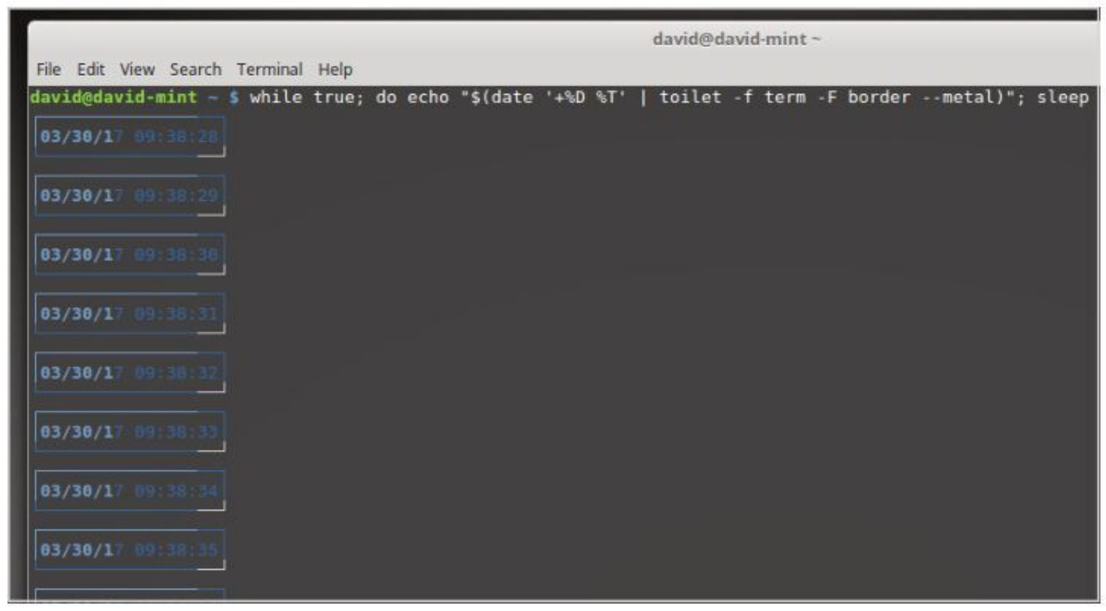
STEP 3 Having a little white cat chase your mouse pointer around the desktop may sound like a terrible waste of time. Oddly though, it isn't. Enter: `sudo apt-get install oneko`, then type `oneko` to have the cat appear. Move your 'mouse' cursor around the screen and the cat will chase it. Use `Ctrl+C` to exit the action.



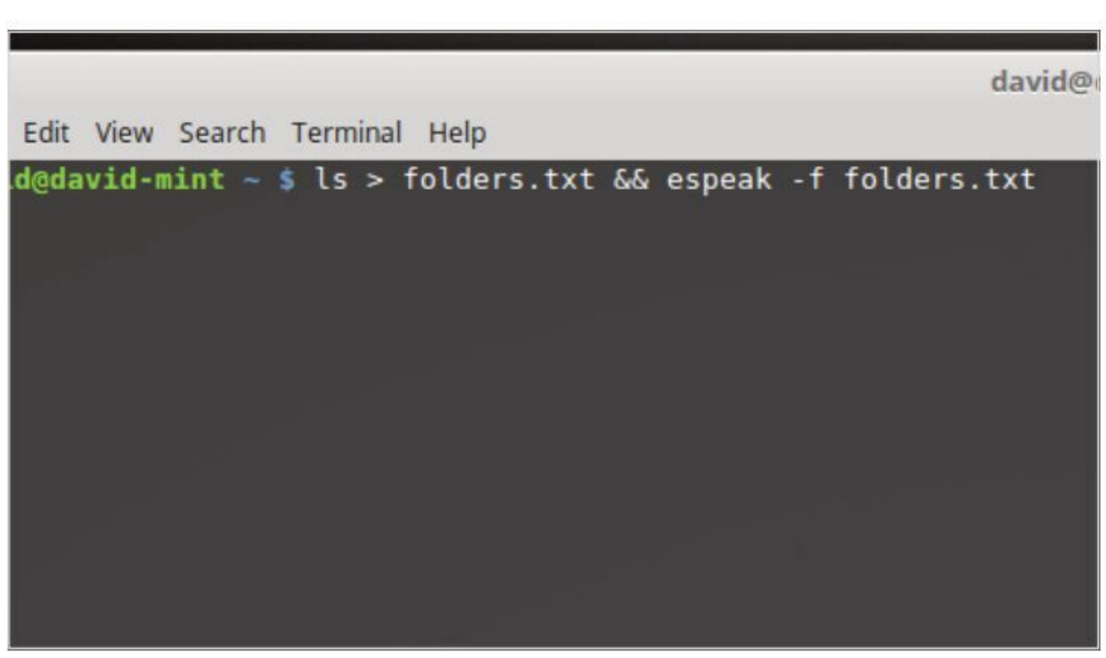
STEP 4 This entry is a little more serious than the previous. It's called the **Fork Bomb** and what it does, basically, is continually replicate itself until it has used up all the available system resources, thus causing your computer to crash. You don't have to try it but it's interesting nonetheless. Simply enter `:(){ :|:& }:` and be prepared to reboot.



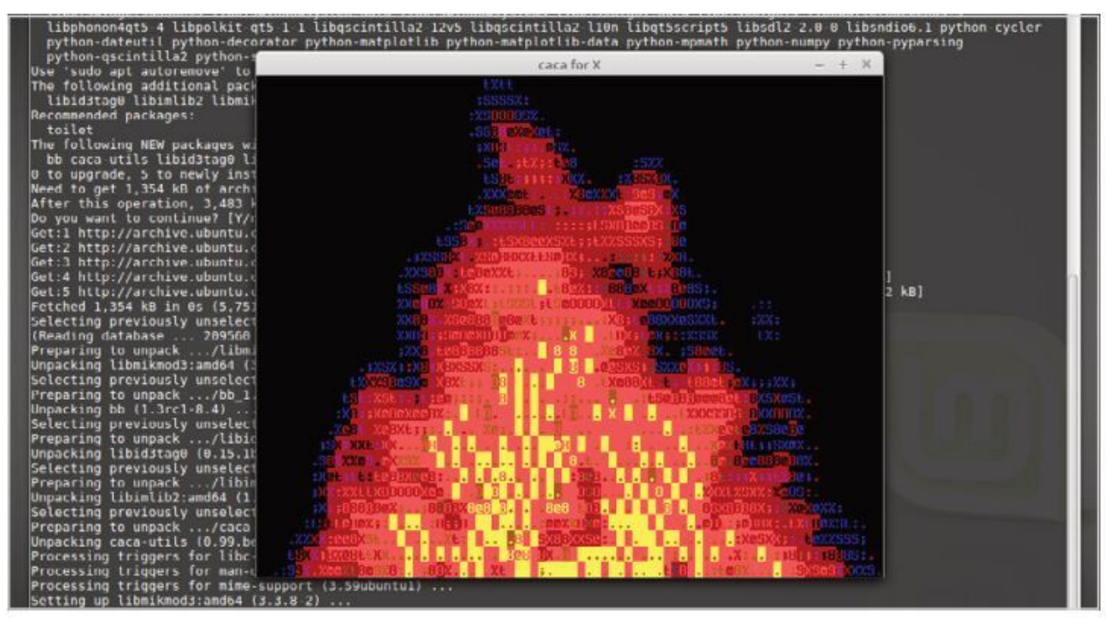
STEP 5 Stringing several commands and piping them through other commands is what makes scripting such a powerful element to an OS. Using the while command, for example, together with toilet, can yield some impressive results. Enter: `while true; do echo "$(date '+%D %T' | toilet -f term -F border --metal)"; sleep 1; done`.



STEP 6 Talking computers were the craze of the 80s, enter: `espeak "Hello, this is Linux Mint"` to have the computer repeat the text inside the quotes to you. Make sure your volume is turned up, and try the following: `ls > folders.txt && espeak -f folders.txt`. This will have Mint read back the contents of the ls command.

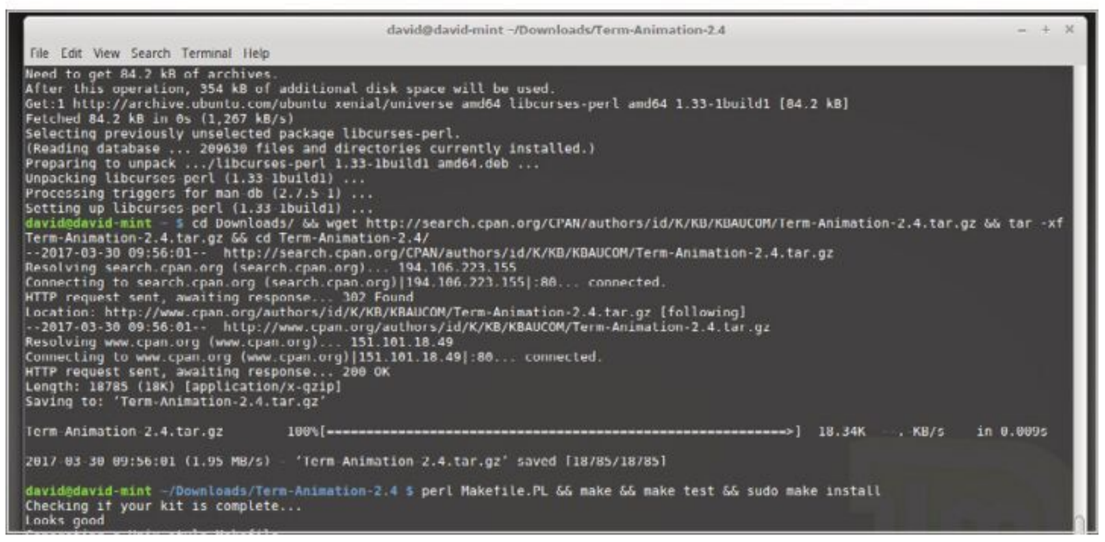


STEP 7 A roaring ASCII fire isn't the most useful command to have at your disposal, but it's fun. Install it with: `sudo apt-get install libaa-bin`, then when installed use: `aafire`. It's not exactly warming but you get the idea. To expand the above, enter: `sudo apt-get install bb caca-utils`, then, `cacafire`.

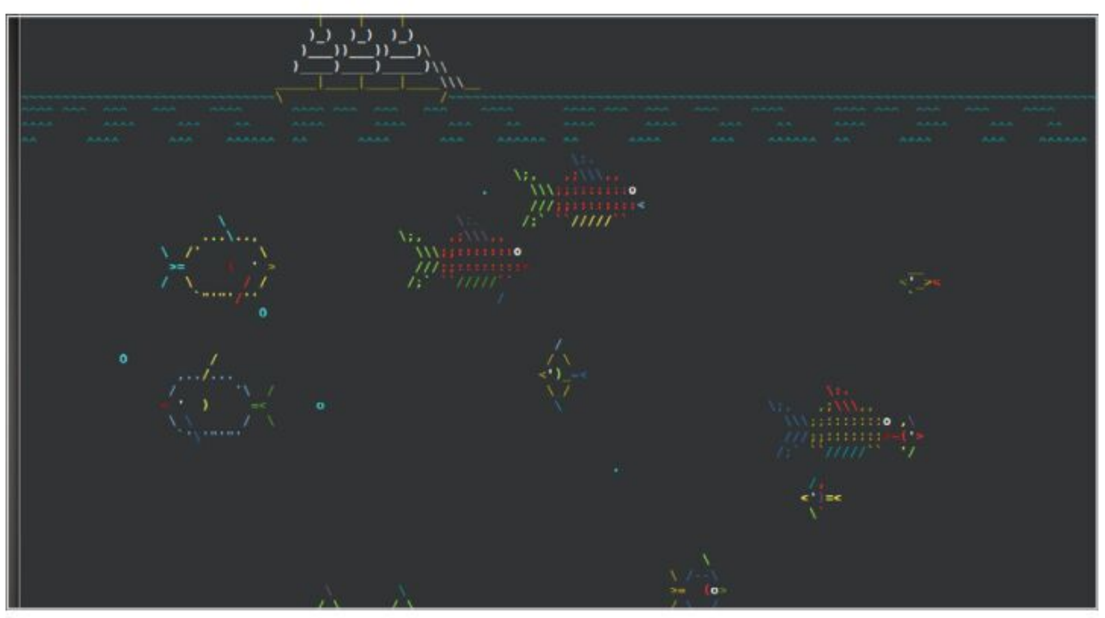


STEP 8 Used as a music demo from the old Amiga and DOS days, the **bb** command reminds us of getting hold of three and a half inch floppies crammed with all manner of demoscene goodies. We've already installed bb from the previous step, so just enter **bb**. Follow the on-screen instructions, and turn up your volume.

STEP 9 This entry is in two parts. First you need to get hold of the necessary packages: `sudo apt-get install libcurses-perl`. When that's done enter: `cd Downloads/ && wget http://search.cpan.org/CPAN/authors/id/K/KB/KBAUCOM/Term-Animation-2.4.tar.gz && tar -xf Term-Animation-2.4.tar.gz && cd Term-Animation-2.4/`. Then: `perl Makefile.PL && make && make test && sudo make install`.



STEP 10 With that little lot done, onto the next. Enter: `cd .. && wget http://www.robobunny.com/projects/asciiquarium/asciiquarium.tar.gz && tar -xf asciiquarium.tar.gz && cd asciiquarium_1.1/ && chmod +x asciiquarium`. Providing all went well, enter `./asciiquarium` and enjoy your very own ASCII-based aquarium.



Linux Tips and Tricks

As you've seen, the Linux Terminal is quite an exceptional environment. With a few extra apps installed, and a smidgen of command knowledge, incredible, and often quite strange, things can be accomplished.

TAKING COMMAND

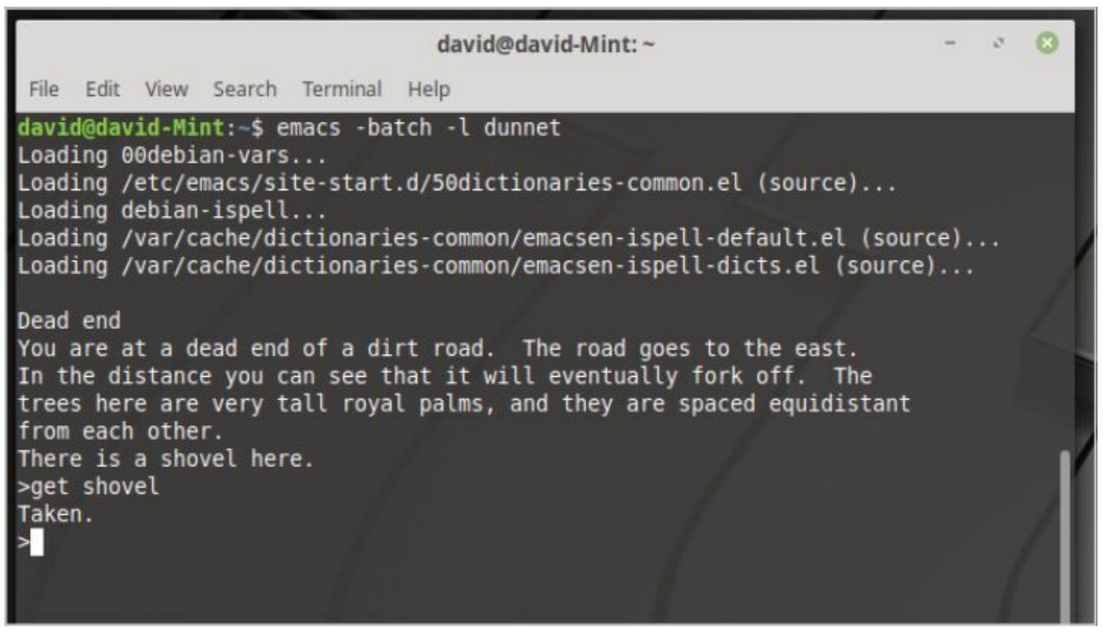
There are countless Linux tips, secrets, hacks and tricks out there. Some are very old, originating from Linux's Unix heritage, while others are recent additions to Linux lore. Here's our favourite ten tips and tricks.

EASTER EGGS

Emacs, the text editor, is a great piece of software, however, did you know it also contains a hidden Easter Egg? With Emacs installed (**sudo apt-get install emacs25**), drop to a Terminal session and enter:

```
emacs -batch -l dunnet
```

Dunnet is a text adventure written by Ron Schnell in 1982, and hidden in Emacs since 1994.



TERMINAL BROWSING

Ever fancied being able to browse the Internet from the Terminal? While not particularly useful, it is quite a fascinating thing to behold. To do so, enter:

```
sudo apt-get install elinks
elinks
```

Enter the website you want to visit.

MOON BUGGY

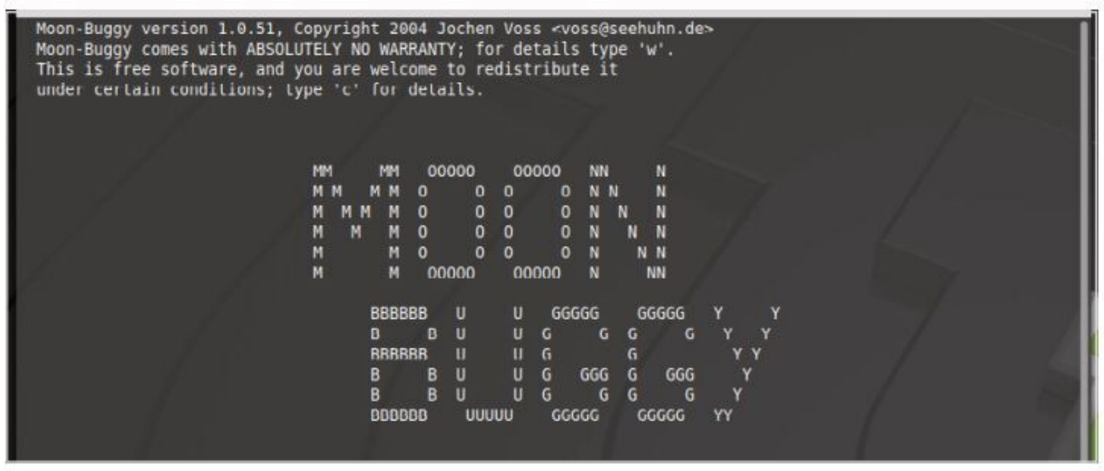
Based on the classic 1982 arcade game, Moon Patrol, Moon Buggy appeared on the home computers of 1985 amid much praise. It's a cracking Atari game, and it's available in the Linux Terminal by entering:

```
sudo apt-get install moon-buggy
```

Then:

```
moon-buggy
```

Enjoy.



LET IT SNOW

Snowing in the Terminal console isn't something you come across every day. If you're interested, however, enter:

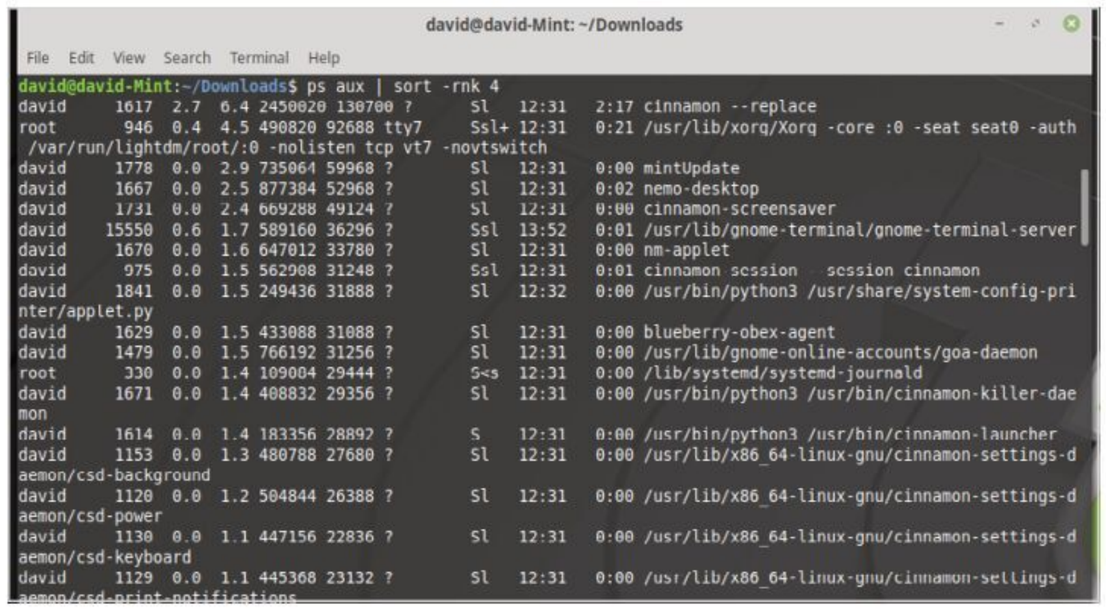
```
wget https://gist.githubusercontent.com/sontek/
        1505483/raw/7d024716ea57e69fb52632fee09f42
        753361c4a2/snowjob.sh
chmod +x snowjob.sh
./snowjob.sh
```

MEMORY HOGS

If you need to see what apps are consuming the most memory on Linux, simply enter:

```
ps aux | sort -rnk 4
```

This sorts the output by system memory use.



BBS

Back in the days of dial-up connections, the online world was made up of Bulletin Board Systems. These remote servers provided hang-outs for users to chat, swap code, play games and more. Using telnet in Linux, we can still connect to some active BBSes:

```
telnet battlestarbbs.dyndns.org
```

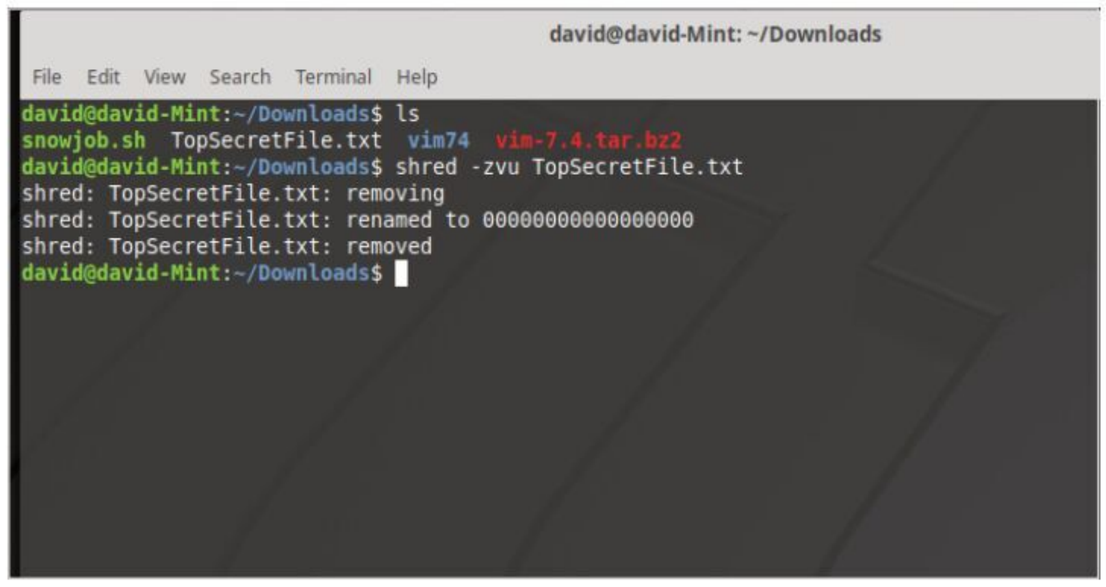
There are countless operational BBSes available, check out **https://www.telnetbbsguide.com/bbs/list/detail/**, for more.

SHREDDER

When you delete a file, there's a chance of someone with the right software being able to retrieve it. However, to securely and permanently delete a file, use Shred:

```
shred -zvu NAMEOFFILE.txt
```

Replace **NAMEOFFILE** with the name of the file to delete.

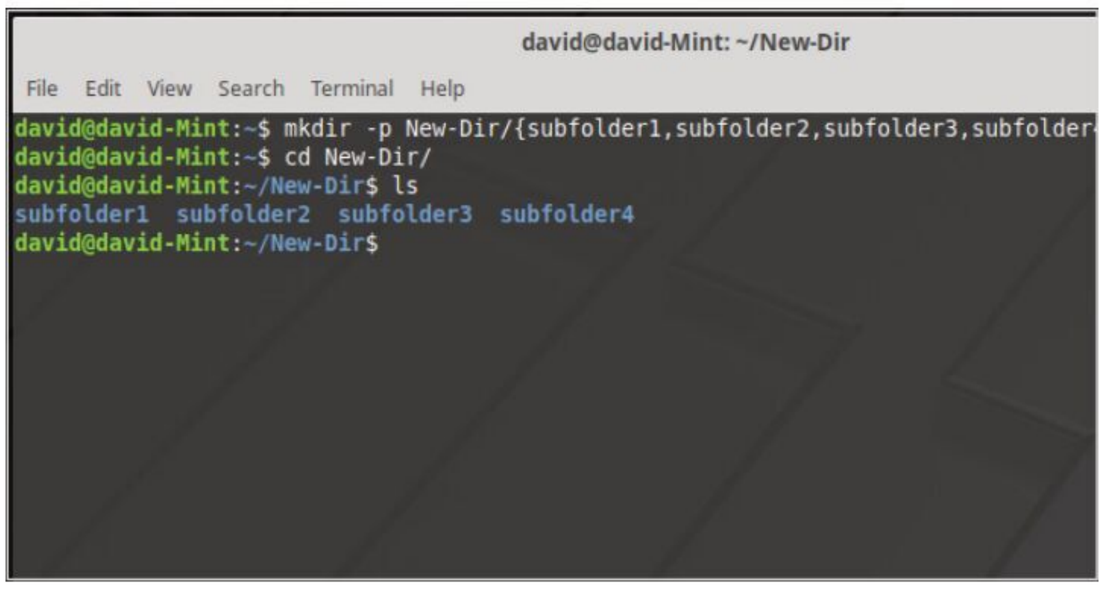


DIRECTORY TREES

If you want to create an entire directory (or folder) tree with a single command, you can use:

```
mkdir -p New-Dir/
{subfolder1,subfolder2,subfolder3,subfolder4}
```

This creates a New-Dir with four sub folders within.

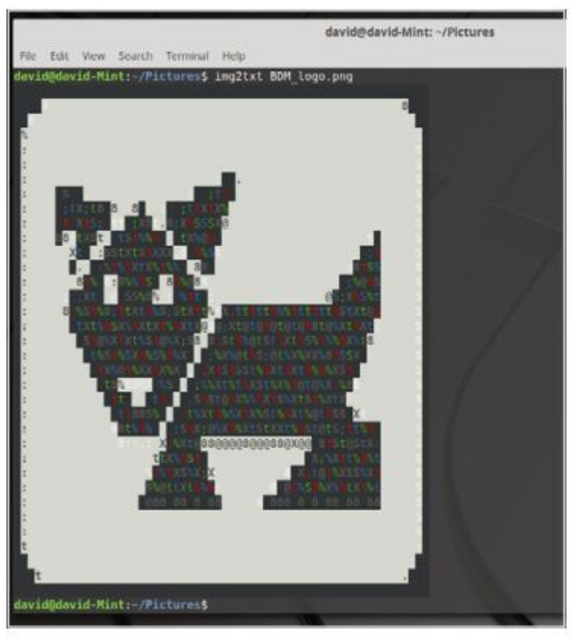


ASCII ART

ASCII art can be quite striking when applied to some images. However, it's often difficult to get just right. You can create some great ASCII art from the images you have by using img2txt:

```
img2txt NAMEOFIMAGEFILE.png
```

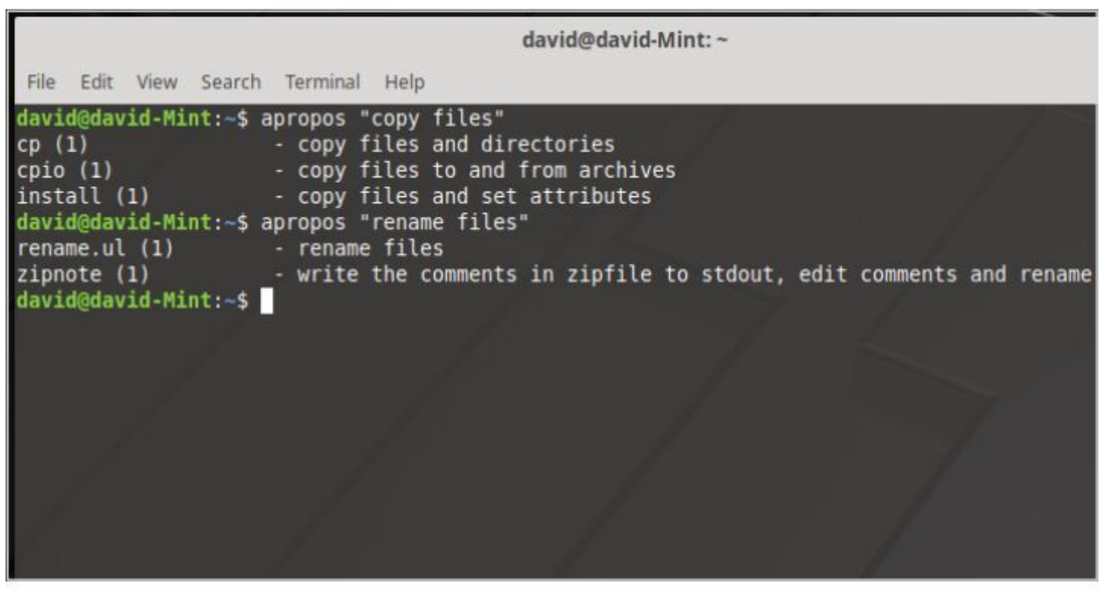
Replace NAMEOFIMAGEFILE with the actual name of the image file on your system. If img2txt install installed, use: `sudo apt-get install caca-utils`.



FORGOTTEN COMMANDS

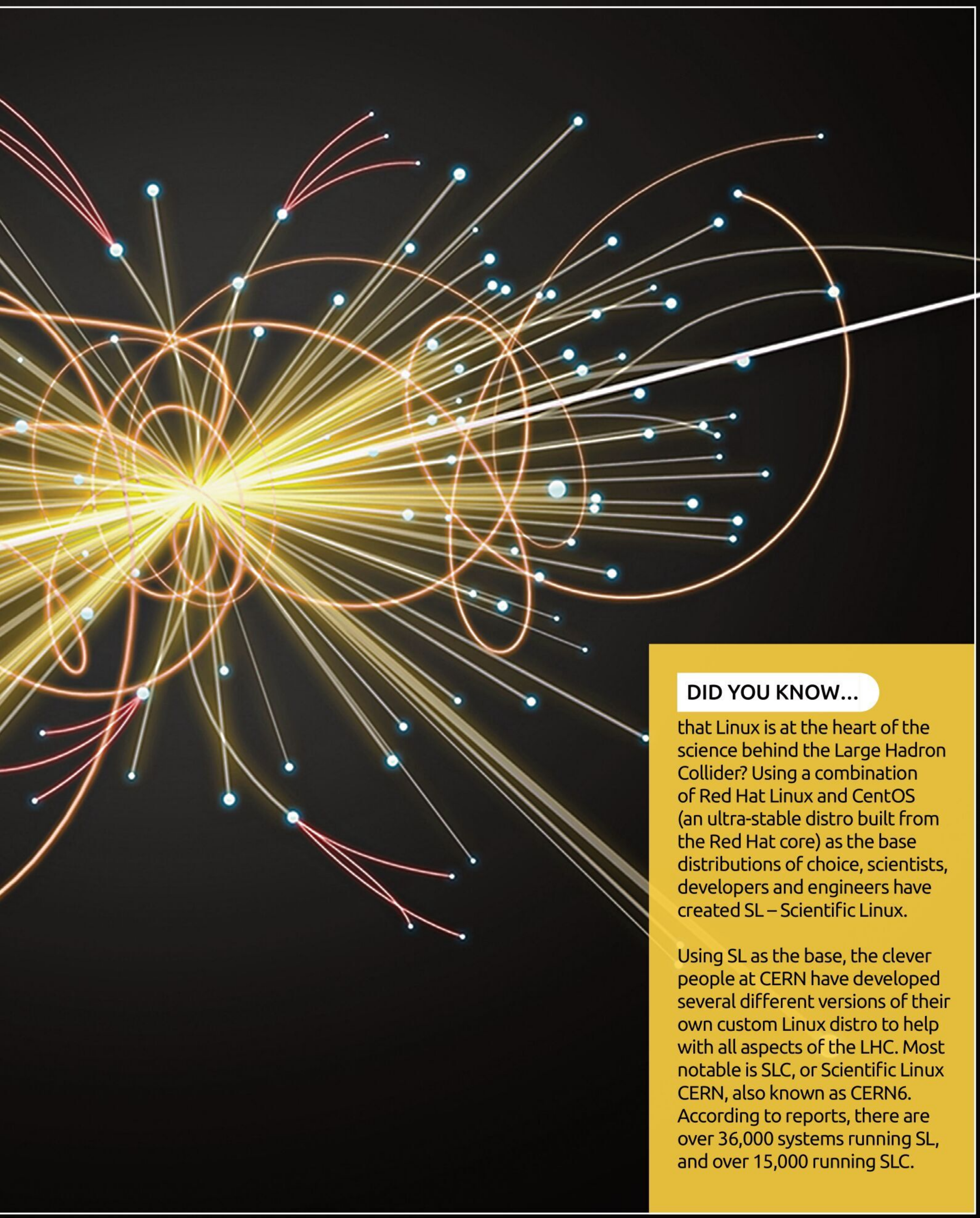
It's not easy trying to remember all the available Linux commands. Thankfully, we can use apropos to help us. Simply use it, along with a description of the command:

```
apropos "copy files"
apropos "rename files"
```



Linux and the **Big Bang**

Linux is colliding particles, so we can understand how the universe works.



DID YOU KNOW...

that Linux is at the heart of the science behind the Large Hadron Collider? Using a combination of Red Hat Linux and CentOS (an ultra-stable distro built from the Red Hat core) as the base distributions of choice, scientists, developers and engineers have created SL – Scientific Linux.

Using SL as the base, the clever people at CERN have developed several different versions of their own custom Linux distro to help with all aspects of the LHC. Most notable is SLC, or Scientific Linux CERN, also known as CERN6. According to reports, there are over 36,000 systems running SL, and over 15,000 running SLC.

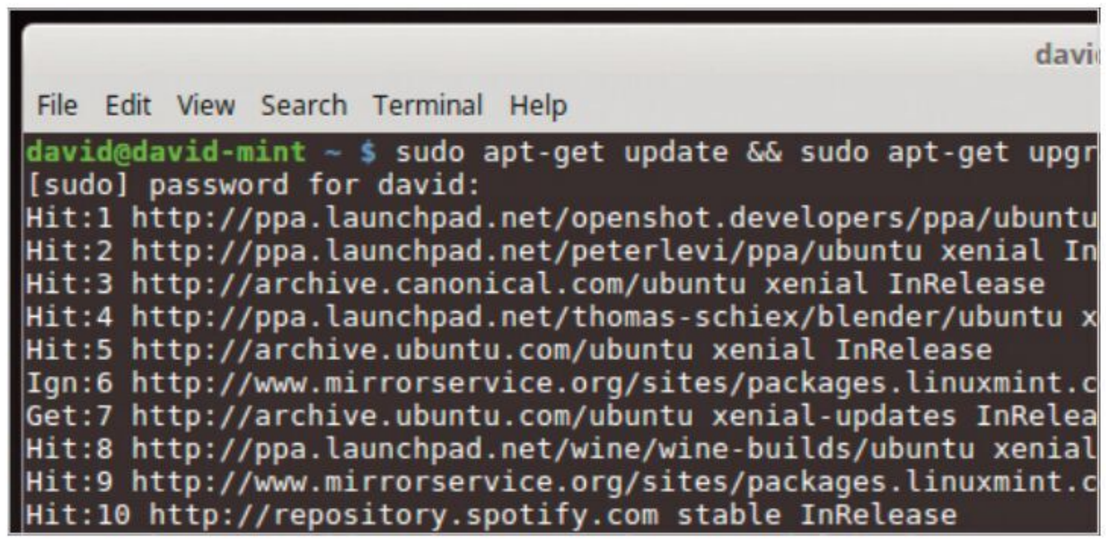
Creating Bash Scripts – Part 1

Eventually, as you advance with Linux Mint, you'll want to start creating your own automated tasks and programs. These are essentially scripts, Bash Shell scripts to be exact, and they work in the same way as a DOS Batch file does, or any other programming language.

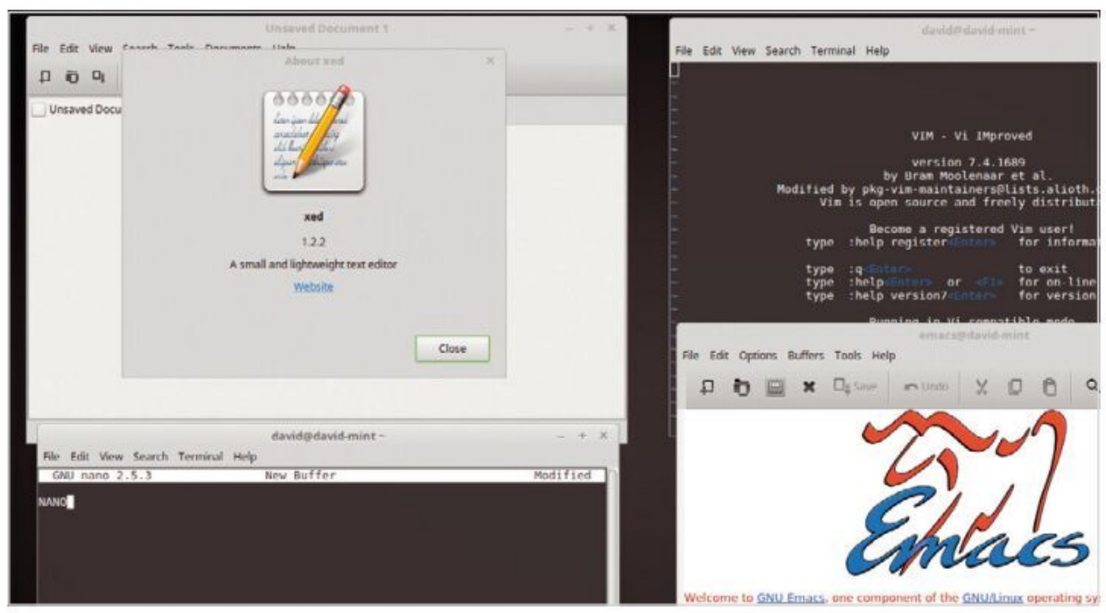
GET SCRIPTING

A Bash script is simply a series of commands that Mint will run through to complete a certain task. They can be simple or remarkably complex, it all depends on the situation.

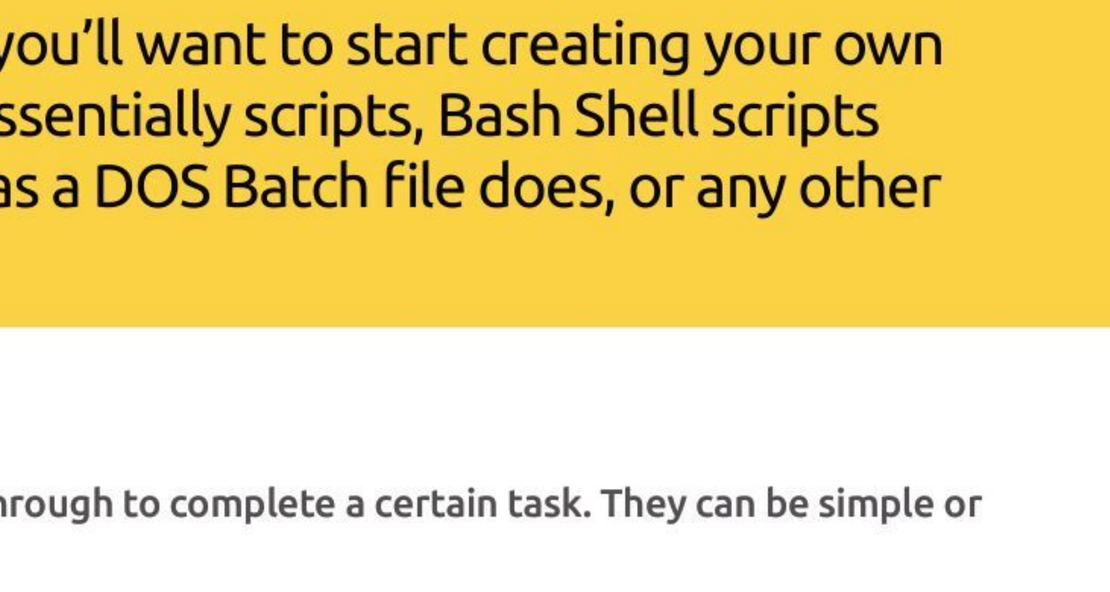
STEP 1 You'll be working within the Terminal and with a text editor throughout the coming pages. There are alternatives to the text editor, which we'll look at in a moment but for the sake of ease, we'll be doing our examples in Xed. Before you begin, however, run through the customary update check: `sudo apt-get update && sudo apt-get upgrade`.



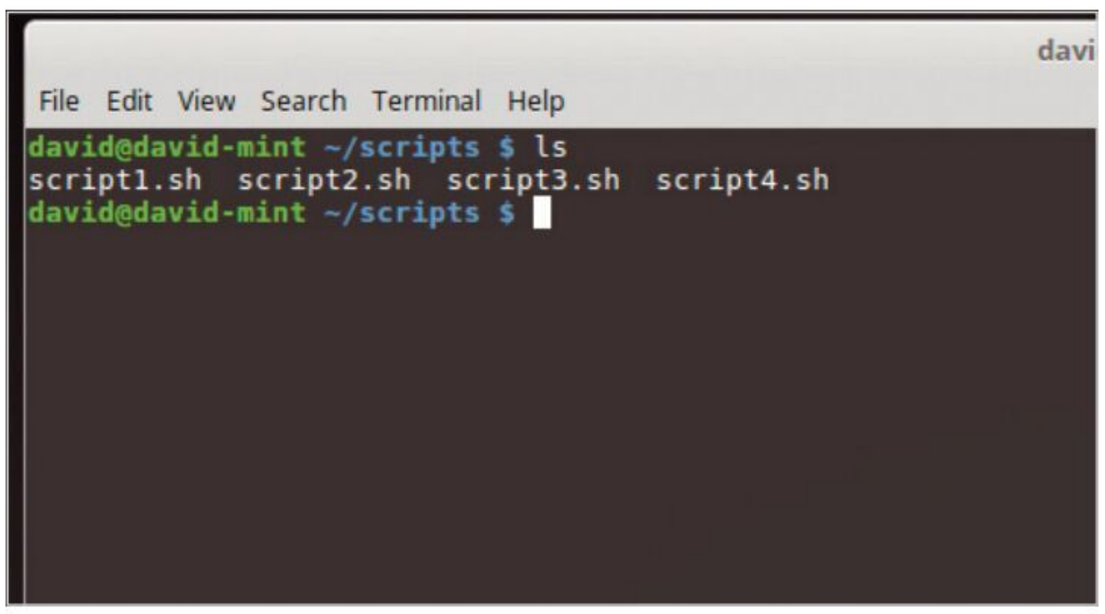
STEP 2 There are several text editors we can use to create a Bash script: Xed, Vi, Nano, Vim, GNU Emacs and so on. In the end it all comes down to personal preference. Our use of Xed is purely due to making it easier to read the script in the screenshots you see below.



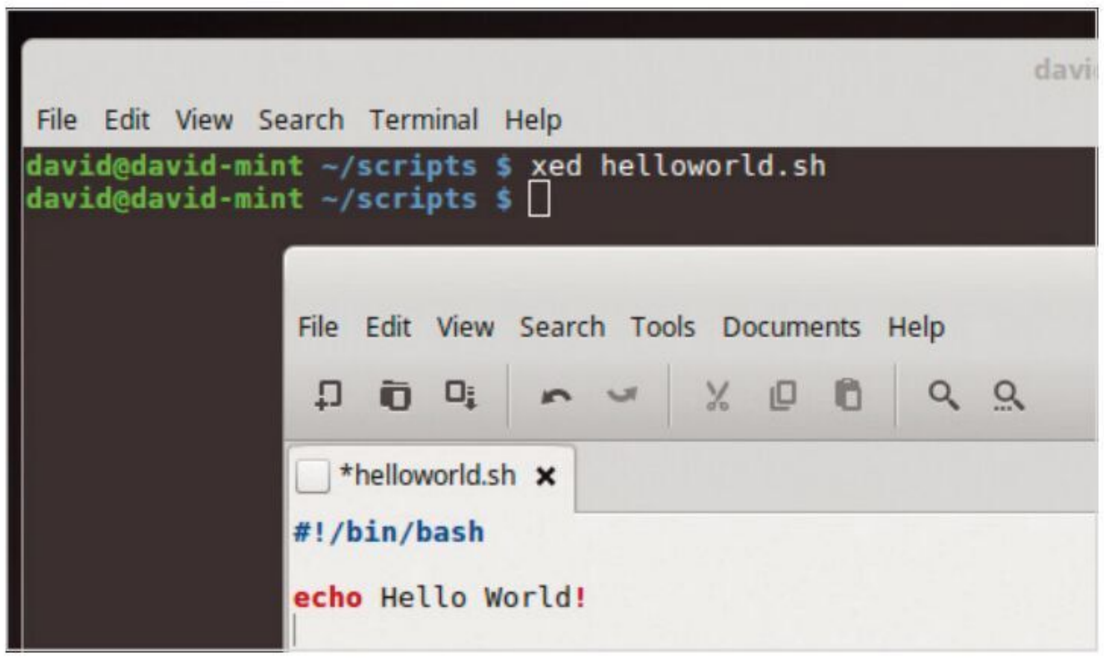
STEP 3 To begin with, and before you start to write any scripts, you need to create a folder where you can put all our scripts into. Start with `mkdir scripts`, and enter the folder `cd scripts/`. This will be our working folder and from here you can create sub-folders if you want of each script you create.



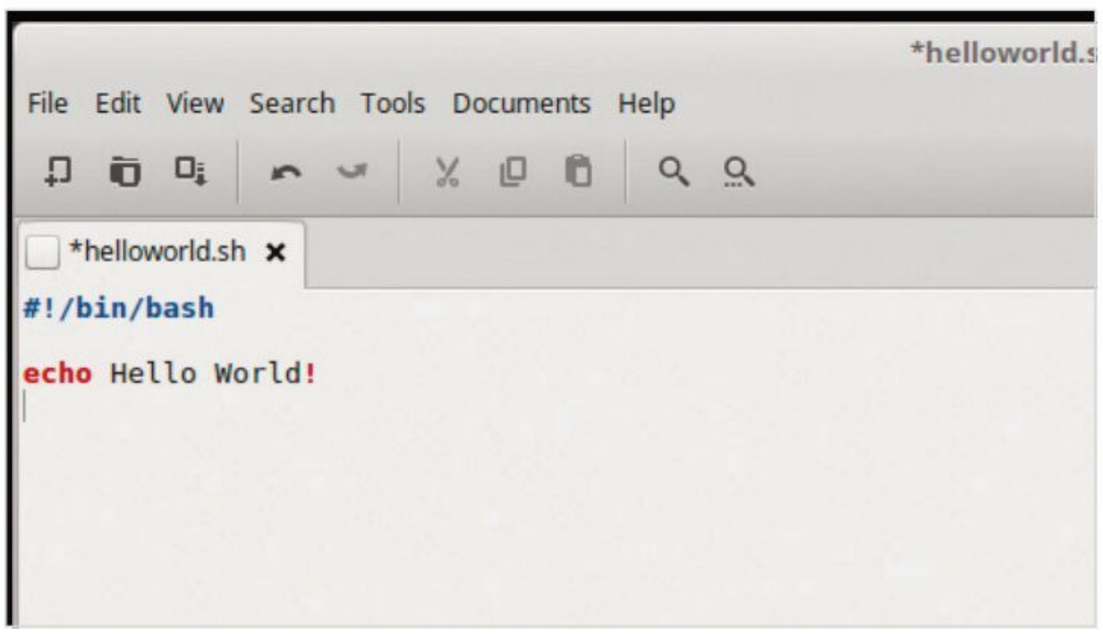
STEP 4 Windows users will be aware that in order for a batch file to work, as in be executed and follow the programming within it, it needs to have a .BAT file extension. Linux is an extension-less operating system but the convention is to give scripts a .sh extension.



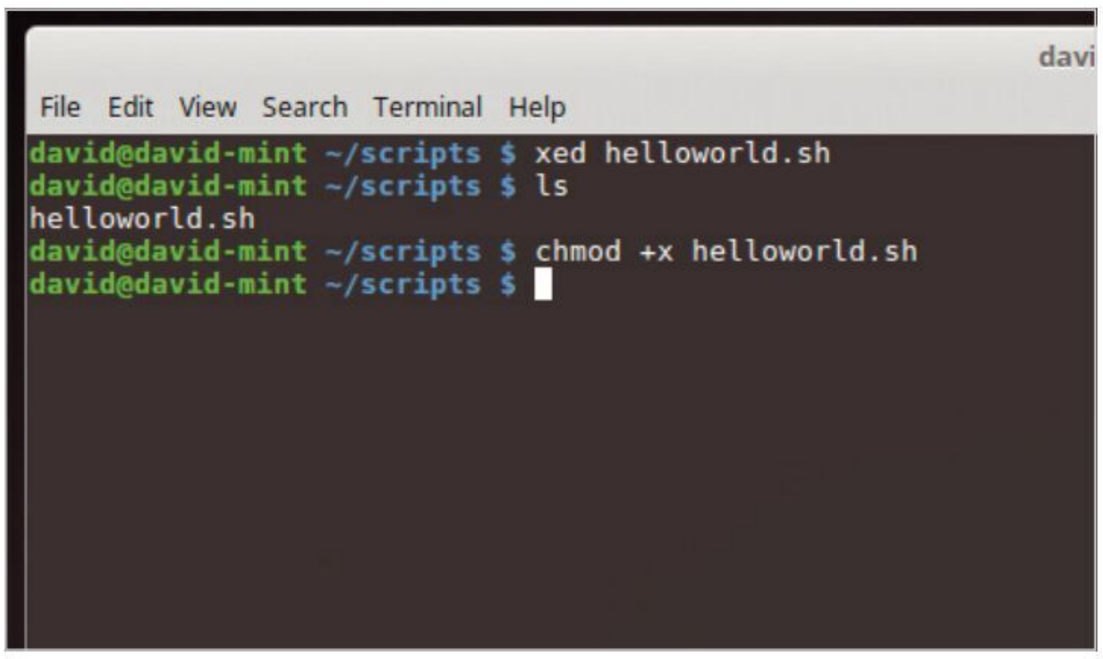
STEP 5 Let's start with a simple script to output something to the Terminal. Enter `xed helloworld.sh`. This will launch Xed and create a file called helloworld.sh. In Xed, enter the following: `#!/bin/bash`, then on a new line: `echo Hello World!`.



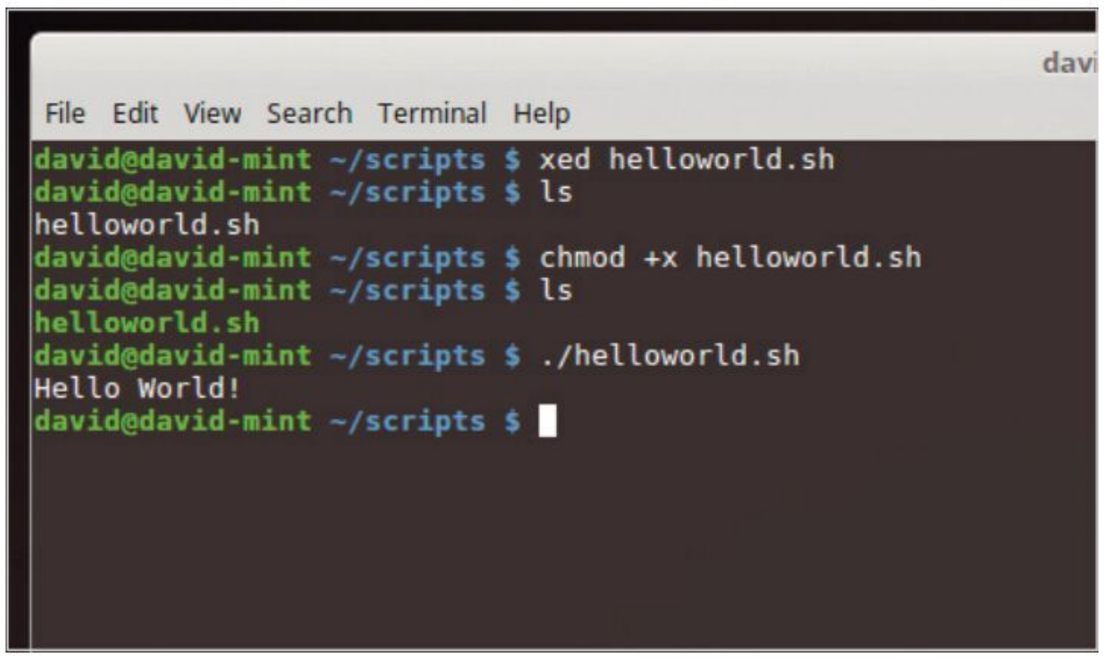
STEP 6 The `#!/bin/bash` line tells the system what Shell you're going to be using, in this case Bash. The hash (#) denotes a comment line, one that is ignored by the system, the exclamation mark (!) means that the comment is bypassed and will force the script to execute the line as a command. This is also known as a Hash-Bang.



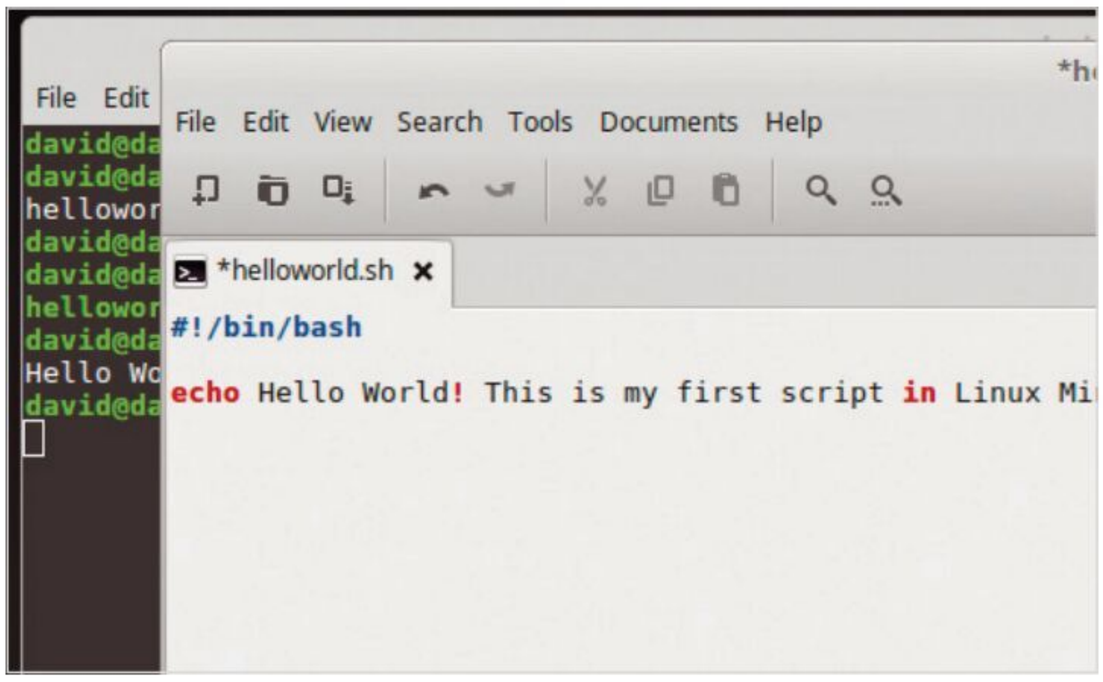
STEP 7 You can save this file, clicking File > Save, and exit back to the Terminal. Entering `ls`, will reveal the script in the folder. To make any script executable, and able to run, you need to modify its permissions. Do this with `chmod +x helloworld.sh`. You need to do this with every script you create.



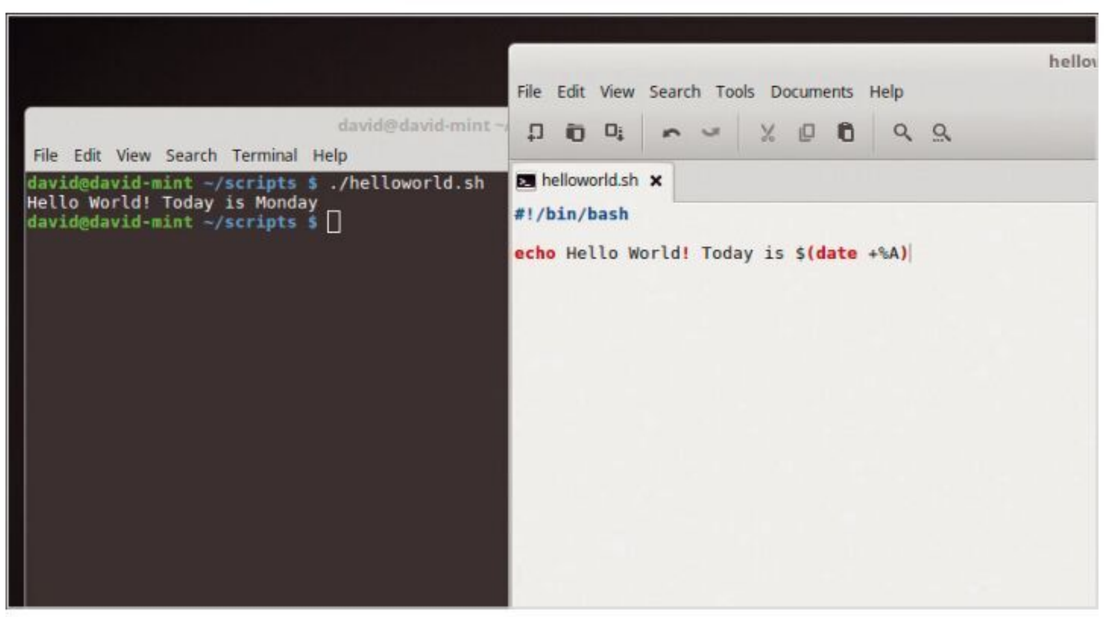
STEP 8 When you enter `ls` again, you can see that the helloworld.sh script has now turned from being white to green, meaning that it's now an executable file. To run the script, in other words make it do the things you've typed into it, enter: `./helloworld.sh`.



STEP 9 Although it's not terribly exciting, the words 'Hello World!' should now be displayed in the Terminal. The echo command is responsible for outputting the words after it in the Terminal, as we move on you can make the echo command output to other sources.



STEP 10 Think of echo as the old BASIC Print command. It displays either text, numbers or any variables that are stored in the system, such as the current system date. Try this example: `echo Hello World! Today is $(date +%A)`. The $(date +%A) is calling the system variable that stores the current day of the week.



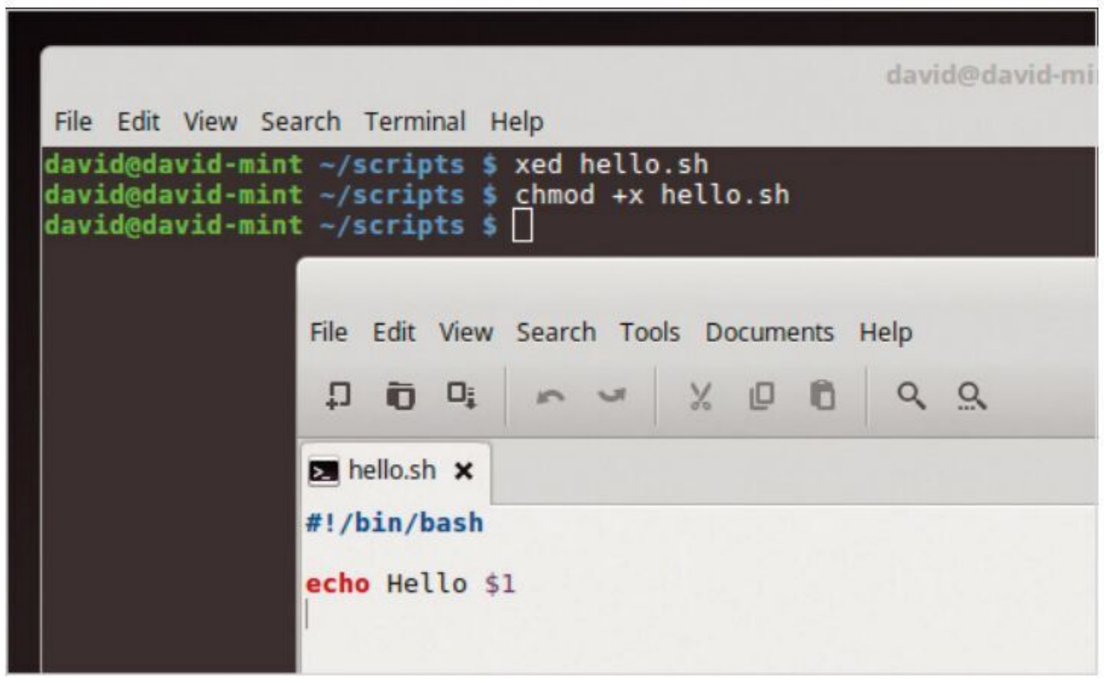
Creating Bash Scripts – Part 2

Previously we looked at creating your first Bash script, Hello World, and adding a system variable. Now you can expand these and see what you can do when you start to play around with creating your own unique variables.

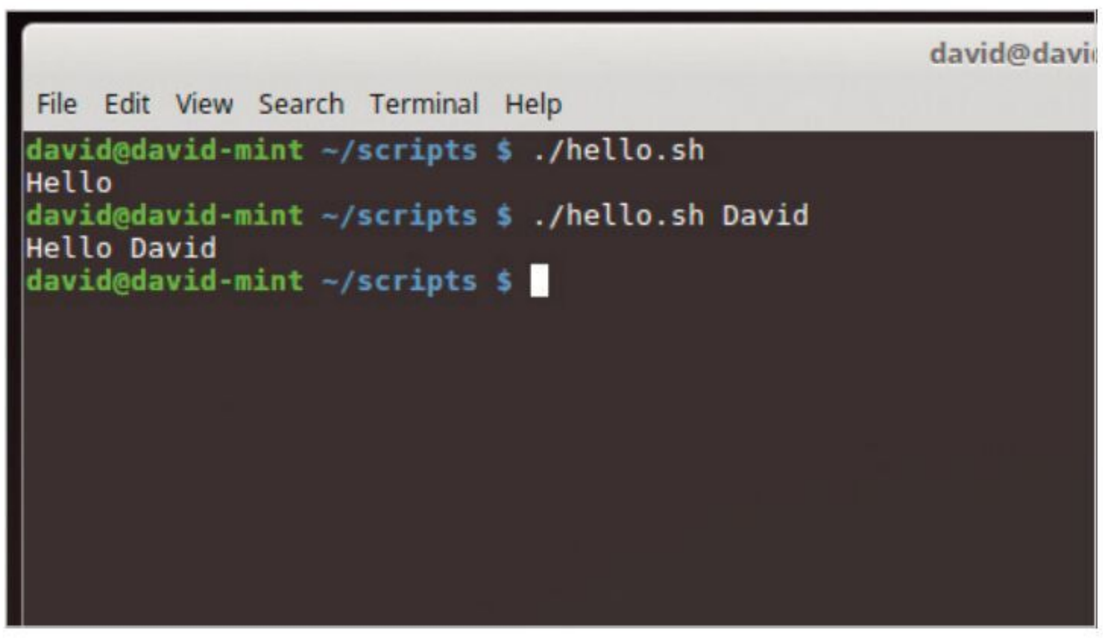
VARIABLES

Just as in every other programming language a Bash script can store and call certain variables from the system, either generic or user created.

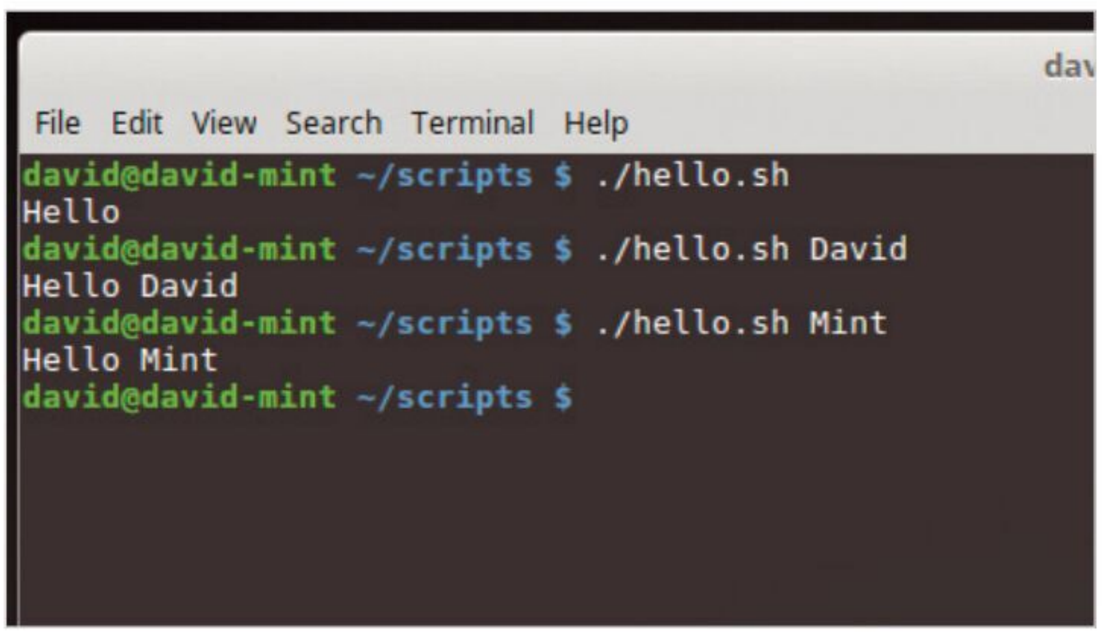
STEP 1 Let's start by creating a new script called hello.sh; `xed hello.sh`. In it enter: `#!/bin/bash`, then, echo `Hello $1`. Save the file and exit Xed. Back in the Terminal make the script executable with: `chmod +x hello.sh`.



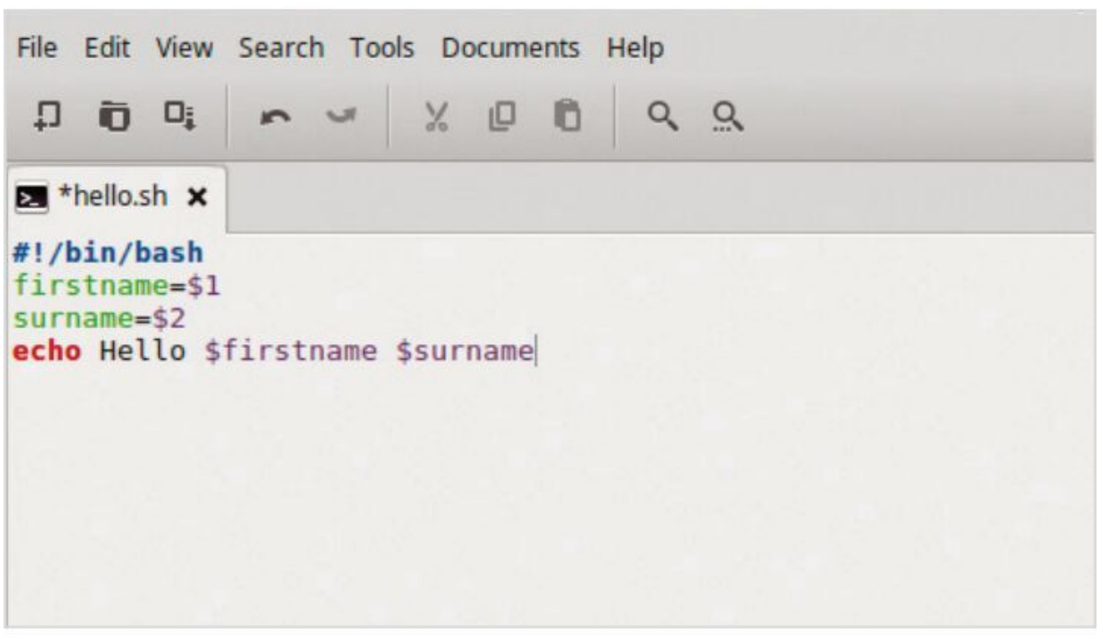
STEP 2 As the script is now executable, run it with `./hello.sh`. Now, as you probably expected a simple 'Hello' is displayed in the Terminal. However, if you then issue the command with a variable, it begins to get interesting. For example, try `./hello.sh David`.



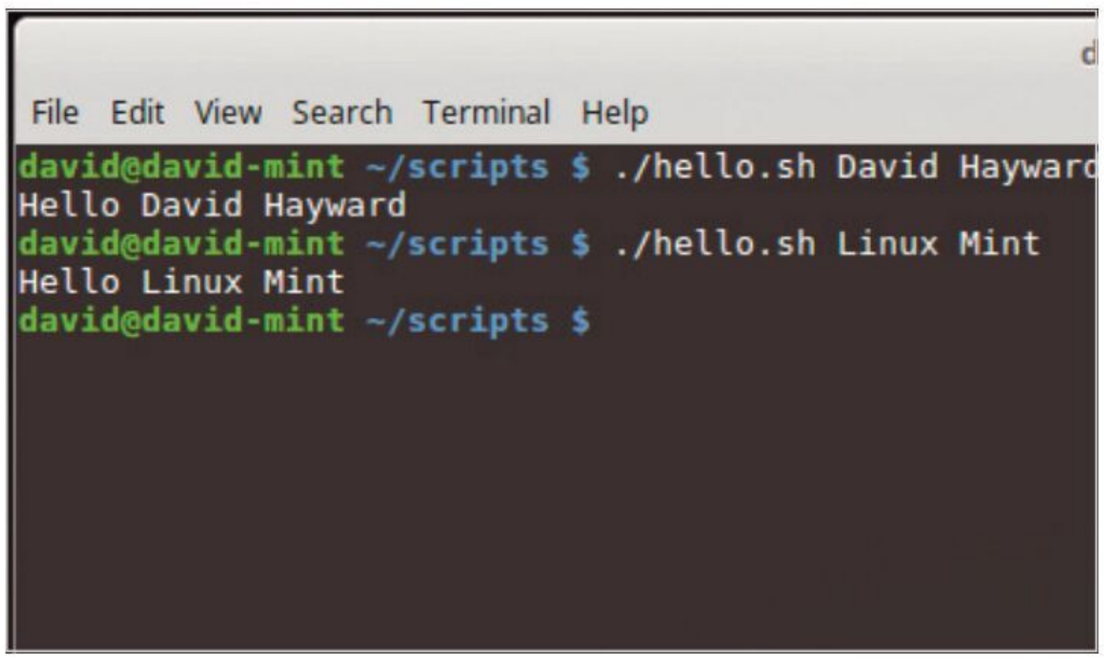
STEP 3 The output now will be Hello David. This is because Bash automatically assigns variables for the user, which are then held and passed to the script. So the variable '$1' now holds 'David'. You can change the variable by entering something different: `./hello.sh Mint`.



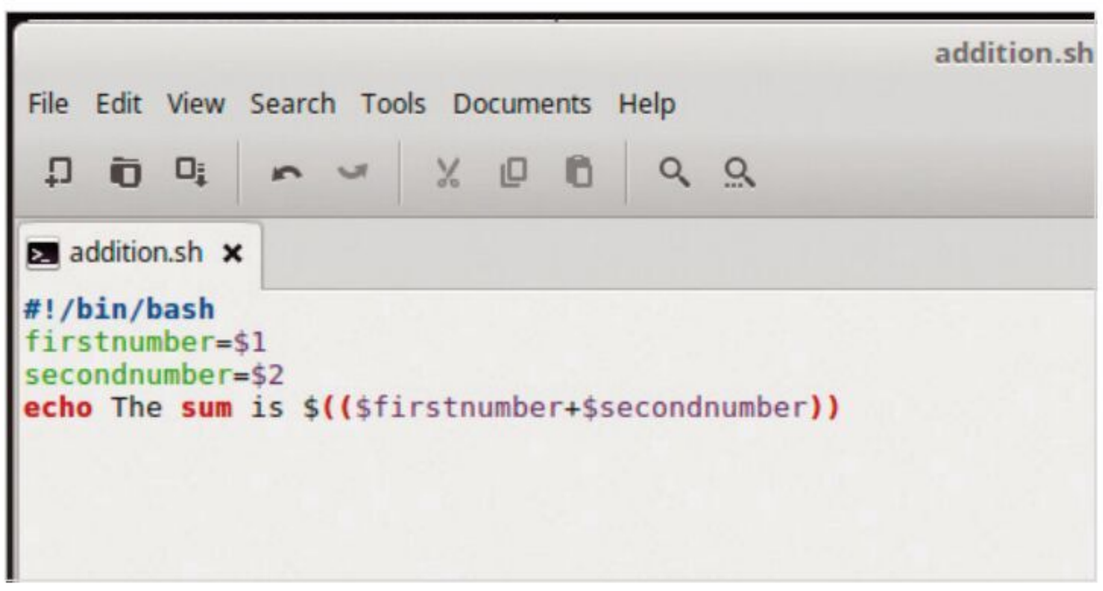
STEP 4 You can even rename variables. Modify the hello.sh script with the following: `firstname=$1, surname=$2, echo Hello $firstname $surname`. Putting each statement on a new line. Save the script and exit back into the Terminal.



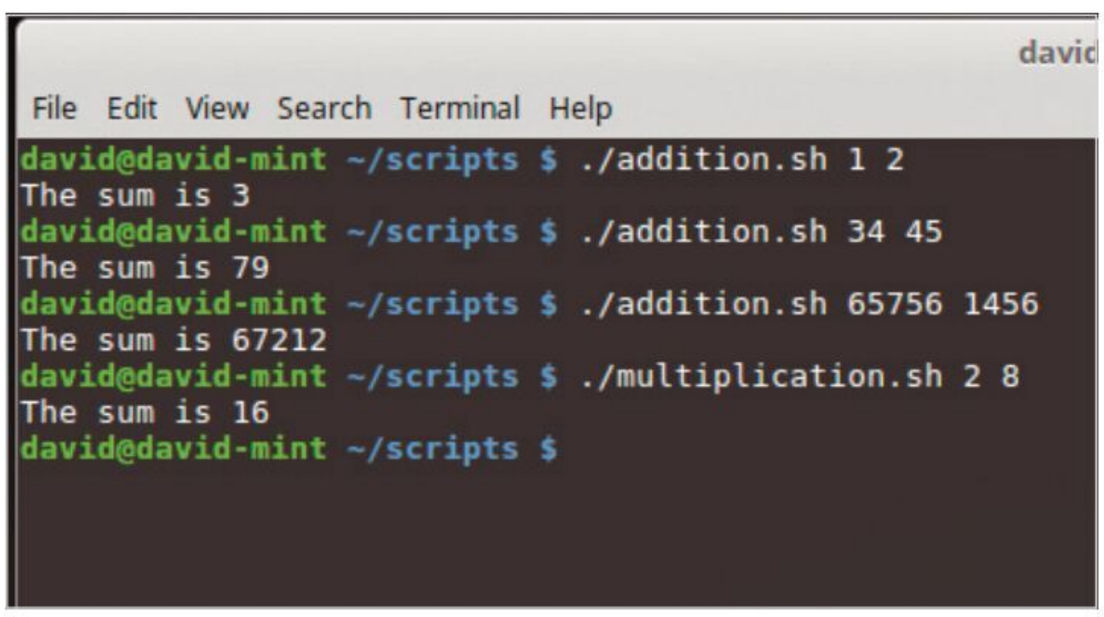
STEP 5 When you run the script now you can use two custom variables: `./hello.sh David Hayward`. Naturally change the two variables with your own name; unless you're also called David Hayward. At the moment we're just printing the contents, so let's expand the two-variable use a little.



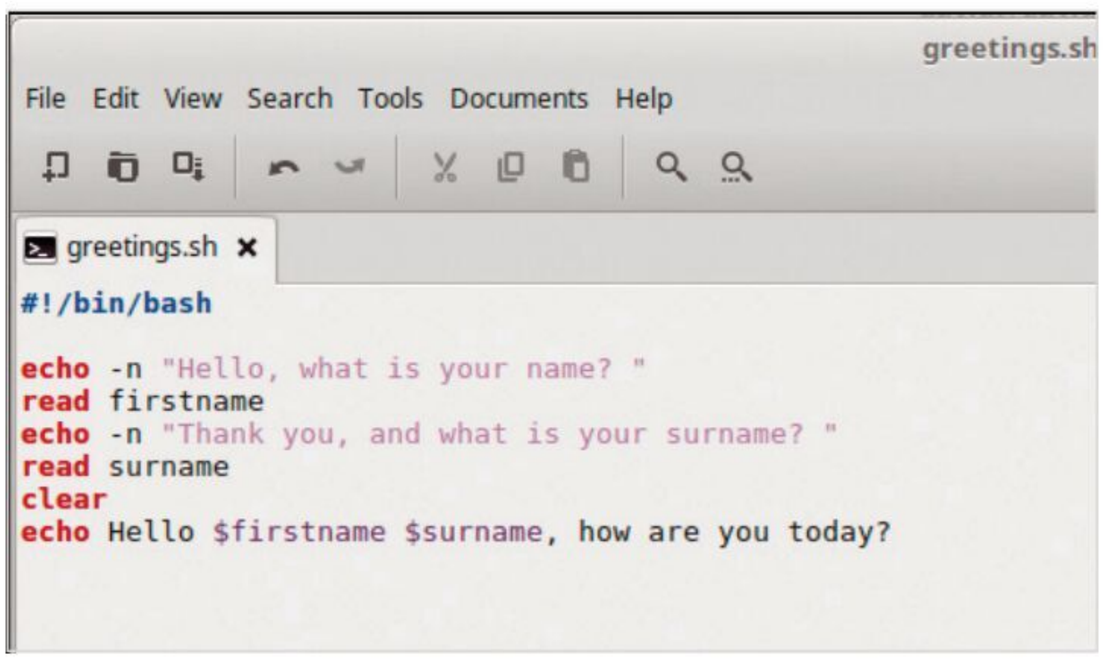
STEP 6 Create a new script called `addition.sh`, using the same format as the hello.sh script, but changing the variable names. Here we've added `firstnumber` and `secondnumber`, and used the echo command to output some simple arithmetic by placing an integer expression, `echo The sum is $(($firstnumber+$secondnumber))`. Save the script, and make it executable (`chmod +x addition.sh`).



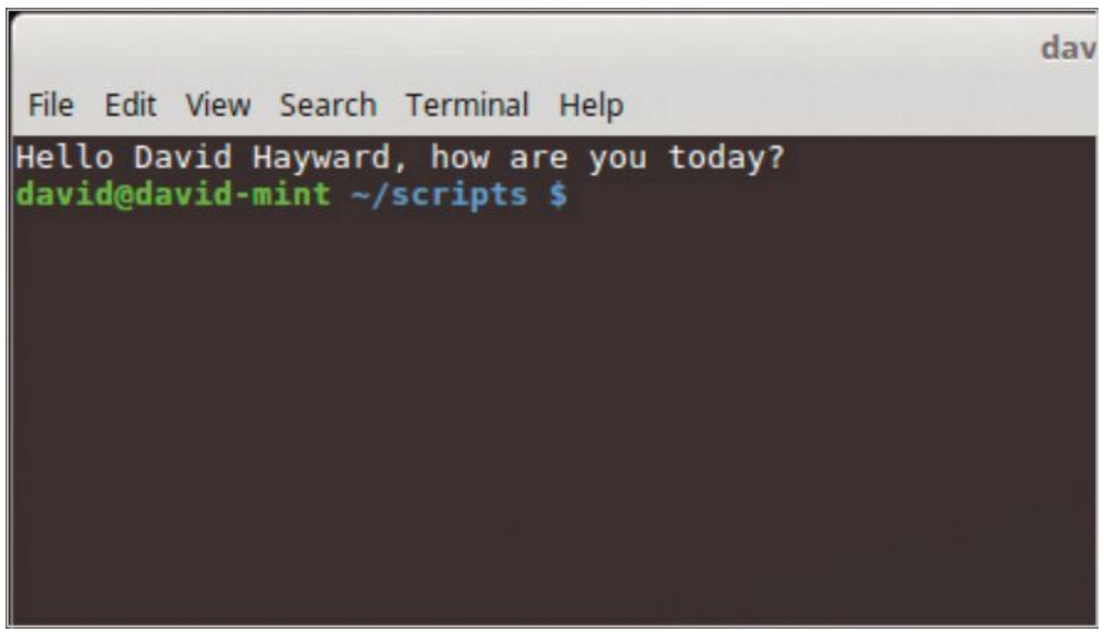
STEP 7 When you now run the addition.sh script we can enter two numbers: `./addition.sh 1 2`. The result will hopefully be 3, with the Terminal displaying 'The sum is 3'. Try it with a few different numbers and see what happens. See also if you can alter the script and rename it do multiplication, and subtraction.



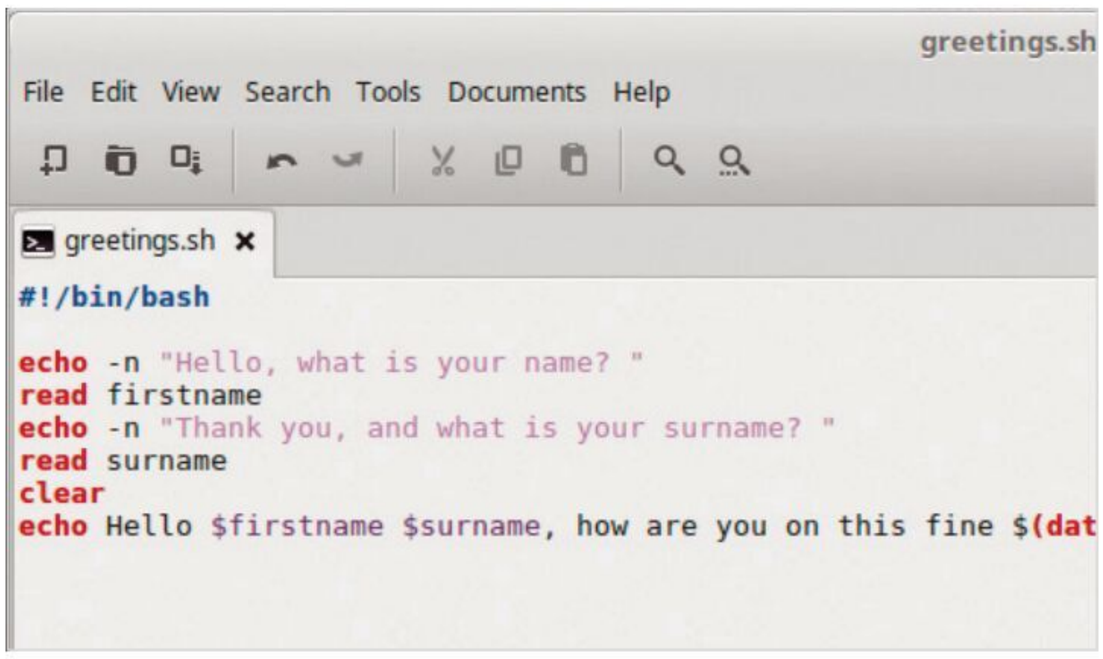
STEP 8 Let's expand things further. Create a new script called `greetings.sh`. Enter the scripting as below in the screenshot, save it and make it executable with the `chmod` command. You can see that there are a few new additions to the script now.



STEP 9 We've added a `–n` to the echo command here which will leave the cursor on the same line as the question, instead of a new line. The `read` command stores the users' input as the variables firstname and surname, to then read back later in the last `echo` line. And the `clear` command clears the screen.



STEP 10 As a final addition, let's include the date variable we used in the last section. Amend the last line of the script to read: `echo Hello $firstname $surname, how are you on this fine $(date +%A)?`. The output should display the current day of the week, calling it from a system variable.



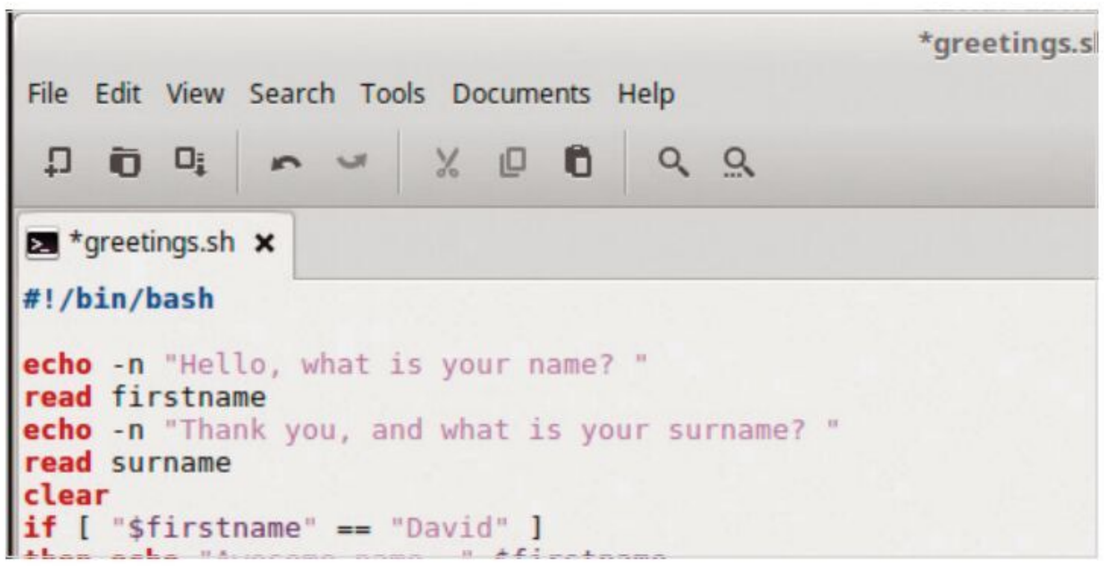
Creating Bash Scripts – Part 3

In the previous pages we looked at some very basic Bash scripting, which involved outputting text to the screen, getting a user's input, storing it and outputting that to the screen; as well as including a system variable using the Date command. Now let's combine what you've achieved so far and introduce Loops.

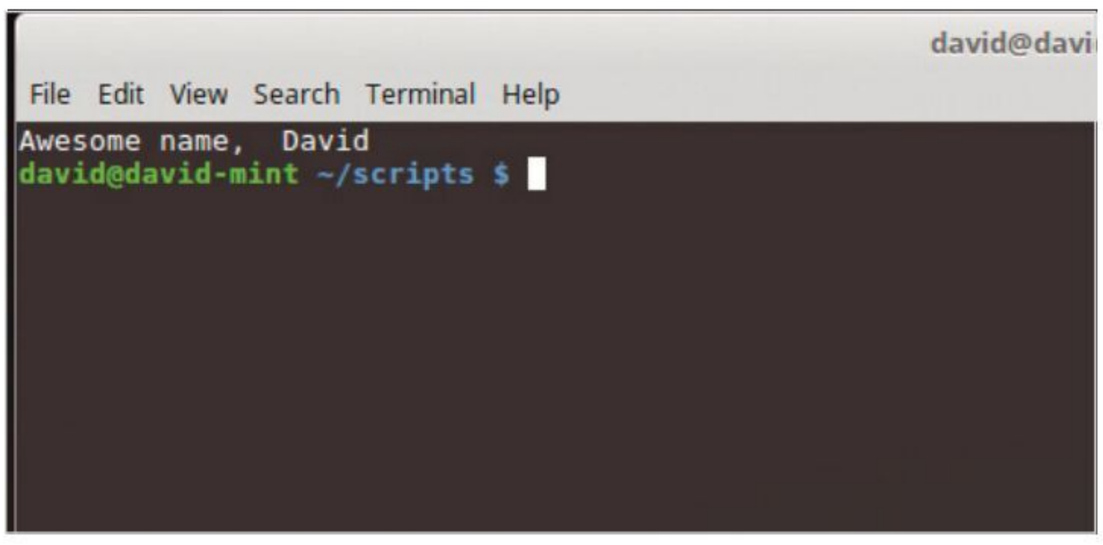
IF, THEN, ELSE

With most programming structures there will come a time where you need to loop through the commands you've entered to create better functionality, and ultimately a better program.

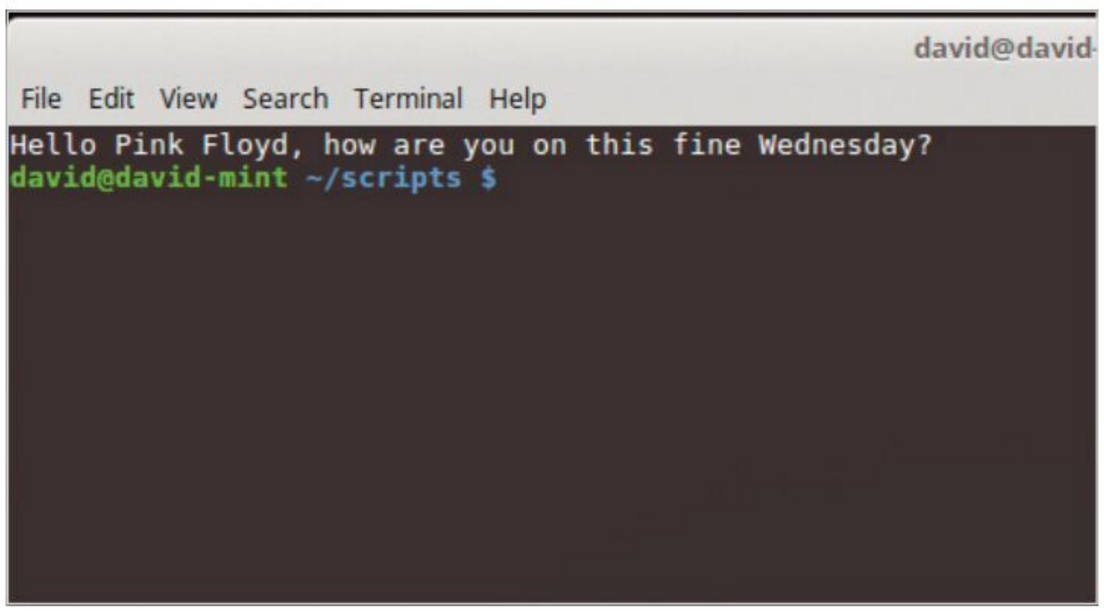
STEP 1 Let's look at the If, Then and Else statements now, which when executed correctly, compare a set of instructions and simply work out that IF something is present, THEN do something, ELSE do something different. Create a new script called `greeting2.sh` and enter the text in the screenshot below into it.



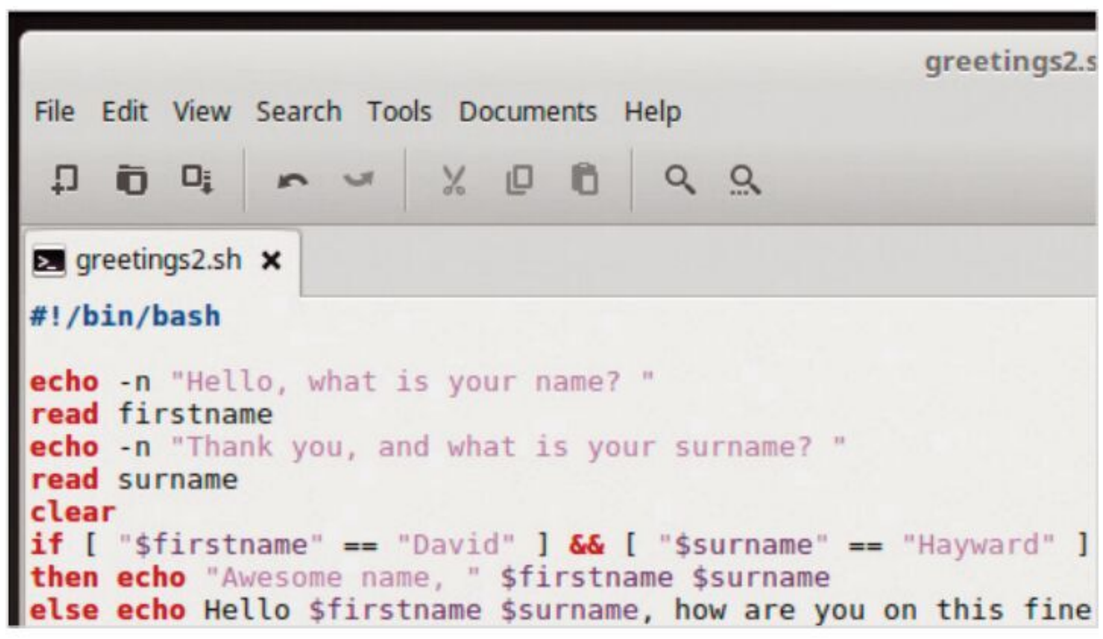
STEP 2 Greeting2.sh is a copy of greeting.sh but with a slight difference. Here we've added a loop starting at the if statement. This means, IF the variable entered is equal to David the next line, THEN, is the reaction to what happens, in this case it will output to the screen 'Awesome name,' followed by the variable (which is David).



STEP 3 The next line, ELSE, is what happens if the variable doesn't equal 'David'. In this case it simply outputs to the screen the now familiar 'Hello…'. The last line, the FI statement, is the command that will end the loop. If you have an If command without a Fi command, then you get an error.



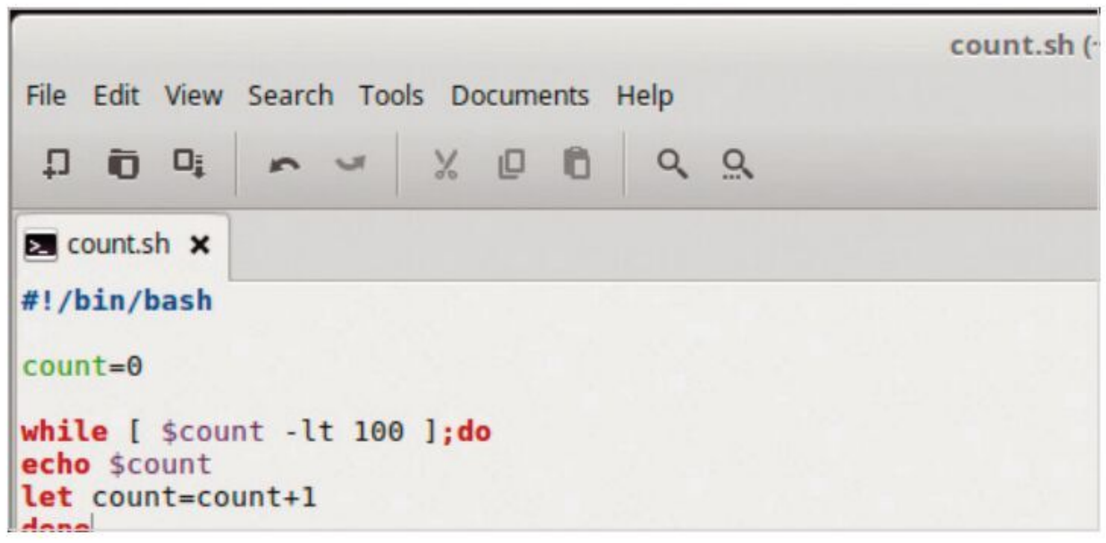
STEP 4 You can obviously play around with the script a little, changing the name variable that triggers a response; or maybe even issuing a response where the first name and surname variables match a specific variable.



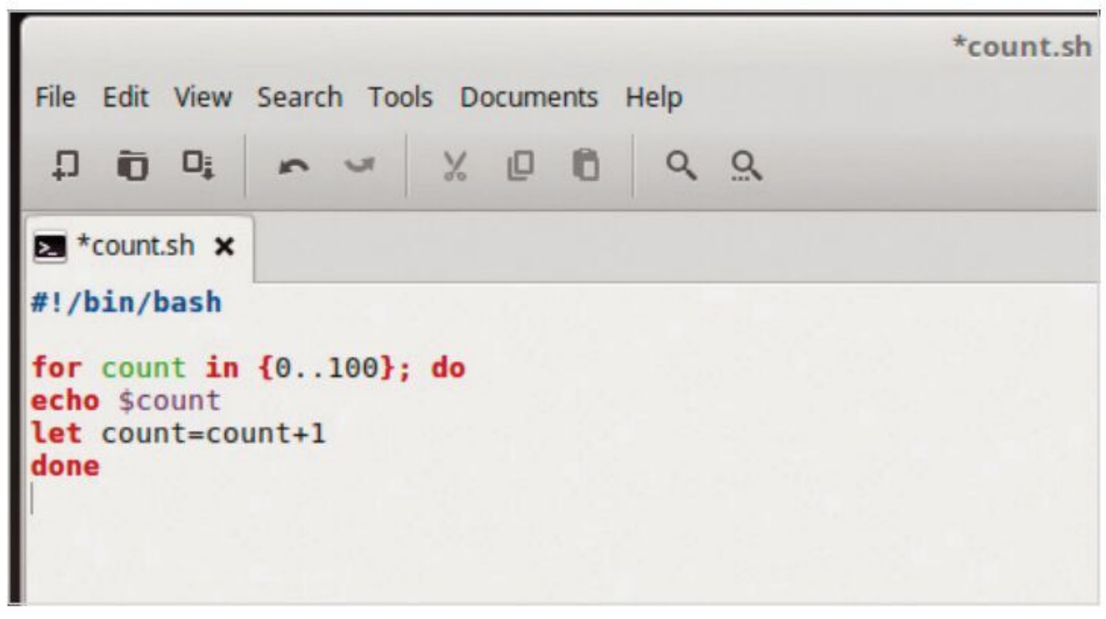
MORE LOOPING

You can loop over data using the FOR, WHILE and UNTIL statements. These can be handy if you're batch naming, copying or running a script where a counter is needed.

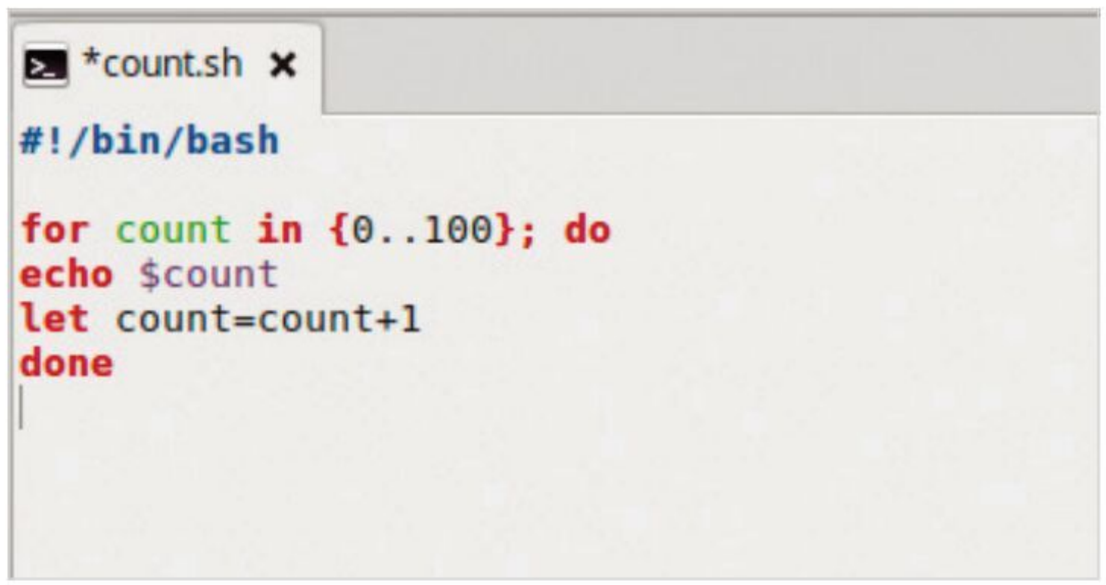
STEP 1 Create a new script called `count.sh`. Enter the text in the screenshot below, save it and make it executable. This creates the variable 'count' which at the beginning of the script equals zero. Then start the WHILE loop, which WHILE count is less than (the LT part) 100 will print the current value of count in the echo command.



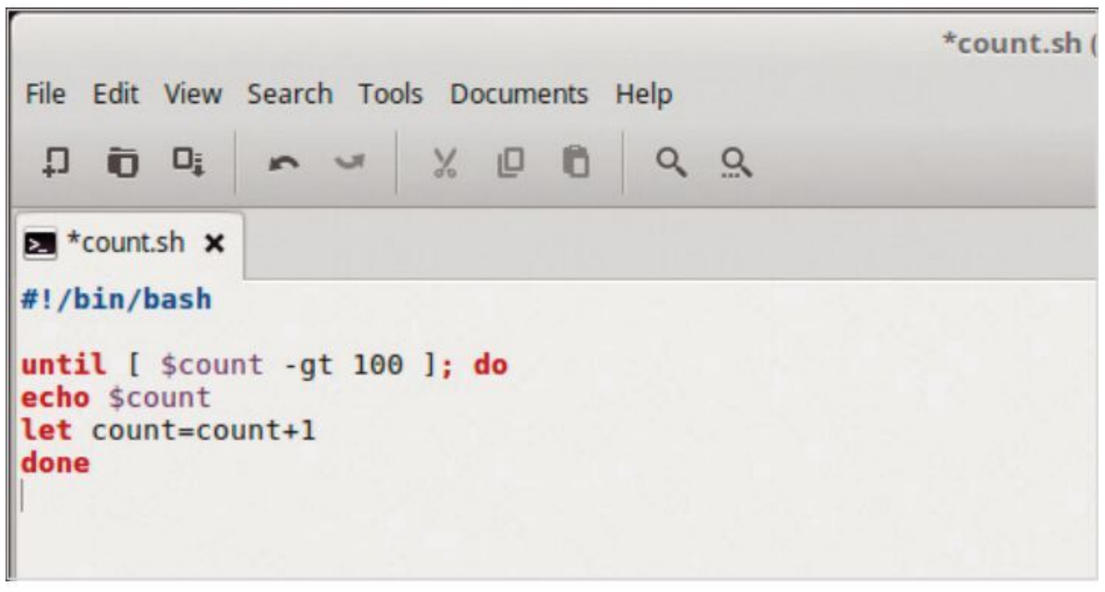
STEP 2 Executing the count.sh script will result in the numbers 0 to 99 listing down the Terminal screen; when it reaches 100 the script will end. Modifying the script with the FOR statement, makes it work in much the same way. To use it in our script, enter the text from the screenshot into the count.sh script.



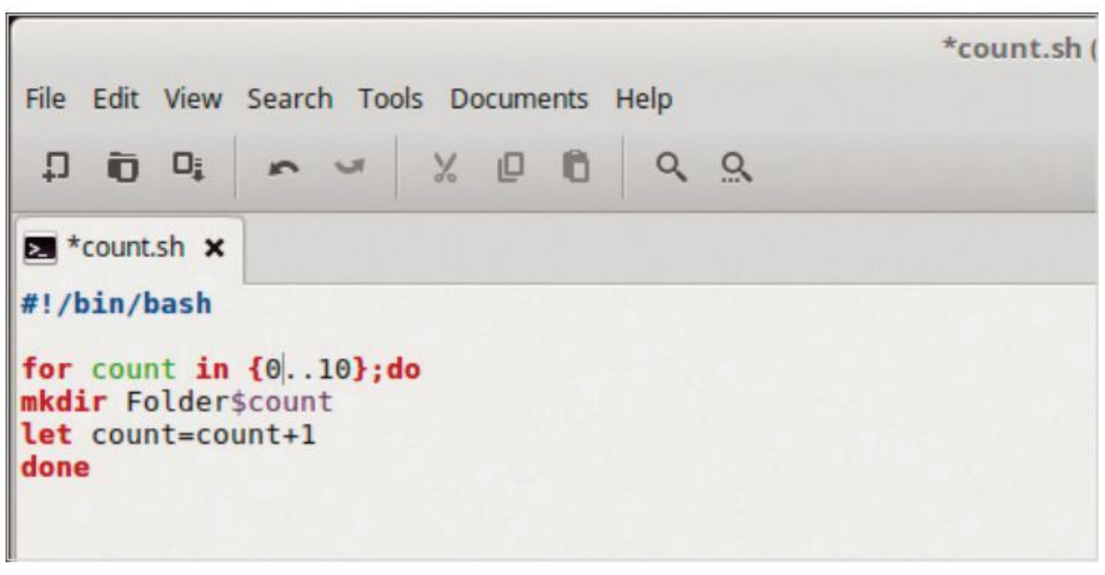
STEP 3 The addition we have here is: `for count in {0..100}; do`. Which means: FOR the variable 'count' IN the numbers from zero to one hundred, then start the loop. The rest of the script is the same. Run this script, and the same output should appear in the Terminal.



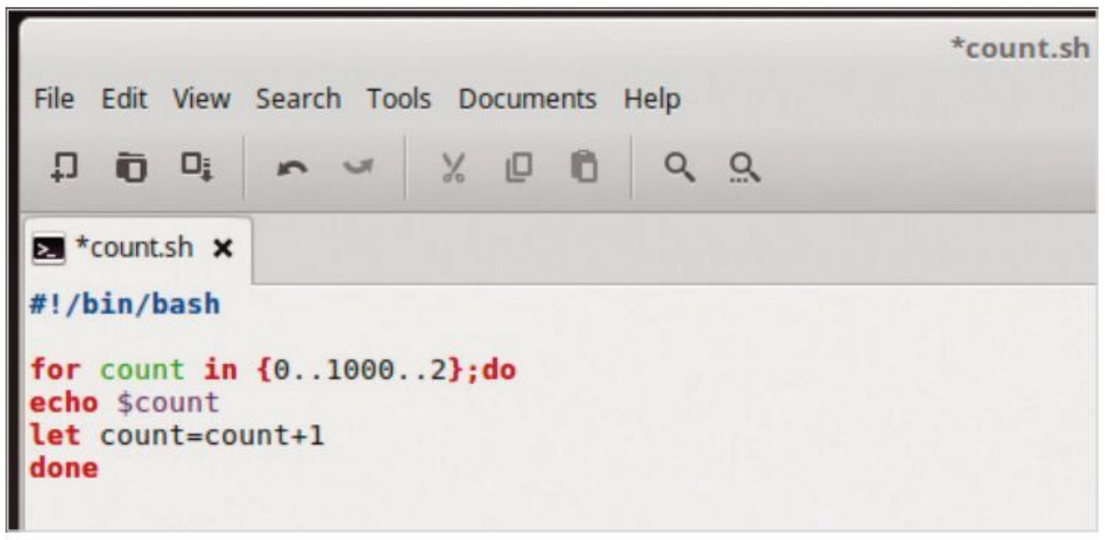
STEP 4 The UNTIL loop works much the same way as the WHILE loop only, more often than not, in reverse. So our counting to a hundred, using `UNTIL, would be: until [ $count -gt 100 ]; do`. The difference being, UNTIL count is greater than (the gt part) one hundred, keep on looping.



STEP 5 You're not limited to numbers zero to one hundred. You can, within the loop, have whatever set of commands you like and execute them as many times as you want the loop to run for. Renaming a million files, creating fifty folders etc. For example, this script will create ten folders named folder1 through to folder10 using the FOR loop.



STEP 6 Using the FOR statement once more, we can execute the counting sequence by manipulating the {0..100} part. This section of the code actually means {START..END..INCREMENT}, if there's no increment then it's just a single digit up to the END. For example, we could get the loops to count up to 1000 in two's with: `for count in {0..1000..2}; do`.



Creating Bash Scripts – Part 4

You've encountered user interaction with your scripts, asking what the user's name is and so on. You've also looked at creating loops within the script to either count or simply do something several times. Let's combine and expand some more.

CHOICES AND LOOPS

Let's bring in another command, CHOICE, along with some nested IF and ELSE statements. Start by creating a new script called `mychoice.sh`.

STEP 1

The mychoice.sh script is beginning to look a lot more complex. What we have here is a list of four choices, with three possible options. The options: Mint, Is, and Awesome will be displayed if the user presses the correct option key. If not, then the menu will reappear, the fourth choice.

mychoice.sh

```
#!/bin/bash

choice=4

echo "1. Mint"
echo "2. Is"
echo "3. Awesome"
echo -n "Please choose an option (1, 2 or 3) "

while [ $choice -eq 4 ]; do

read choice

if [ $choice -eq 1 ] ; then
    echo "You have chosen: Mint"
else
    if [ $choice -eq 2 ] ; then
        echo "You have chosen: Is"
    else
        if [ $choice -eq 3 ] ; then
            echo "You have chosen: Awesome"
        else
            echo "Please make a choice between 1 to 3"
            echo "1. Mint"
            echo "2. Is"
            echo "3. Awesome"
            echo -n "Please choose an option (1, 2 or 3) "
            choice=4
        fi
    fi
fi
done
```

STEP 2

If you follow the script through you soon get the hang of what's going on, based on what we've already covered. WHILE, IF, and ELSE, with the FI closing loop statement will run through the options and bring you back to the start if you pick the wrong option.

```
david@david-mint ~/scripts
File  Edit  View  Search  Terminal  Help
david@david-mint ~/scripts $ ./mychoice.sh
1. Mint
2. Is
3. Awesome
Please choose an option (1, 2 or 3) 1
You have chosen: Mint
david@david-mint ~/scripts $ ./mychoice.sh
1. Mint
2. Is
3. Awesome
Please choose an option (1, 2 or 3) 2
You have chosen: Is
david@david-mint ~/scripts $ ./mychoice.sh
1. Mint
2. Is
3. Awesome
Please choose an option (1, 2 or 3) 3
You have chosen: Awesome
david@david-mint ~/scripts $ ./mychoice.sh
1. Mint
2. Is
3. Awesome
```

STEP 3

You can, of course, increase the number of choices but you need to make sure that you match the number of choices to the number of IF statements. The script can quickly become a very busy screen to look at. This lengthy script is another way of displaying a menu, this time with a fancy colour scheme too.

```
#!/bin/bash

     E='echo -e';e='echo -en';trap "R;exit" 2
   ESC=$( $e "\e")
  TPUT(){ $e "\e[${1};${2}H";}
 CLEAR(){ $e "\ec";}
 CIVIS(){ $e "\e[?25l";}
  DRAW(){ $e "\e%@\e(0";}
 WRITE(){ $e "\e(B";}
  MARK(){ $e "\e[7m";}
UNMARK(){ $e "\e[27m";}
     R(){ CLEAR ;stty sane;$e "\ec\e[37;44m\e[J";};
  HEAD(){ DRAW
          for each in $(seq 1 13);do
          $E "    x                                       x"
          done
          WRITE;MARK;TPUT 1 5
          $E "BASH SELECTION MENU                        ";UNMARK;}
          i=0; CLEAR; CIVIS;NULL=/dev/null
  FOOT(){ MARK;TPUT 13 5
          printf "ENTER - SELECT,NEXT                    ";UNMARK;}
 ARROW(){ read -s -n3 key 2>/dev/null >&2
          if [[ $key = $ESC[A ]];then echo up;fi
          if [[ $key = $ESC[B ]];then echo dn;fi;}
    M0(){ TPUT  4 20; $e "Login info";}
    M1(){ TPUT  5 20; $e "Network";}
    M2(){ TPUT  6 20; $e "Disk";}
    M3(){ TPUT  7 20; $e "Routing";}
    M4(){ TPUT  8 20; $e "Time";}
    M5(){ TPUT  9 20; $e "ABOUT  ";}
    M6(){ TPUT 10 20; $e "EXIT   ";}
      LM=6
  MENU(){ for each in $(seq 0 $LM);do M${each};done;}
   POS(){ if [[ $cur == up ]];then ((i--));fi
          if [[ $cur == dn ]];then ((i++));fi
          if [[ $i -lt 0   ]];then i=$LM;fi
          if [[ $i -gt $LM ]];then i=0;fi;}
REFRESH(){ after=$((i+1)); before=$((i-1))
          if [[ $before -lt 0  ]];then before=$LM;fi
          if [[ $after -gt $LM ]];then after=0;fi
          if [[ $j -lt $i      ]];then UNMARK;M$before;else UNMARK;M$after;fi
          if [[ $after -eq 0 ]] || [ $before -eq $LM ];then
          UNMARK; M$before; M$after;fi;j=$i;UNMARK;M$before;M$after;}
  INIT(){ R;HEAD;FOOT;MENU;}
    SC(){ REFRESH;MARK;$S;$b;cur=`ARROW`;}
    ES(){ MARK;$e "ENTER = main menu ";$b;read;INIT;};INIT
while [[ "$O" != " " ]]; do case $i in
      0) S=M0;SC;if [[ $cur == "" ]];then R;$e "\n$(w        )\n";ES;fi;;
      1) S=M1;SC;if [[ $cur == "" ]];then R;$e "\n$(ifconfig )\n";ES;fi;;
      2) S=M2;SC;if [[ $cur == "" ]];then R;$e "\n$(df -h    )\n";ES;fi;;
      3) S=M3;SC;if [[ $cur == "" ]];then R;$e "\n$(route -n )\n";ES;fi;;
      4) S=M4;SC;if [[ $cur == "" ]];then R;$e "\n$(date     )\n";ES;fi;;
      5) S=M5;SC;if [[ $cur == "" ]];then R;$e "\n$($e by oTo)\n";ES;fi;;
      6) S=M6;SC;if [[ $cur == "" ]];then R;exit 0;fi;;
esac;POS;done
```

STEP 4

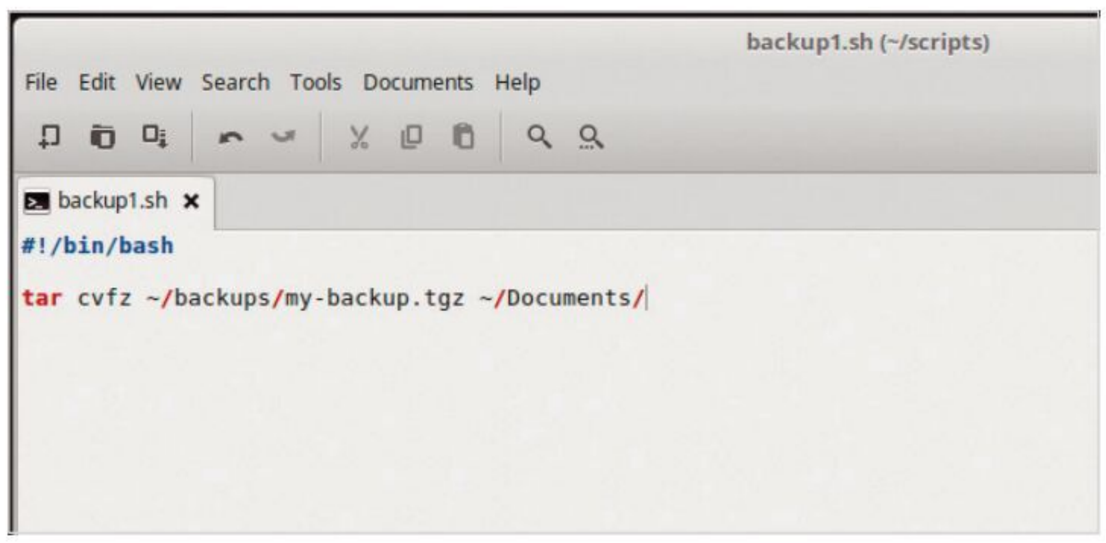
You can use the arrow keys and Enter in the menu setup in the script. Each choice is an external command that feeds back various information. Play around with the commands and choices, and see what you can come up with. It's a bit beyond what we've looked at but it gives a good idea of what can be achieved.

```
BASH SELECTION MENU

          Login info
          Network
          Disk
          Routing
          Time
          ABOUT
          EXIT

ENTER - SELECT,NEXT
```

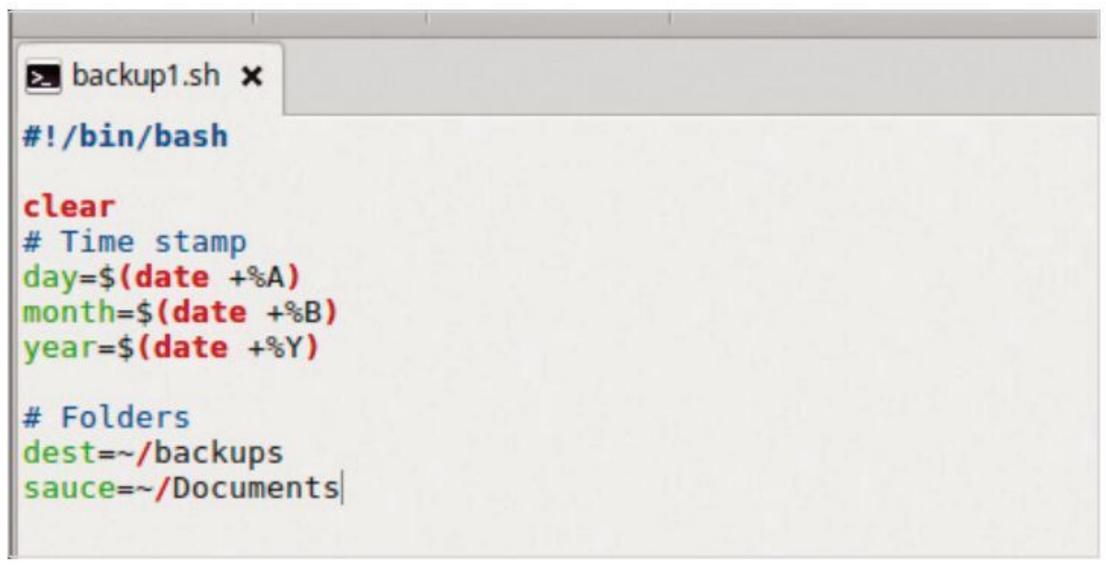
CREATING A BACKUP TASK SCRIPT

One of the most well used examples of Bash scripting is the creation of a backup routine, one that automates the task as well as adding some customisations along the way.

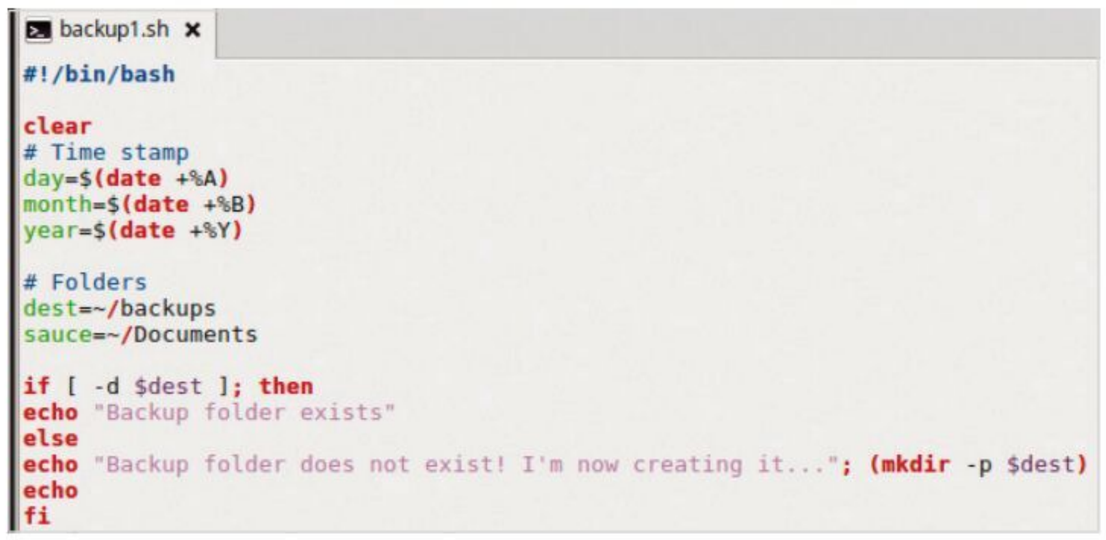
STEP 1 A very basic backup script would look something along the lines of: `#!/bin/bash, then, tar cvfz ~/backups/my-backup.tgz ~/Documents/`. This will create a compressed file backup of the ~/Documents folder, with everything in it, and put it in a folder called /backups with the name my-backup.tgz.



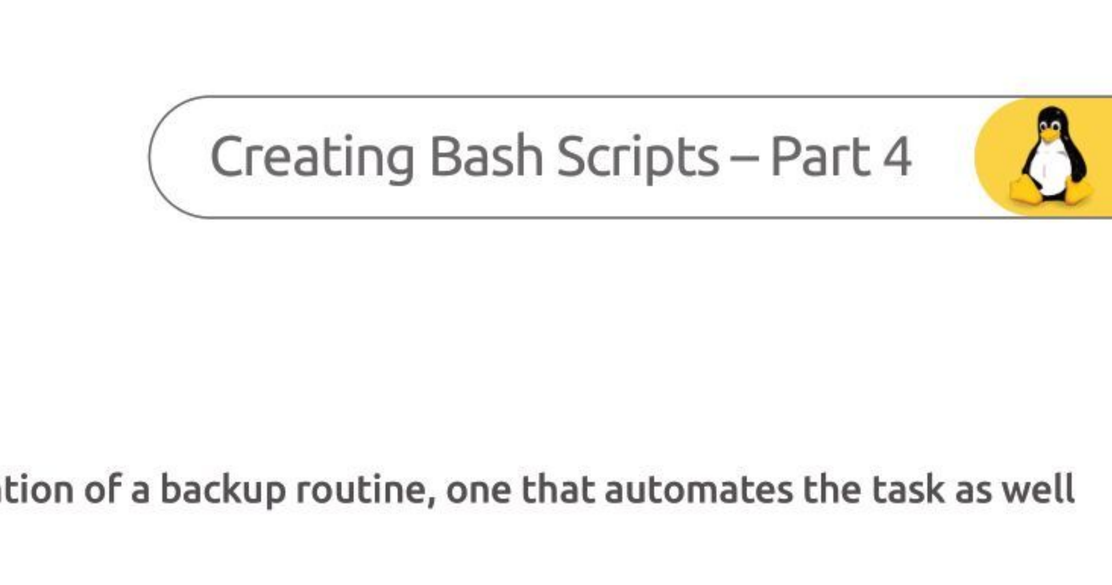
STEP 2 While perfectly fine, we can make the simple script a lot more interactive. Let's begin with defining some variables. Enter the text in the screenshot into a new backup.sh script. Notice that we've misspelt 'source' as 'sauce', this is because there's already a built-in command called 'source' hence the different spelling on our part.



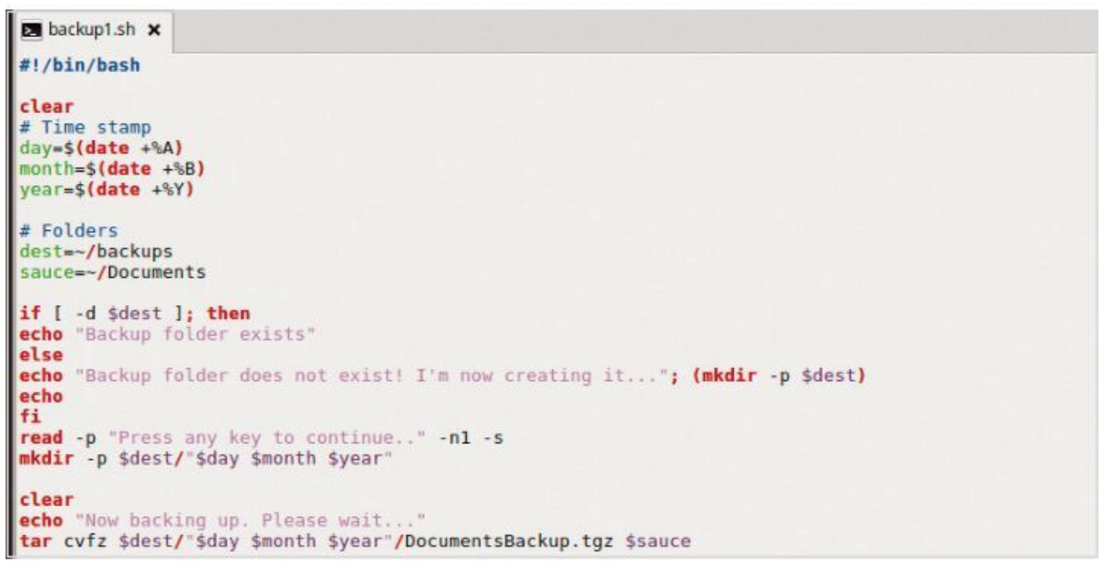
STEP 3 The previous script entries allowed you to create a Time Stamp, so you know when the backup was taken. You also created a 'dest' variable, which is the folder where the backup file will be created (~/backups). You can now add a section of code to first check if the ~/backups folder exits, if not, then it creates one.



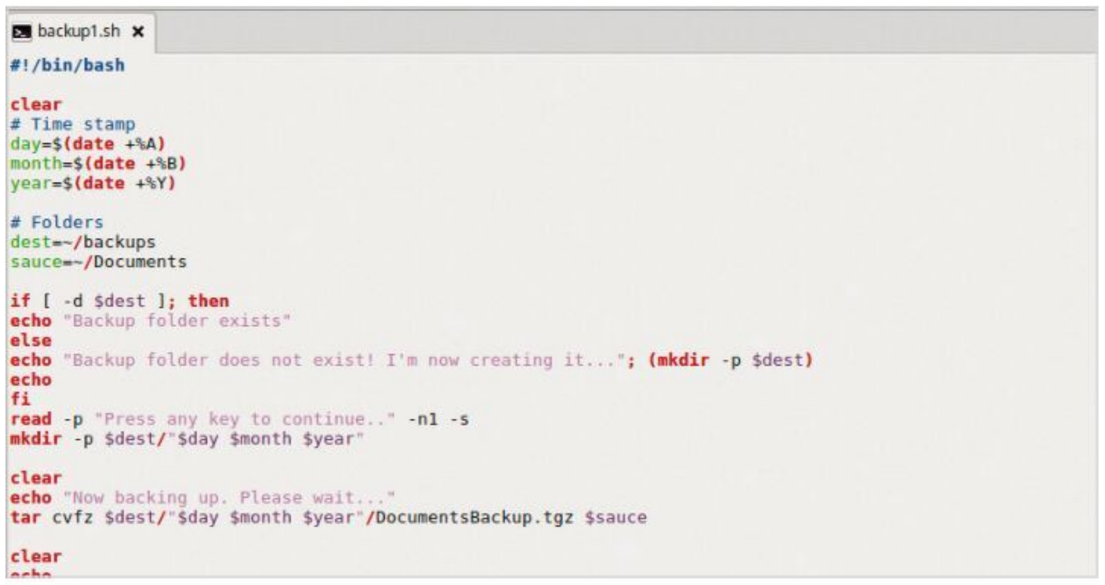
STEP 4 Once the ~/backups folder is created, we can now create a new subfolder within it based on the Time Stamp variables you set up at the beginning. Add `mkdir -p $dest/"$day $month $year"`. It's in here that you put the backup file relevant to that day/month/year.



STEP 5 With everything in place, you can now enter the actual backup routine, based on the Tar command from Step 5. Combined with the variables, you have: `tar cvfz $dest/"$day $month $year"/DocumentsBackup.tgz $sauce`. In the screenshot, we added a handy "Now backing up..." echo command.



STEP 6 Finally, you can add a friendly message: `echo "Backup complete. All done..."`. The completed script isn't too over-complex and it can be easily customised to include any folder within your Home area, as well as the entire Home area itself.



Creating Bash Scripts – Part 5

The backup script we looked at previously can be further amended to incorporate choices, user-interaction with regards to where the backup file will be copied to and so on. Automating tasks is one of the main benefits of Bash scripting, a simple script can help you out in many ways.

EASY AUTOMATION AND HANDY SCRIPTS

Entering line after line of commands to retrieve system information, find a file or rename a batch of files? A script is a better answer.

STEP 1 Let's start by creating a script to help display the Mint system information; always a handy thing to have. Create a new script called `sysinfo.sh` and enter the following into Xed, or the text editor of your choice.

sysinfo.sh

```
#!/bin/bash

# -Hostname information:
echo -e "\e[31;43m***** HOSTNAME INFORMATION *****\e[0m"
hostnamectl
echo ""

# -File system disk space usage:
echo -e "\e[31;43m***** FILE SYSTEM DISK SPACE USE *****\e[0m"
df -h
echo ""

# -Free and used memory:
echo -e "\e[31;43m***** FREE AND USED MEMORY *****\e[0m"
free
echo ""

# -System uptime and performance load:
echo -e "\e[31;43m***** SYSTEM UPTIME AND LOAD *****\e[0m"
uptime
echo ""

# -Users currently logged in:
echo -e "\e[31;43m***** CURRENT USERS *****\e[0m"
who
echo ""

# -Top five processes being used by the system:
echo -e "\e[31;43m***** TOP 5 MEMORY CONSUMING PROCESSES *****\e[0m"
ps -eo %mem,%cpu,comm --sort=-%mem | head -n 6
echo ""
echo -e "\e[1;32mDone.\e[0m"
```

STEP 2 We've included a couple of extra commands in this script. The first is the -e extension for echo, this means it'll enable echo interpretation of additional instances of a new line, as well as other special characters. The proceeding '31;43m' element enables colour for foreground and background.

```
david@david-mint ~/scripts $ ./sysinfo.sh
***** HOSTNAME INFORMATION *****
   Static hostname: david-mint
         Icon name: computer-vm
           Chassis: vm
        Machine ID: 5ab3c275b7304ed3b8aeef9ffcc37eb4
           Boot ID: 61ce1baadf934f649cf5ac809abe7e18
    Virtualization: oracle
  Operating System: Linux Mint 18.1
            Kernel: Linux 4.4.0-53-generic
      Architecture: x86-64

***** FILE SYSTEM DISK SPACE USE *****
```

STEP 3 Each of the sections runs a different Terminal command, outputting the results under the appropriate heading. You can include a lot more, such as the current aliases being used in the system, the current time and date and so on. Plus, you could also pipe all that information into a handy HTML file, ready to be viewed in a browser.

```
david@david-mint ~/scripts
File  Edit  View  Search  Terminal  Help
david@david-mint ~/scripts $ ./sysinfo.sh > sysinfo.html
david@david-mint ~/scripts $
```

STEP 4 Although there are simple Terminal commands to help you look for a particular file or folder, it's often more fun to create a script to help you. Plus, you can use that script for other non-technical users. Create a new script called `look4.sh`, entering the content from the screenshot below.

look4.sh

```
#!/bin/bash

target=~/

read name

output=$( find "$target" -iname "*.$name" 2> /dev/null )

if [[ -n "$output" ]]; then
    echo "$output"
else
    echo "No match found"
fi
```

STEP 5 When executed the script waits for input from the user, in this case the file extension, such as jpg, mp4 and so on. It's not very friendly though. Let's make it a little friendlier. Add an echo, with: `echo -n "Please enter the extension of the file you're looking for: "`, just before the read command.

*look4.sh

```
#!/bin/bash

target=~/

echo -n "Please enter the extensions of the file you're looking for: "
read name

output=$( find "$target" -iname "*.$name" 2> /dev/null )

if [[ -n "$output" ]]; then
    echo "$output"
else
    echo "No match found"
fi
```

STEP 6 Here's an interesting, fun kind of script using the app espeak. Install espeak with `sudo apt-get install espeak`, then enter the text below into a new script called `speak.sh`. As you can see it's a rehash of the first greeting script we ran. Only this time, it uses the variables in the espeak output.

speak.sh

```
#!/bin/bash

echo -n "Hello, what is your first name? "
read firstname
echo -n "Thank you, and what is your surname? "
read surname
clear
espeak "Hello $firstname $surname, how are you on this fine $(date +%A)?"
```

STEP 7 We briefly looked at putting some colours in the output for our scripts. Whilst it's too long to dig a little deeper into the colour options, here's a script that outputs what's available. Create a new script called `colours.sh` and enter the text (see below) into it.

colours.sh

```
#!/bin/bash

clear
echo -e "Normal \e[1mBold"
echo -e "Normal \e[2mDim"
echo -e "Normal \e[4mUnderlined"
echo -e "Normal \e[5mBlink"
echo -e "Normal \e[7minverted"
echo -e "Normal \e[8mHidden"
echo
echo -e "\e[0mNormal Text"
echo

echo -e "Default \e[39mDefault"
echo -e "Default \e[30mBlack"
echo -e "Default \e[31mRed"
echo -e "Default \e[32mGreen"
echo -e "Default \e[33mYellow"
echo -e "Default \e[34mBlue"
echo -e "Default \e[35mMagenta"
echo -e "Default \e[36mCyan"
echo -e "Default \e[37mLight gray"
echo -e "Default \e[90mDark gray"
echo -e "Default \e[91mLight red"
echo -e "Default \e[92mLight green"
echo -e "Default \e[93mLight yellow"
echo -e "Default \e[94mLight blue"
echo -e "Default \e[95mLight magenta"
echo -e "Default \e[96mLight cyan"
echo -e "Default \e[97mWhite"
echo
echo -e "Default \e[49mDefault"
echo -e "Default \e[40mBlack"
echo -e "Default \e[41mRed"
echo -e "Default \e[42mGreen"
echo -e "Default \e[43mYellow"
echo -e "Default \e[44mBlue"
echo -e "Default \e[45mMagenta"
echo -e "Default \e[46mCyan"
echo -e "Default \e[47mLight gray"
echo -e "Default \e[100mDark gray"
echo -e "Default \e[101mLight red"
echo -e "Default \e[102mLight green"
echo -e "Default \e[103mLight yellow"
echo -e "Default \e[104mLight blue"
echo -e "Default \e[105mLight magenta"
echo -e "Default \e[106mLight cyan"
echo -e "Default \e[107mWhite"
```

STEP 8 The output from colours.sh can, of course, be mixed together, bringing different effects depending on what you want to the output to say. For example, white text in a red background flashing (or blinking). Sadly the blinking effect doesn't work on all Terminals, so you may need to change to a different Terminal.

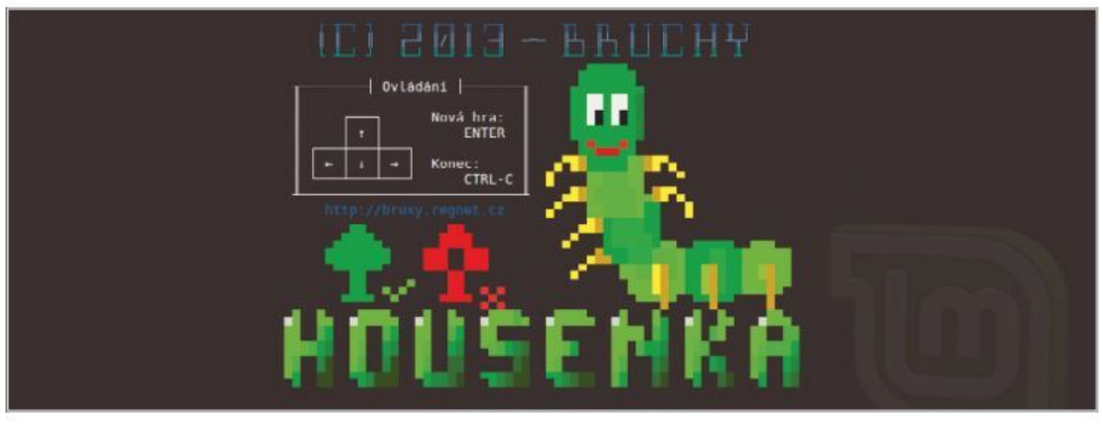
```
Normal Text

Default Default
Default Black
Default Red
Default Green
Default Yellow
Default Blue
Default Magenta
Default Cyan
Default Light gray
Default Dark gray
Default Light red
Default Light green
Default Light yellow
Default Light blue
Default Light magenta
Default Light cyan
Default White

Default Default
Default Black
Default Red
Default Green
Default Yellow
Default Blue
Default Magenta
Default Cyan
Default Light gray
Default Dark gray
Default Light red
Default Light green
Default Light yellow
Default Light blue
Default Light magenta
Default Light cyan
Default
david@david-mint ~/scripts $
```

STEP 9 Whilst we're on making fancy scripts, how about using Zenity to output a graphical interface? Enter what you see below into a new script, `mmenu.sh`. Make it executable and then run it. You should have a couple of dialogue boxes appear, followed by a final message.

mmenu.sh

```
#!/bin/bash

firstname=$(zenity --entry --title="Your Name" --text="What is your first name?")

surname=$(zenity --entry --title="Your Name" --text="What is your first surname?")

zenity --info --title="Hello!" --text="Welcome to Linux Mint.\n\n Have fun, $firstname $surname."
```

STEP 10 While gaming in a Bash script isn't something that's often touched upon, it is entirely possible, albeit, a little basic. If you fancy playing a game, enter `wget http://bruxy.regnet.cz/linux/housenka/housenka.sh`, make the script executable and run it. It's in Polish, written by Martin Bruchanov but we're sure you can modify it. Hint: the title screen is in Base64.

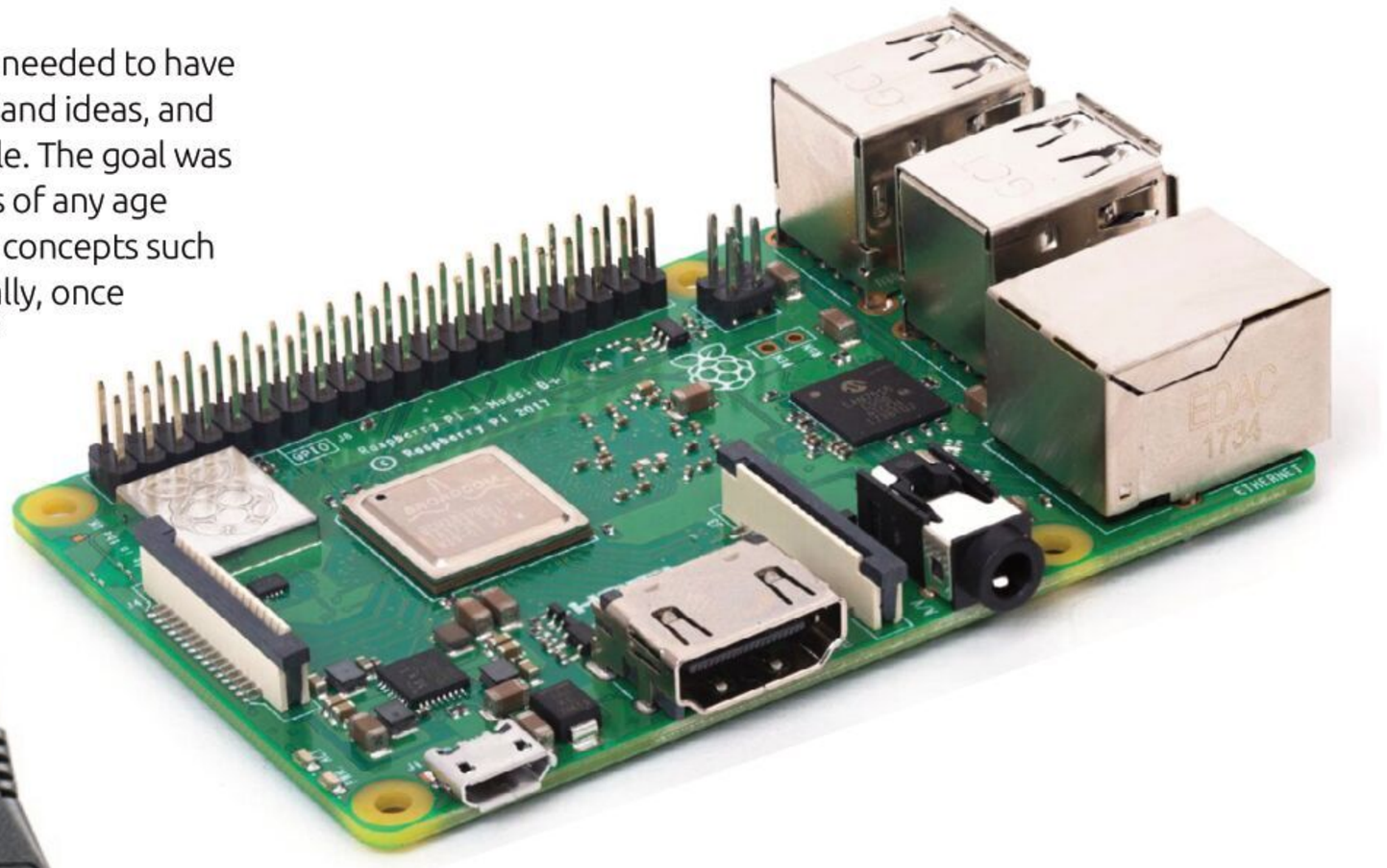


Pi x Linux = The Perfect Combination

The Raspberry Pi is a remarkable piece of hardware, and to help drive its potential it needs a remarkable operating system. Thankfully, Linux is the default and recommended OS of choice for the Pi, and together they make a winning team.

When the Raspberry Pi was in its developmental stages its designers needed to ensure that they were creating a piece of hardware that could offer much more than simply being a cheap, but small, computer.

The Pi needed to be flexible with what it could do, it needed to have room to grow into more ambitious project concepts and ideas, and it needed to do so in as easy to use fashion as possible. The goal was to create a universal learning platform, that students of any age could expand on and tinker with, while learning new concepts such as programming, electronics, and computing. Naturally, once the hardware was developed, the only real choice of operating system was of course, Linux.



The versatility of Linux is legendary. This incredible core OS is so malleable that it can be steered toward near any aspect of computing, from supercomputing to robotics, the space industry to more terrestrial engineering; education and science, manufacturing and the Internet. Think of an industry, and you will likely find a Linux installation somewhere in the background keeping it all together.

Raspbian is the recommended operating system for the Raspberry Pi, a customised Debian-based distribution that comes packed with a collection of useful tools that cater for coding, electronics, and general desktop computing duties. Alongside the apps are pre-loaded modules to help get the most from each of the programming languages you decide to learn and use, as well as software to hardware modules that will enable you to power and use the hardware specific items unique to the Pi. For example, there are Python modules that interact with the Raspberry Pi's 40-pin GPIO, allowing you to create content for any of the Hardware Attached on Top devices.

This combination is what makes the Raspberry Pi an excellent base of operations to learn not only coding on, but also Linux in general. Raspbian, being Debian-based, will be able to run any of the Terminal commands listed in this book, as Linux Mint and even Ubuntu are also Debian-based. And since the Raspberry is so small, and costs very little setup, you're able to have both your regular, Windows or macOS computers, and have a Raspberry Pi as a headless (a powered device that doesn't need a keyboard, mouse or monitor attached, as you connect to it remotely) computer from which you can connect to and learn how to use Linux and how to code.

WHICH PI?

There are several Raspberry Pi models available, with each offering something slightly different from the others. The most recent Pi released is the Raspberry Pi 3 Model B+, and while this model is slightly more expensive than some of the other examples, it is the most powerful and feature-rich Pi available. Overall, it's probably best to start experimenting with the Pi using the Pi 3 Model B+, then moving on to one of the other models as you develop your skills and focus on a particular project, such as the need to use one of the Pi Zero models, with a smaller footprint and WiFi enabled connectivity.

Raspberry Pi 3 Model A+

Raspberry Pi 3 Model B+

Raspberry Pi 3 Model B

Raspberry Pi 2 Model B

Raspberry Pi 1 Model A+

Raspberry Pi Zero

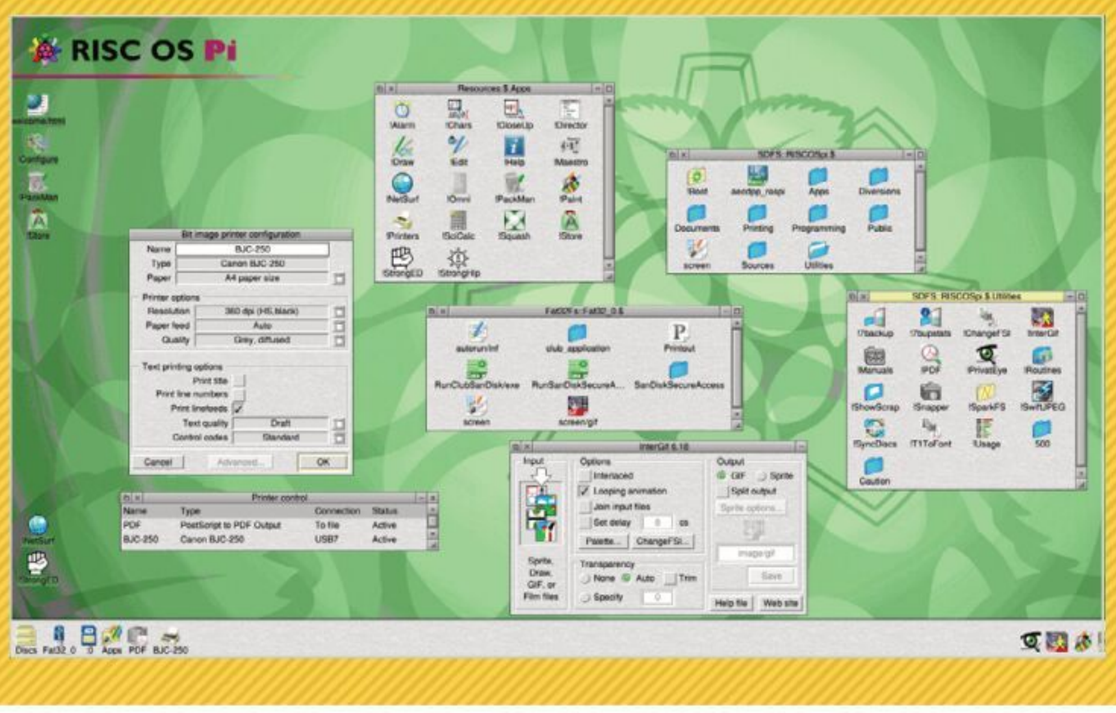
Raspberry Pi 1 Model B+

Raspberry Pi Zero W

BEYOND RASPBIAN

The flexibility of the Raspberry Pi's ARM processor means that it's capable of running other operating systems beyond Raspbian. Still keeping with Linux, you can instead install Ubuntu MATE, Pidora (a Fedora-based distribution), Lakka, PiPlay (a retro emulation distribution) and Arch Linux ARM. There are also systems based loosely on the Linux kernel, such as Android, Minibian, and Chromium OS.

There's Windows 10 IoT Core, FreeBSD, RISC OS PI, Plan 9 and, remarkably, AROS – an Amiga OS clone. Needless to say, that once you've finished experimenting with one version of Linux, just as you would with a desktop version of Linux, you can hop to another on the Raspberry Pi and see how that one works, and whether it will work for you.



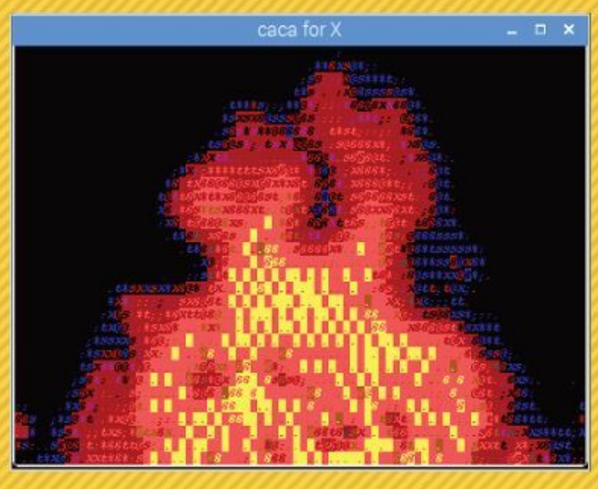
PI, LINUX AND CODING

As previously mentioned, the Raspberry Pi is an excellent code base, and with Linux as the backbone you're able to start learning how to code in a multitude of different programming languages.

Bash scripting works perfectly, since Raspbian is Linux, and Debian-based, and you can easily expand your scripts to encompass much of the Pi's functionality. Aside from creating backup scripts, you can also create scripts that can access the GPIO pins on the Pi, and in so be able to control LEDs, and even more complex HATs.

Python is by far the most popular choice for beginners, and Raspbian comes pre-installed with everything you will need to get the most from your Python experience. There are countless pre-loaded modules, as well as the most recent stable release of the language.

C++ is one of the most powerful programming languages to learn. It's used for games, apps, and even entire operating systems. The Raspberry Pi comes with a great C++ editor, that's easy to use and can help you develop amazing content. Whichever way you decide to take your coding and Linux adventure, the Raspberry Pi is an excellent platform from which to begin on.



Command Line Quick Reference

When you start using Linux full time, you will quickly realise that the graphical interfaces of Ubuntu, Mint, etc. are great for many tasks but not great for all tasks. Understanding how to use the command line not only builds your understanding of Linux but also improves your knowledge of coding and programming in general. Our command line quick reference guide is designed to help you master Linux quicker.

TOP 10 COMMANDS

These may not be the most common commands used by everyone but they will certainly feature frequently for many users of Linux and the command line.

cd

The `cd` command is one of the commands you will use the most at the command line in Linux. It allows you to change your working directory. You use it to move around within the hierarchy of your file system. You can also use chdir.

mv

The `mv` command moves a file to a different location or renames a file. For example `mv file sub` renames the original file to `sub`. `mv sub ~/Desktop` moves the file 'sub' to your desktop directory but does not rename it. You must specify a new filename to rename a file.

ls

The `ls` command shows you the files in your current directory. Used with certain options, it lets you see file sizes, when files where created and file permissions. For example, `ls ~` shows you the files that are in your home directory.

chown

The `chown` command changes the user and/or group ownership of each given file. If only an owner (a user name or numeric user ID) is given, that user is made the owner of each given file, and the files' group is not changed.

cp

The `cp` command is used to make copies of files and directories. For example, `cp file sub` makes an exact copy of the file whose name you entered and names the copy sub but the first file will still exist with its original name.

cmod

The `chmod` command changes the permissions on the files listed. Permissions are based on a fairly simple model. You can set permissions for user, group and world and you can set whether each can read, write and or execute the file.

pwd

The `pwd` command prints the full pathname of the current working directory (pwd stands for "print working directory"). Note that the GNOME terminal also displays this information in the title bar of its window.

rm

The `rm` command removes (deletes) files or directories. The removal process unlinks a filename in a filesystem from data on the storage device and marks that space as usable by future writes. In other words, removing files increases the amount of available space on your disk.

clear

The `clear` command clears your screen if this is possible. It looks in the environment for the terminal type and then in the terminfo database to figure out how to clear the screen. This is equivalent to typing Control-L when using the bash shell.

mkdir

Short for "make directory", `mkdir` is used to create directories on a file system, if the specified directory does not already exist. For example, `mkdir work` creates a work directory. More than one directory may be specified when calling mkdir.

USEFUL HELP/INFO COMMANDS

The following commands are useful for when you are trying to learn more about the system or program you are working with in Linux. You might not need them every day, but when you do, they will be invaluable.

free

The `free` command displays the total amount of free and used physical and swap memory in the system. For example, `free -m` gives the information using megabytes.

sed

The `sed` command opens a stream editor. A stream editor is used to perform text transformations on an input stream: a file or input from a pipeline.

df

The `df` command displays filesystem disk space usage for all partitions. The command `df -h` is probably the most useful (the -h means human-readable).

adduser

The `adduser` command adds a new user to the system. Similarly, the addgroup command adds a new group to the system.

top

The `top` program provides a dynamic real-time view of a running system. It can display system summary information, as well as a list of processes.

deluser

The `deluser` command removes a user from the system. To remove the user's files and home directory, you need to add the `-remove-home` option.

unmame-a

The `uname` command with the `-a` option prints all system information, including machine name, kernel name, version and a few other details.

delgroup

The `delgroup` command removes a group from the system. You cannot remove a group that is the primary group of any users.

ps

The `ps` command allows you to view all the processes running on the machine. Every operating system's version of ps is slightly different but all do the same thing.

man man

The `man man` command brings up the manual entry for the man command, which is a great place to start when using it.

grep

The `grep` command allows you to search inside a number of files for a particular search pattern and then print matching lines. An example would be: `grep blah file`.

man intro

The `man intro` command is especially useful. It displays the Introduction to User Commands, which is a well written, fairly brief introduction to the Linux command line.

A-Z of Linux Commands

There are literally thousands of Linux commands, so while this is not a complete A-Z, it does contain many of the commands you will most likely need. You will probably find that you end up using a smaller set of commands over and over again but having an overall knowledge is still very useful.

A

`adduser`	Add a new user
`arch`	Print machine architecture
`awk`	Find and replace text within file(s)

B

`bc`	An arbitrary precision calculator language

C

`cat`	Concatenate files and print on the standard output
`chdir`	Change working directory
`chgrp`	Change the group ownership of files
`chroot`	Change root directory
`cksum`	Print CRC checksum and byte counts
`cmp`	Compare two files
`comm`	Compare two sorted files line by line
`cp`	Copy one or more files to another location
`crontab`	Schedule a command to run at a later time
`csplit`	Split a file into context-determined pieces
`cut`	Divide a file into several parts

D

`date`	Display or change the date & time
`dc`	Desk calculator
`dd`	Data Dump, convert and copy a file
`diff`	Display the differences between two files
`dirname`	Convert a full path name to just a path
`du`	Estimate file space usage

E

`echo`	Display message on screen
`ed`	A line oriented text editor (edlin)
`egrep`	Search file(s) for lines that match an extended expression
`env`	Display, set or remove environment variables
`expand`	Convert tabs to spaces
`expr`	Evaluate expressions

F

`factor`	Print prime factors
`fdisk`	Partition table manipulator for Linux
`fgrep`	Search file(s) for lines that match a fixed string
`find`	Search for files that meet a desired criteria
`fmt`	Reformat paragraph text
`fold`	Wrap text to fit a specified width
`format`	Format disks or tapes
`fsck`	Filesystem consistency check and repair

G

`gawk`	Find and Replace text within file(s)
`grep`	Search file(s) for lines that match a given pattern
`groups`	Print group names a user is in
`gzip`	Compress or decompress named file(s)

H

`head`	Output the first part of file(s)
`hostname`	Print or set system name

I

`id`	Print user and group ids
`info`	Help info
`install`	Copy files and set attributes

J

`join`	Join lines on a common field

K

`kill`	Stop a process from running

L

`less`	Display output one screen at a time
`ln`	Make links between files
`locate`	Find files

`logname`	Print current login name
`lpc`	Line printer control program
`lpr`	Off line print
`lprm`	Remove jobs from the print queue

M

`man`	See Help manual
`mkdir`	Create new folder(s)
`mkfifo`	Make FIFOs (named pipes)
`mknod`	Make block or character special files
`more`	Display output one screen at a time
`mount`	Mount a file system

N

`nice`	Set the priority of a command or job
`nl`	Number lines and write files
`nohup`	Run a command immune to hangups

P

`passwd`	Modify a user password
`paste`	Merge lines of files
`pathchk`	Check file name portability
`pr`	Convert text files for printing
`printcap`	Printer capability database
`printenv`	Print environment variables
`printf`	Format and print data

Q

`quota`	Display disk usage and limits
`quotacheck`	Scan a file system for disk usage
`quotactl`	Set disk quotas

R

`ram`	Ram disk device
`rcp`	Copy files between two machines
`rm`	Remove files
`rmdir`	Remove folder(s)
`rpm`	Remote Package Manager
`rsync`	Remote file copy (synchronise file trees)

S

`screen`	Terminal window manager
`sdiff`	Merge two files interactively
`select`	Accept keyboard input
`seq`	Print numeric sequences
`shutdown`	Shutdown or restart linux
`sleep`	Delay for a specified time
`sort`	Sort text files
`split`	Split a file into fixed-size pieces
`su`	Substitute user identity
`sum`	Print a checksum for a file
`symlink`	Make a new name for a file
`sync`	Synchronise data on disk with memory

T

`tac`	Concatenate and write files in reverse
`tail`	Output the last part of files
`tar`	Tape Archiver
`tee`	Redirect output to multiple files
`test`	Evaluate a conditional expression
`time`	Measure Program Resource Use
`touch`	Change file timestamps
`top`	List processes running on the system
`traceroute`	Trace Route to Host
`tr`	Translate, squeeze and or delete characters
`tsort`	Topological sort

U

`umount`	Unmount a device
`unexpand`	Convert spaces to tabs
`uniq`	Uniquify files
`units`	Convert units from one scale to another
`unshar`	Unpack shell archive scripts
`useradd`	Create new user account
`usermod`	Modify user account
`users`	List users currently logged in

V

`vdir`	Verbosely list directory contents (`ls -l -b')

W

`watch`	Execute or display a program periodically
`wc`	Print byte, word, and line counts
`whereis`	Report all known instances of a command
`which`	Locate a program file in the user's path
`who`	Print all usernames currently logged in
`whoami`	Print the current user id and name

X

`xargs`	Execute utility, passing constructed argument list(s)

Y

`yes`	Print a string until interrupted

DID YOU KNOW... that NASA's state-of-the-art supercomputing cluster, Pleiades, is powered by Linux? Pleiades consists of 160 racks (11,440 nodes), 245,536 CPU cores, a total memory count of 935TB, 184,320 NVIDIA CUDA cores, and over six miles of fibre optic cabling.

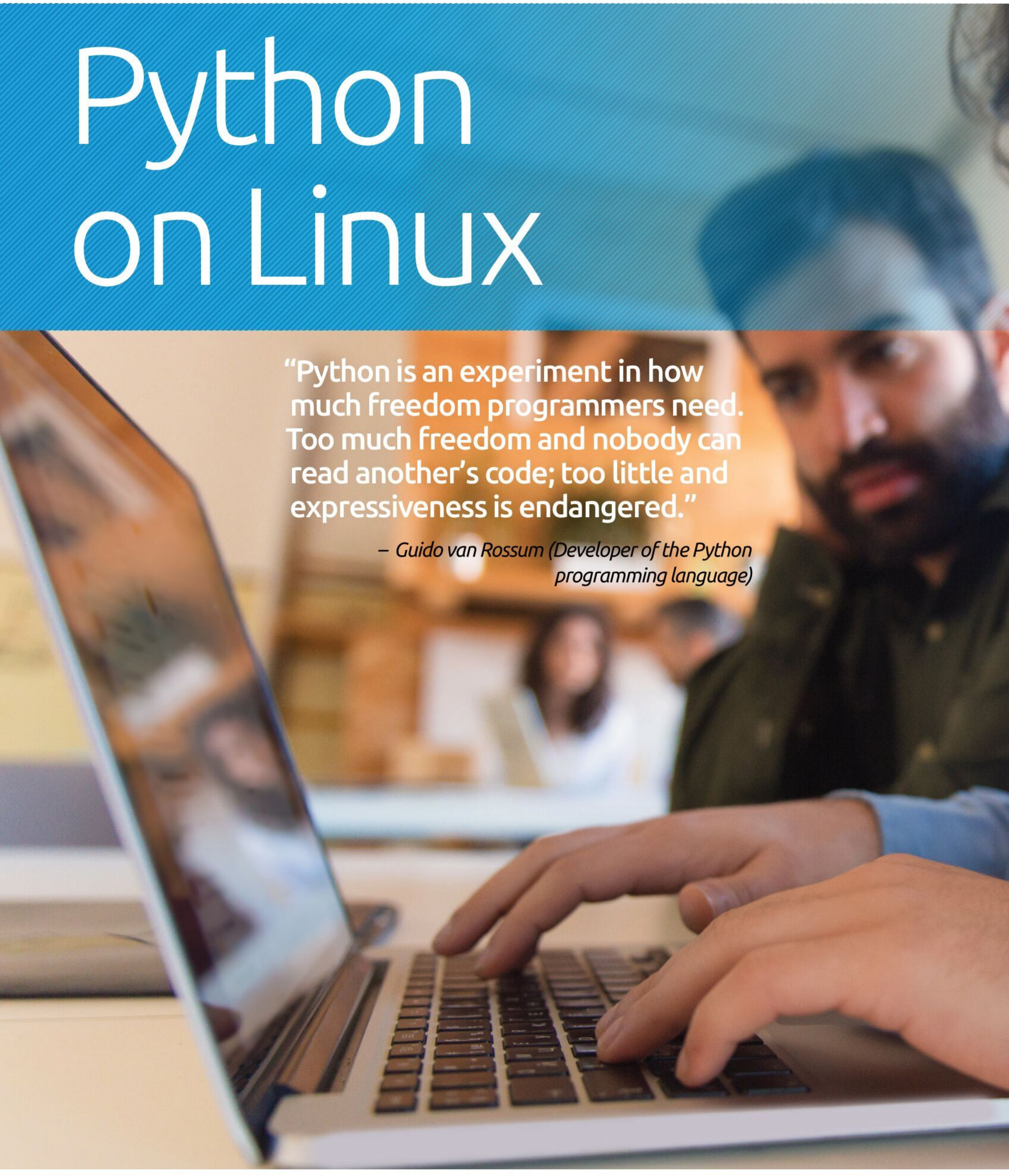
At the heart of it lies a complex setup of SUSE Linux Enterprise Server, along with many custom development packages and other software unique to NASA's projects. The projects NASA's Advanced Supercomputing Division uses Pleiades for are: The Kepler Mission – searching for extra-solar Earth-like planets; CFD (Computational Fluid Dynamics) modelling to help create more efficient space launch systems; Dark Matter research and simulation; and visualisation of Earth's ocean currents, building data for the ECCO (Estimating the Circulation and Climate of the Ocean) Project.

Did You Know…

NASA's Linux-powered supercomputer, Pleiades. Currently ranked the 11th most powerful in the world.

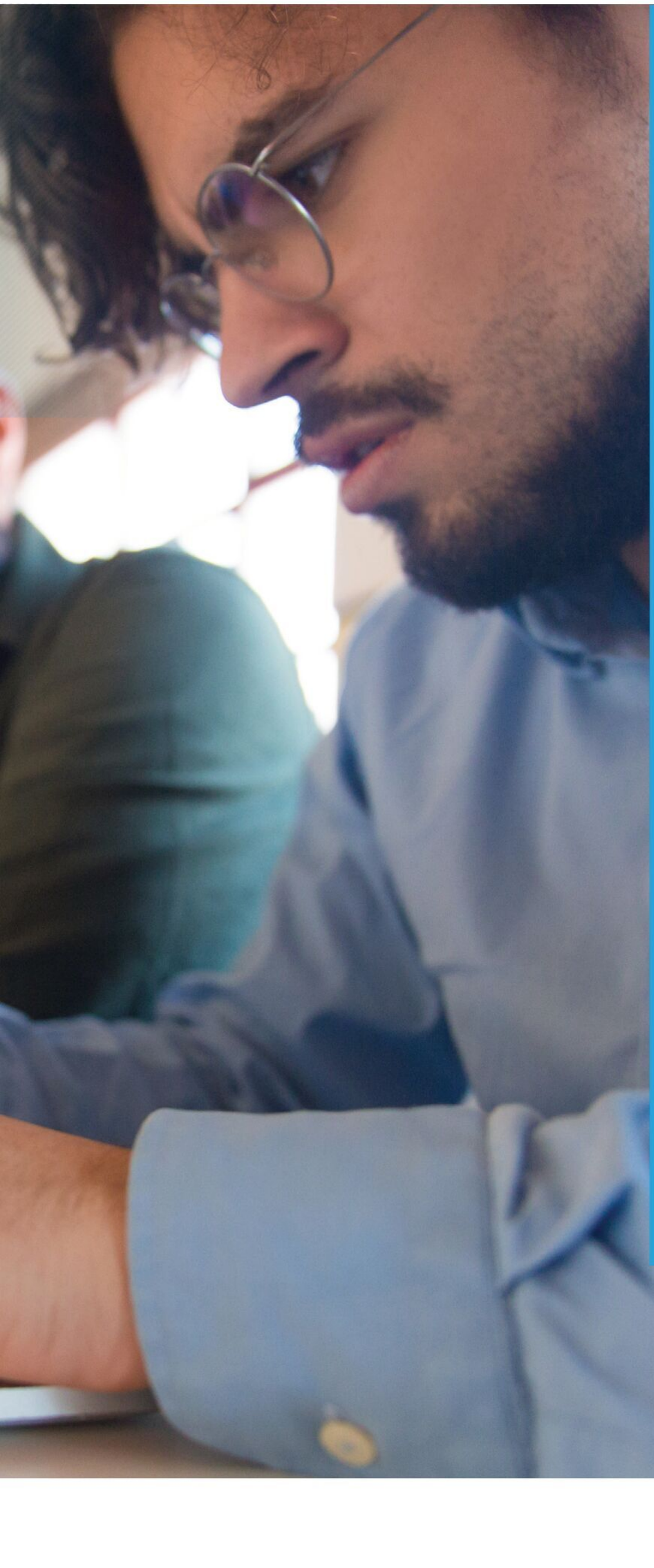
Good enough for **NASA**

Python on Linux



"Python is an experiment in how much freedom programmers need. Too much freedom and nobody can read another's code; too little and expressiveness is endangered."

– Guido van Rossum (Developer of the Python programming language)



Python is a fantastic programming language, combining ease of use with a generous helping of power to allow the user to create both minor utilities or performance-heavy computational tasks.

However, there's more to Python than simply being another programming language. It has a vibrant and lively community behind it that shares knowledge, code and project ideas, as well as bug fixes for future releases.

We've used a Raspberry Pi for this section of the book, as it's Linux based and one of the best coding platforms available. The Pi's features and functions make it the perfect Python programming partner, so let's get started.

88 Why Python?
90 How to Set Up Python in Linux
92 Starting Python for the First Time
94 Your First Code
96 Saving and Executing Your Code
98 Executing Code from the Terminal
100 Did You Know…Space Invaders
102 Numbers and Expressions
104 Using Comments
106 Working with Variables
108 User Input
110 Creating Functions
112 Conditions and Loops
114 Python Modules
116 Did You Know…Debugging

Why Python?

There are many different programming languages available for the modern computer, and some still available for older 8 and 16-bit computers too. Some of these languages are designed for scientific work, others for mobile platforms and such. So why choose Python out of all the rest?

PYTHON POWER

Ever since the earliest home computers were available, enthusiasts, users and professionals have toiled away until the wee hours, slaving over an overheating heap of circuitry to create something akin to magic.

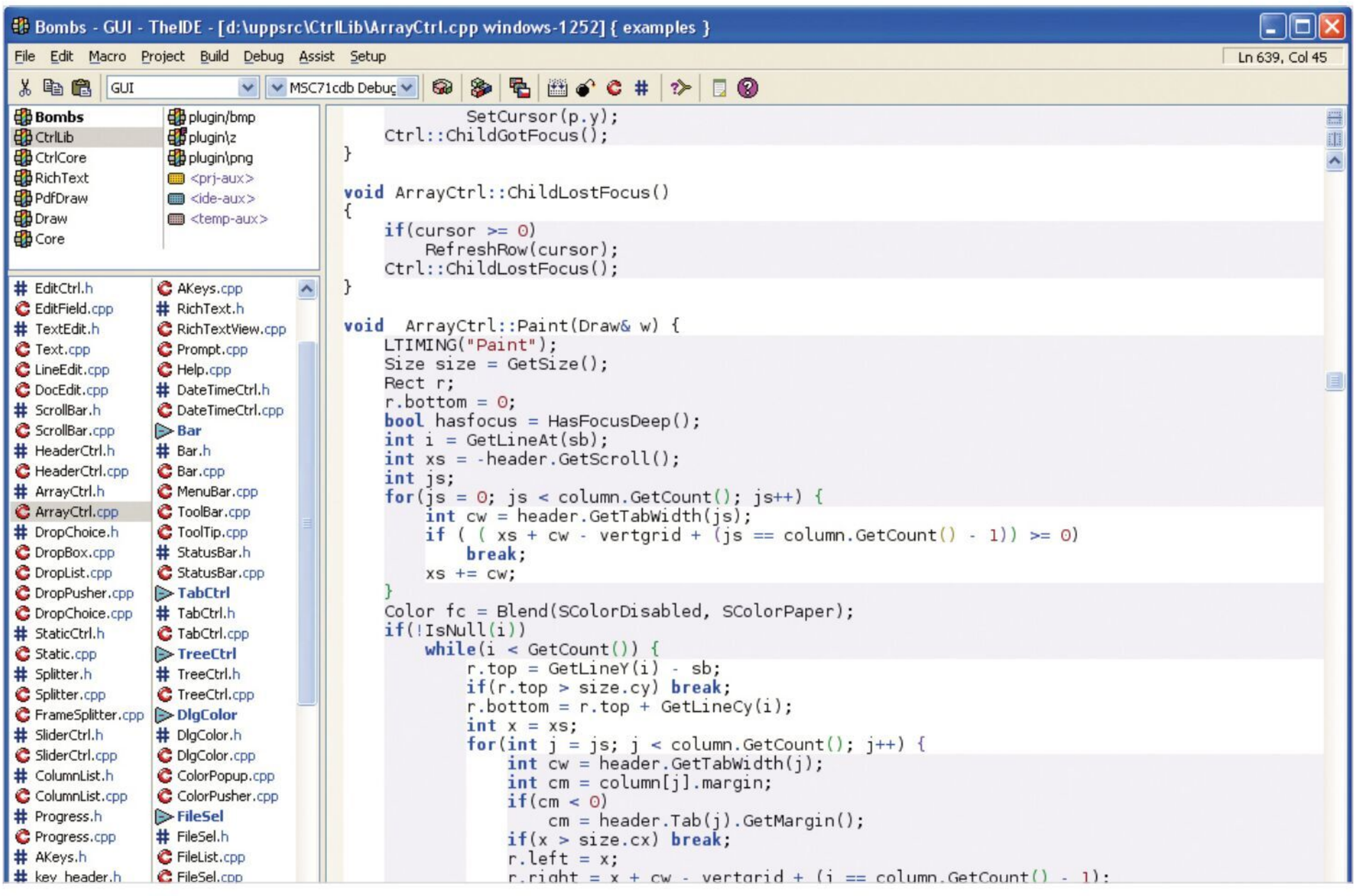
These pioneers of programming carved their way into a new frontier, forging small routines that enabled the letter 'A' to scroll across the screen. It may not sound terribly exciting to a generation that's used to ultra high-definition graphics and open world, multi-player online gaming. However, forty-something years ago it was blindingly brilliant.

Naturally these bedroom coders helped form the foundations for every piece of digital technology we use today. Some went on to become chief developers for top software companies, whereas others pushed the available hardware to its limits and founded the billion pound gaming empire that continually amazes us.

Regardless of whether you use an Android device, iOS device, PC, Mac, Linux, Smart TV, games console, MP3 player, GPS device built-in to a car, set-top box or a thousand other connected and 'smart' appliances, behind them all is programming.

All those aforementioned digital devices need instructions to tell them what to do, and allow them to be interacted with. These instructions form the programming core of the device and that core can be built using a variety of programming languages.

The languages in use today differ depending on the situation, the platform, the device's use and how the device will interact with its



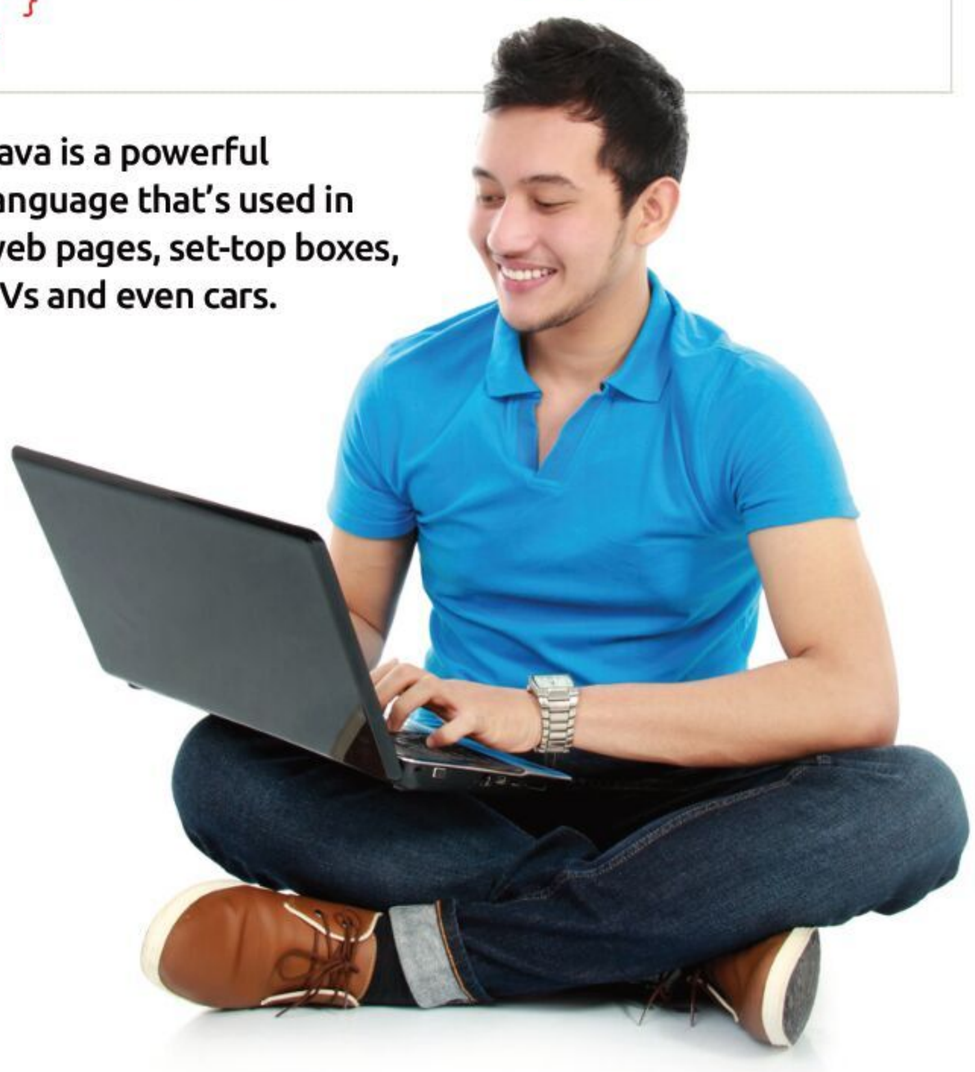
environment or users. Operating systems, such as Windows, macOS and such are usually a combination of C++, C#, assembly and some form of visual-based language. Games generally use C++ whilst web pages can use a plethora of available languages such as HTML, Java, Python and so on.

More general-purpose programming is used to create programs, apps, software or whatever else you want to call them. They're widely used across all hardware platforms and suit virtually every conceivable application. Some operate faster than others and some are easier to learn and use than others. Python is one such general-purpose language.

Python is what's known as a High-Level Language, in that it 'talks' to the hardware and operating system using a variety of arrays, variables, objects, arithmetic, subroutines, loops and countless more interactions. Whilst it's not as streamlined as a Low-Level Language, which can deal directly with memory addresses, call stacks and registers, its benefit is that it's universally accessible and easy to learn.

```
1  //file: Invoke.java
2  import java.lang.reflect.*;
3
4  class Invoke {
5    public static void main( String [] args ) {
6      try {
7        Class c = Class.forName( args[0] );
8        Method m = c.getMethod( args[1], new Class
         [] { } );
9        Object ret =  m.invoke( null, null );
10       System.out.println(
11           "Invoked static method: " + args[1]
12           + " of class: " + args[0]
13           + " with no args\nResults: " + ret );
14     } catch ( ClassNotFoundException e ) {
15       // Class.forName(  ) can't find the class
16     } catch ( NoSuchMethodException e2 ) {
17       // that method doesn't exist
18     } catch ( IllegalAccessException e3 ) {
19       // we don't have permission to invoke that
         method
20     } catch ( InvocationTargetException e4 ) {
21       // an exception ocurred while invoking that
         method
22       System.out.println(
23           "Method threw an: " + e4.
             getTargetException(  ) );
24     }
25   }
26 }
```

Java is a powerful language that's used in web pages, set-top boxes, TVs and even cars.



Python was created over twenty six years ago and has evolved to become an ideal beginner's language for learning how to program a computer. It's perfect for the hobbyist, enthusiast, student, teacher and those who simply need to create their own unique interaction between either themselves or a piece of external hardware and the computer itself.

Python is free to download, install and use and is available for Linux, Windows, macOS, MS-DOS, OS/2, BeOS, IBM i-series machines, and even RISC OS. It has been voted one of the top five programming languages in the world and is continually evolving ahead of the hardware and Internet development curve.

So to answer the question: why python? Simply put, it's free, easy to learn, exceptionally powerful, universally accepted, effective and a superb learning and educational tool.

```
  40 LET py=15
  70 FOR w=1 TO 10
  71 CLS
  75 LET by=INT (RND*28)
  80 LET bx=0
  90 FOR d=1 TO 20
 100 PRINT AT px,py;" U "
 110 PRINT AT bx,by;"o"
 120 IF INKEY$="p" THEN LET py=p
y+1
 130 IF INKEY$="o" THEN LET py=p
y-1                U
 135 FOR n=1 TO 100: NEXT n
 140 IF py<2 THEN LET py=2
 150 IF py>27 THEN LET py=27
 180 LET bx=bx+1
 185 PRINT AT bx-1,by;" "
 190 NEXT d
 200 IF (by-1)=py THEN LET s=s+1
 210 PRINT AT 10,10;"score=";s
 220 FOR v=1 TO 1000: NEXT v
 300 NEXT w

0 OK, 0:1
```

BASIC was once the starter language that early 8-bit home computer users learned.

```
print(HANGMAN[0])
attempts = len(HANGMAN) - 1


while (attempts != 0 and "-" in word_guessed):
    print(("\nYou have {} attempts remaining").format(attempts))
    joined_word = "".join(word_guessed)
    print(joined_word)

    try:
        player_guess = str(input("\nPlease select a letter between A-Z" + "\n> ")).
    except: # check valid input
        print("That is not valid input. Please try again.")
        continue
    else:
        if not player_guess.isalpha(): # check the input is a letter. Also checks a
            print("That is not a letter. Please try again.")
            continue
        elif len(player_guess) > 1: # check the input is only one letter
            print("That is more than one letter. Please try again.")
            continue
        elif player_guess in guessed_letters: # check it letter hasn't been guessed
            print("You have already guessed that letter. Please try again.")
            continue
        else:
            pass

    guessed_letters.append(player_guess)

    for letter in range(len(chosen_word)):
        if player_guess == chosen_word[letter]:
            word_guessed[letter] = player_guess # replace all letters in the chosen

    if player_guess not in chosen_word:
```

Python is a more modern take on BASIC, it's easy to learn and makes for an ideal beginner's programming language.

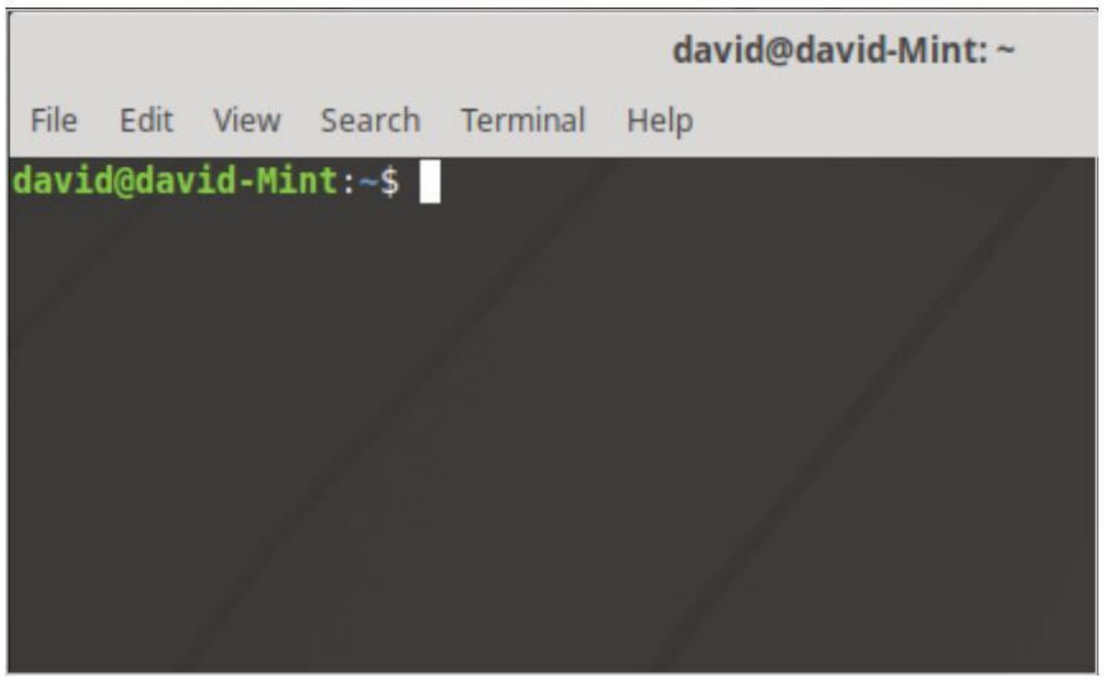
How to Set Up Python in Linux

While the Raspberry Pi's operating system contains the latest, stable version of Python, other Linux distros don't come with Python 3 preinstalled. If you're not going down the Pi route, then here's how to check and install Python for Linux.

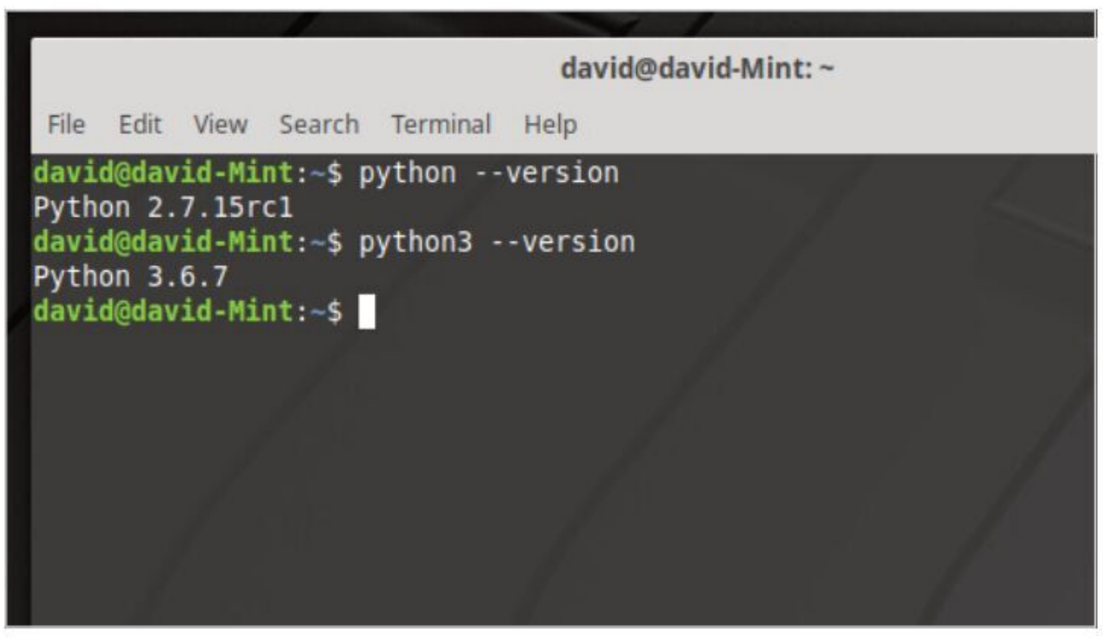
PYTHON PENGUIN

Linux is such a versatile operating system that it's often difficult to nail down just one way of doing something. Different distributions go about installing software in different ways, so we're sticking with Linux Mint for this particular tutorial.

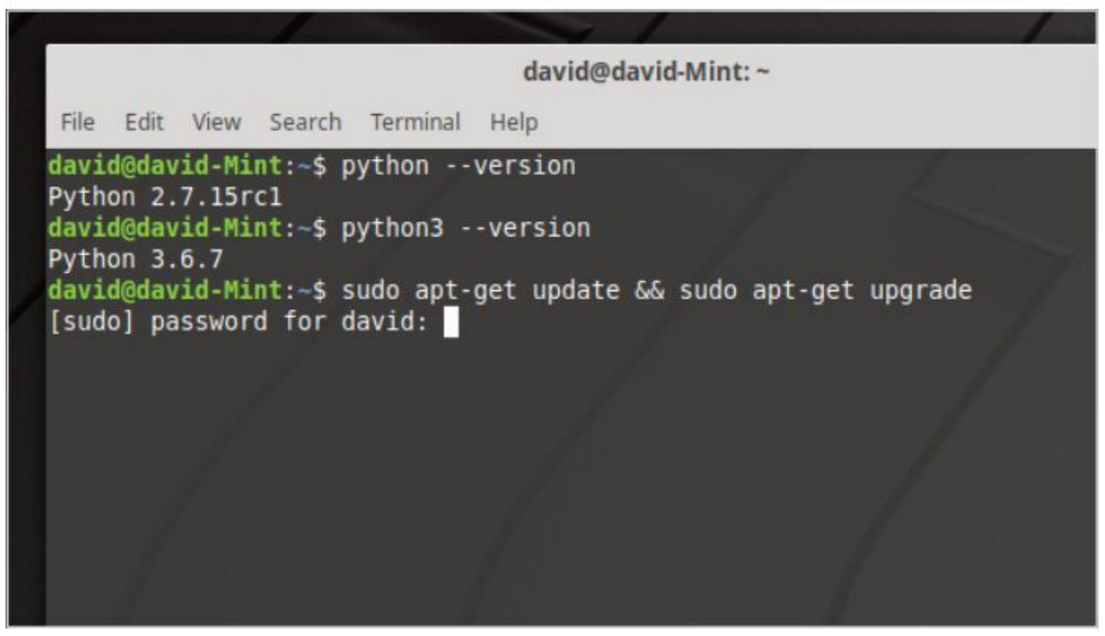
STEP 1 First you need to ascertain which version of Python is currently installed in your Linux system. To begin with, drop into a Terminal session from your distro's menu or hit the Ctrl+Alt+T keys.



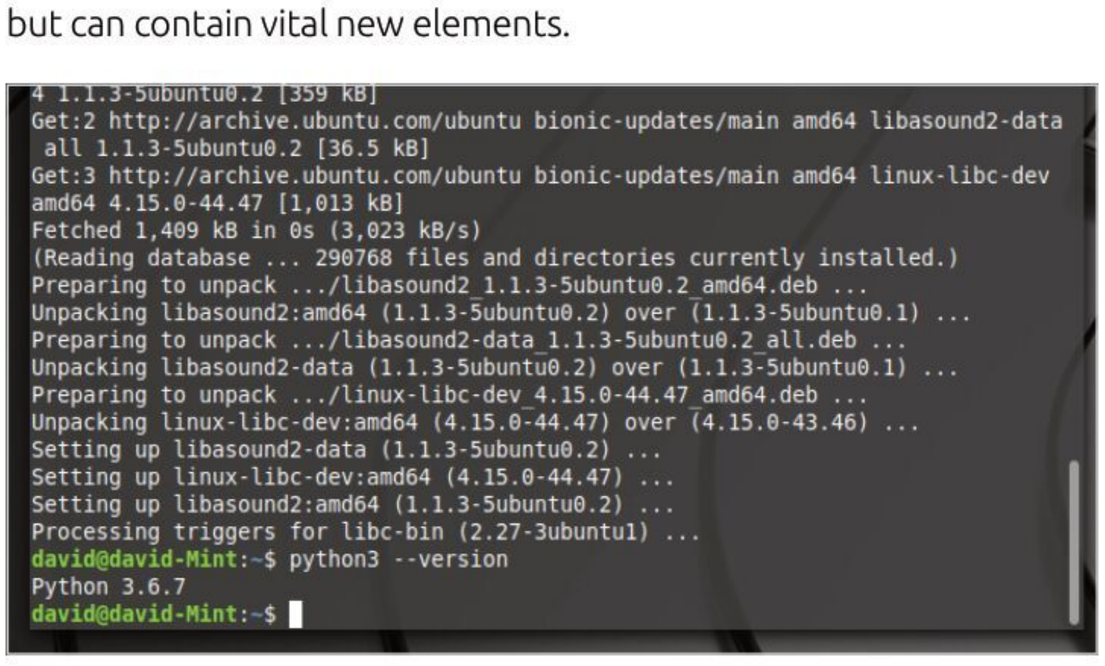
STEP 2 Next enter `python --version` into the Terminal screen. You should have the output relating to version 2.x of Python in the display. By default, most Linux distros come with both Python 2 and 3, as there's plenty of code out there still available for Python 2. Now enter: `python3 --version`.



STEP 3 In our case we have both Python 2 and 3 installed; as long as Python 3.x.x is installed, then the code in our tutorials will work. It's always worth checking to see if the distro has been updated with the latest versions, so enter: `sudo apt-get update && sudo apt-get upgrade` to update the system.

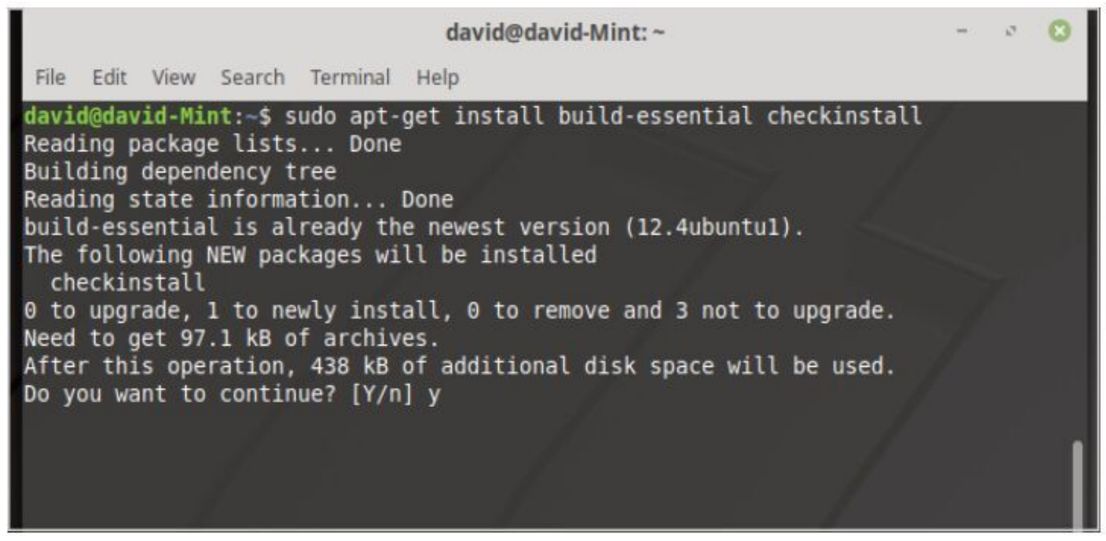


STEP 4 Once the update and upgrade completes, enter: `python3 --version` again to see if Python 3.x is updated or even installed; as long as you have Python 3.x, you're running the most recent major version. The numbers after the 3. indicate patches and further updates. Often they're unnecessary but can contain vital new elements.

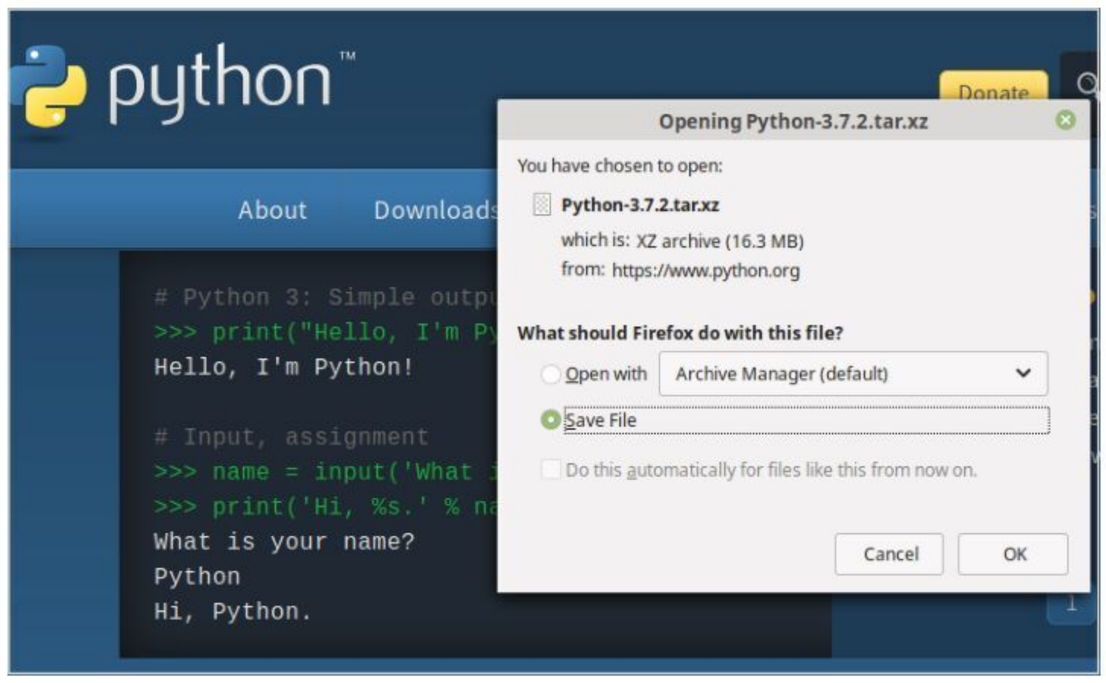


STEP 5 However, if you want the latest, cutting edge version, you need to build Python from source. Start by entering these commands into the Terminal:

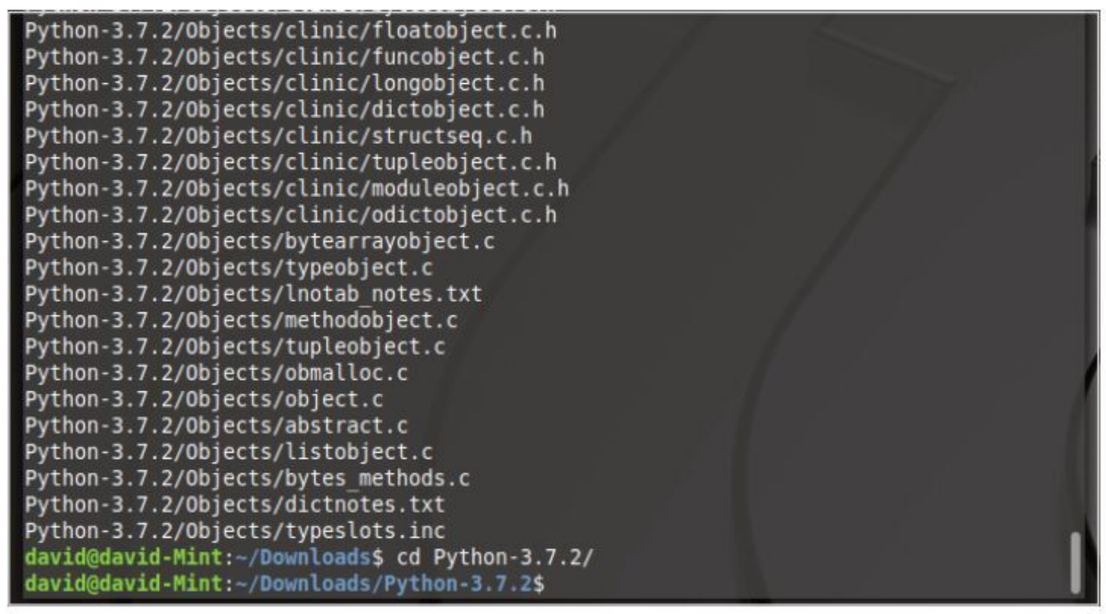
```
sudo apt-get install build-essential checkinstall
sudo apt-get install libreadline-gplv2-dev
libncursesw5-dev libssl-dev libsqlite3-dev tk-dev
libgdbm-dev libc6-dev libbz2-dev
```



STEP 6 Open up your Linux web browser and go to the Python download page: **www.python.org/downloads**. Click on Downloads followed by the button under the Python Source window. This opens a download dialogue box; choose a location and start the download process.



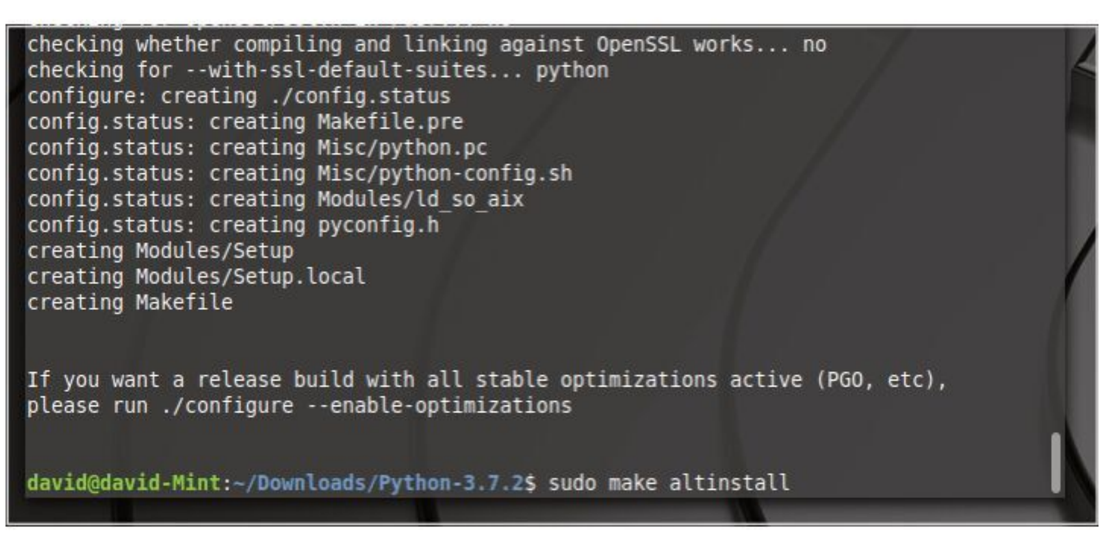
STEP 7 In the Terminal, go the Downloads folder by entering `cd Downloads/`. Then unzip the contents of the downloaded Python source code with: `tar -xvf Python-3.Y.Y.tar.xz` replacing the Y's with the version numbers you've downloaded. Now enter the newly unzipped folder with `cd Python-3.Y.Y/`.



STEP 8 Within the Python folder, enter:

```
./configure
sudo make altinstall
```

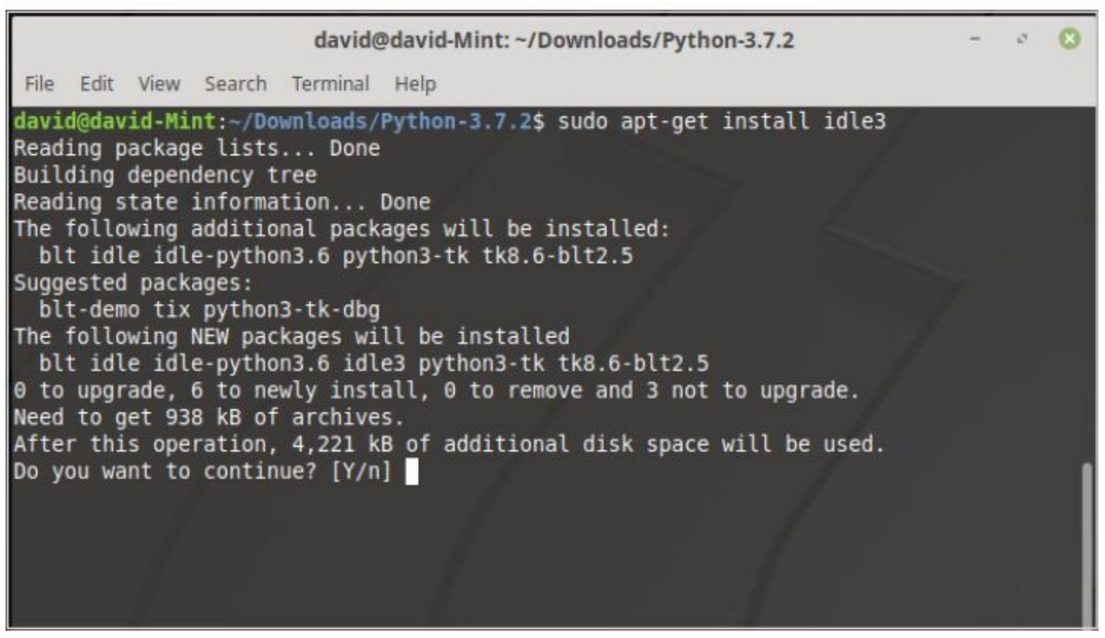
This could take a while depending on the speed of your computer. Once finished, enter: `python3.7 --version` to check the installed latest version. You now have Python 3.7 installed, alongside older Python 3.x.x and Python 2.



STEP 9 For the GUI IDLE, you need to enter the following command into the Terminal:

```
sudo apt-get install idle3
```

The IDLE can then be started with the command idle3. Note that IDLE runs a different version from the one you installed from source.

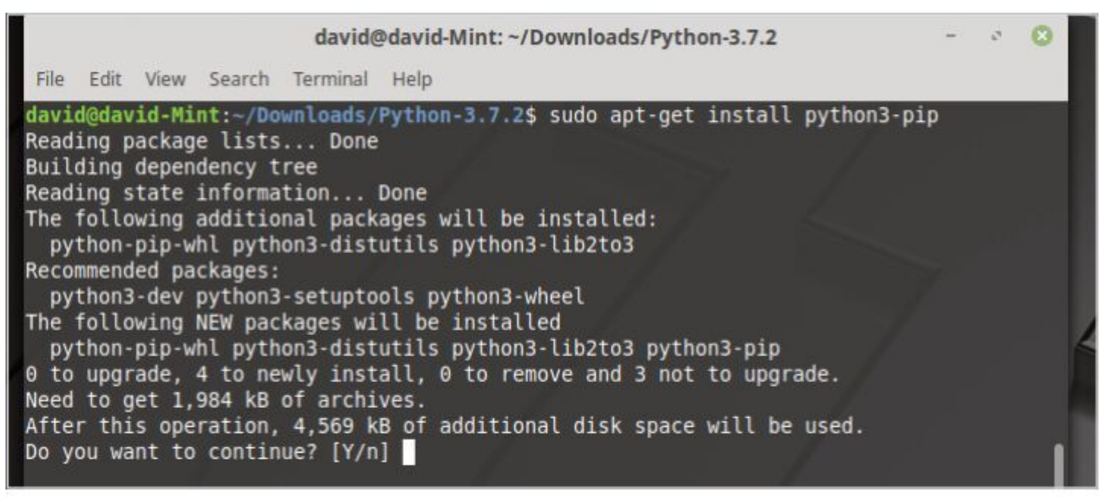


STEP 10 You also need PIP (Pip Installs Packages) which is a tool to help you install more modules and extras. Enter: `sudo apt-get install python3-pip`

When PIP is installed, check for the latest update with:

```
pip3 install --upgrade pip
```

When complete, close the Terminal and Python 3.x will be available via the Programming section in your distro's menu.



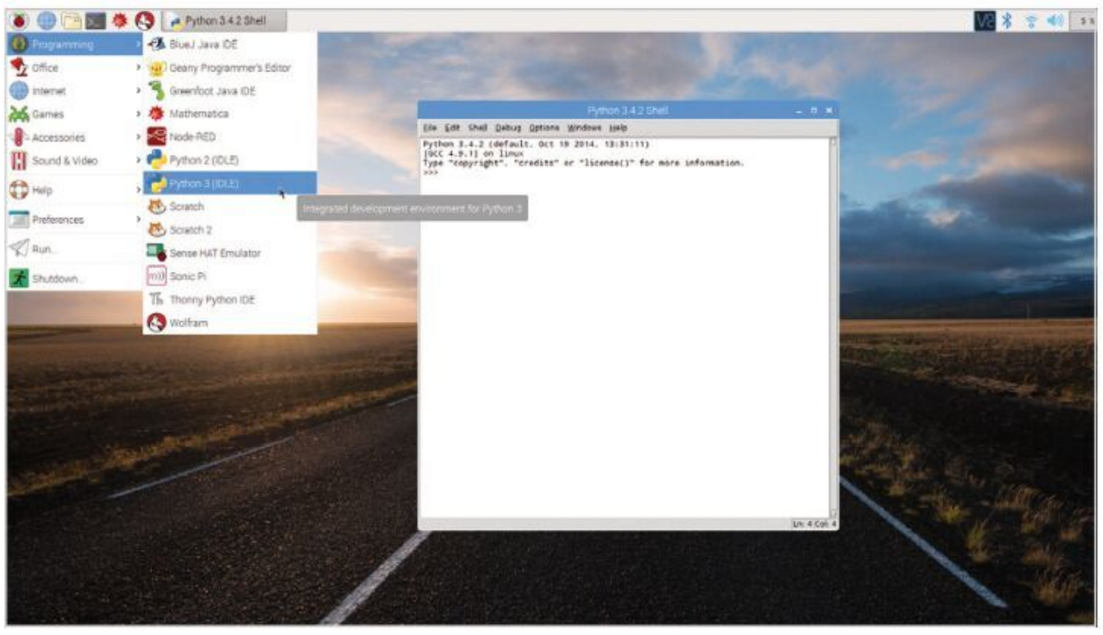
Starting Python for the First Time

We're going to be using the Raspberry Pi as our Python 3 hardware platform. The latest version of Raspbian comes preinstalled with Python 3, version 3.4.2 to be exact, so as long as you have a version 3 Shell, all our code will work.

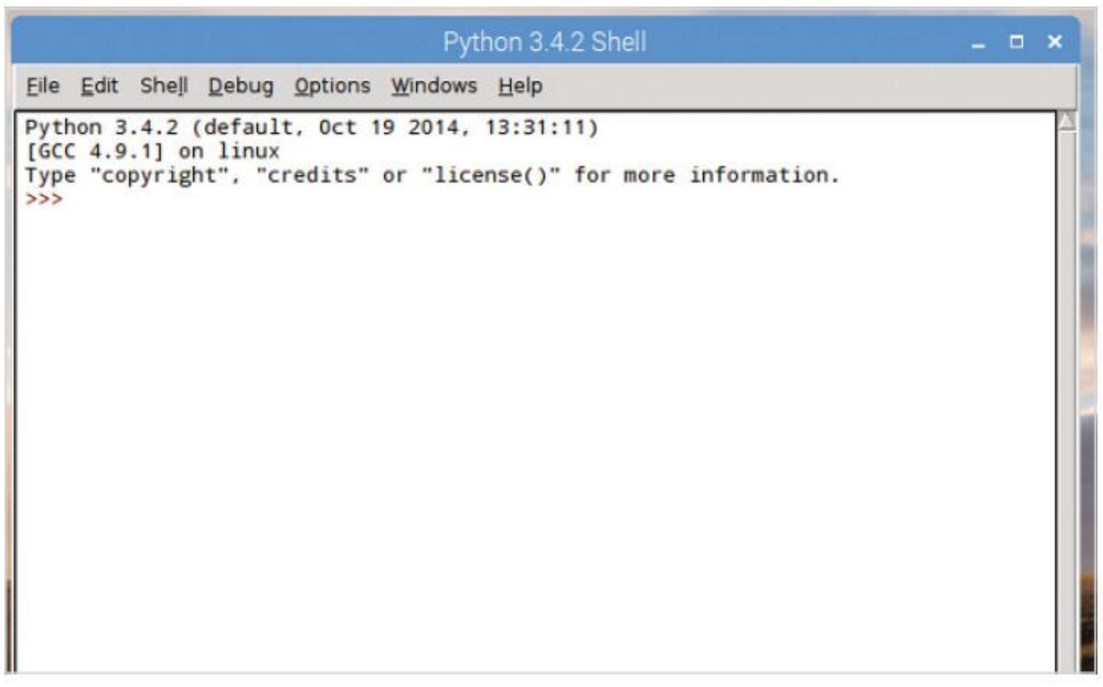
STARTING PYTHON

We're not going to go into the details of getting the Raspberry Pi up and running, there's plenty of material already available on that subject. However, once you're ready, fire up your Pi and get ready for coding.

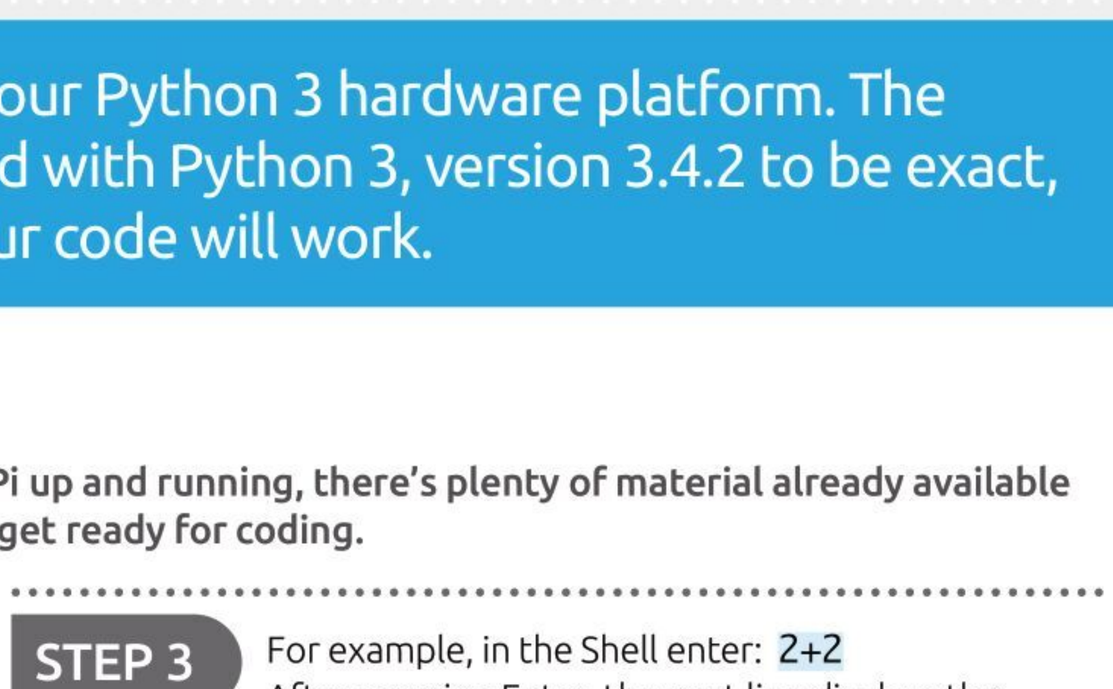
STEP 1 With the Raspbian desktop loaded, click on the Menu button followed by Programming > Python 3 (IDLE). This opens the Python 3 Shell. Windows and Mac users can find the Python 3 IDLE Shell from within the Windows Start button menu and via Finder.



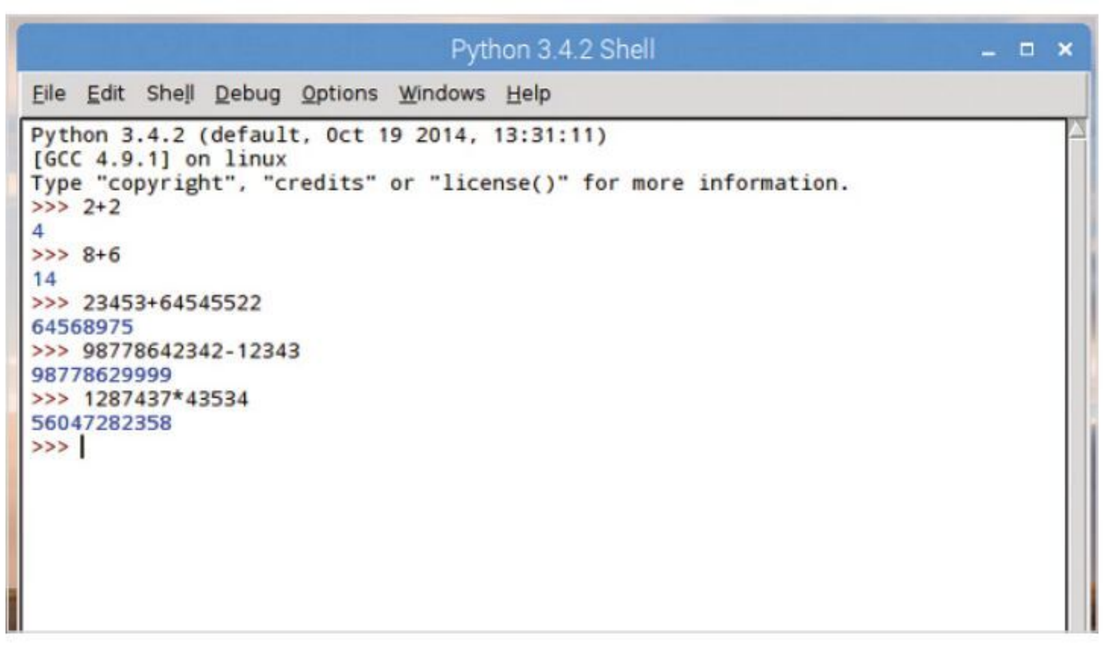
STEP 2 The Shell is where you can enter code and see the responses and output of code you've programmed into Python. This is a kind of sandbox, where you're able to try out some simple code and processes.



STEP 3 For example, in the Shell enter: `2+2` After pressing Enter, the next line displays the answer: 4. Basically, Python has taken the 'code' and produced the relevant output.



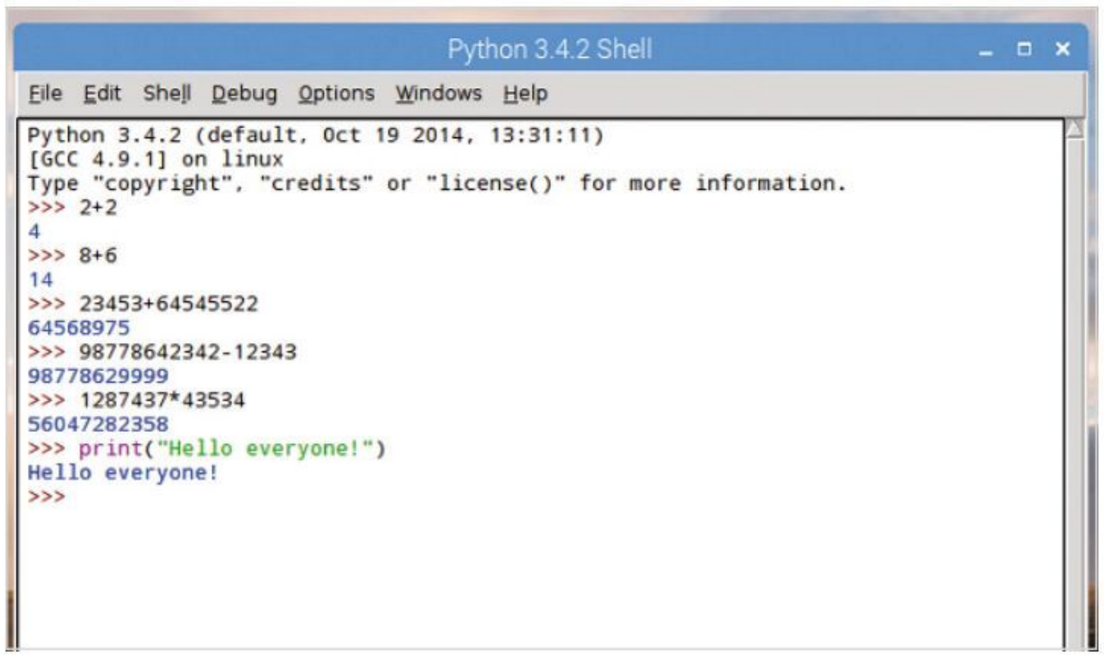
STEP 4 The Python Shell acts very much like a calculator, since code is basically a series of mathematical interactions with the system. Integers, which are the infinite sequence of whole numbers can easily be added, subtracted, multiplied and so on.



STEP 5 While that's very interesting, it's not particularly exciting. Instead, try this:

```
print("Hello everyone!")
```

Just enter it into the IDLE as you've done in the previous steps.



STEP 6 This is a little more like it, since you've just produced your first bit of code. The Print command is fairly self-explanatory, it prints things. Python 3 requires the brackets as well as quote marks in order to output content to the screen, in this case the 'Hello everyone!' bit.

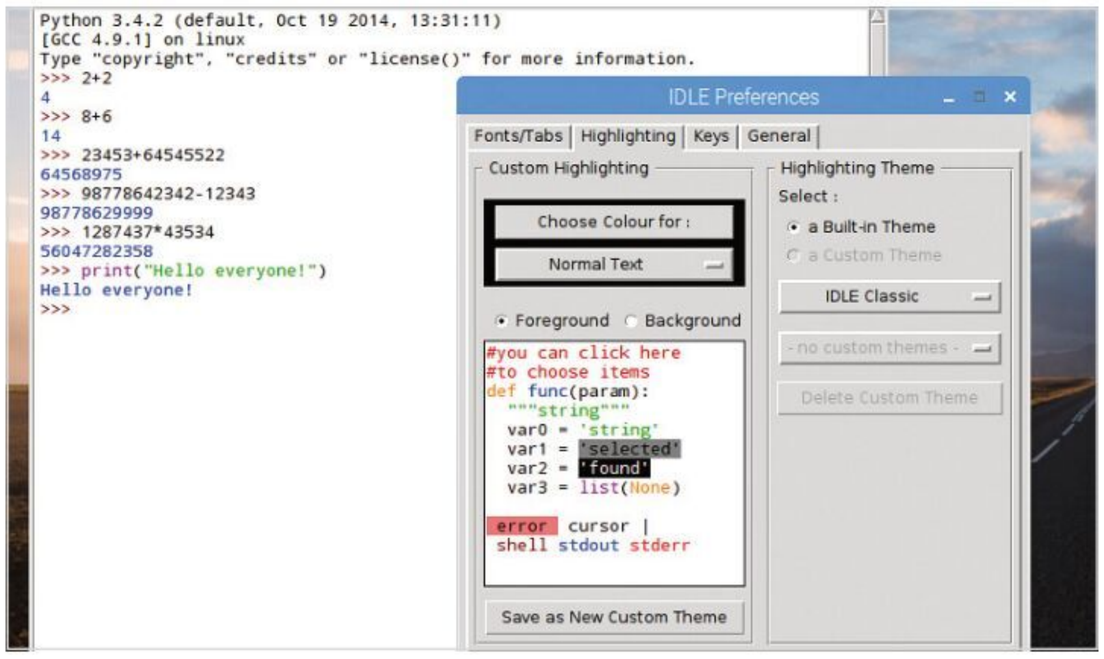
STEP 7 You may have noticed the colour coding within the Python IDLE. The colours represent different elements of Python code. They are:

Black – Data and Variables
Green – Strings
Purple – Functions
Orange – Commands
Blue – User Functions
Dark Red – Comments
Light Red – Error Messages

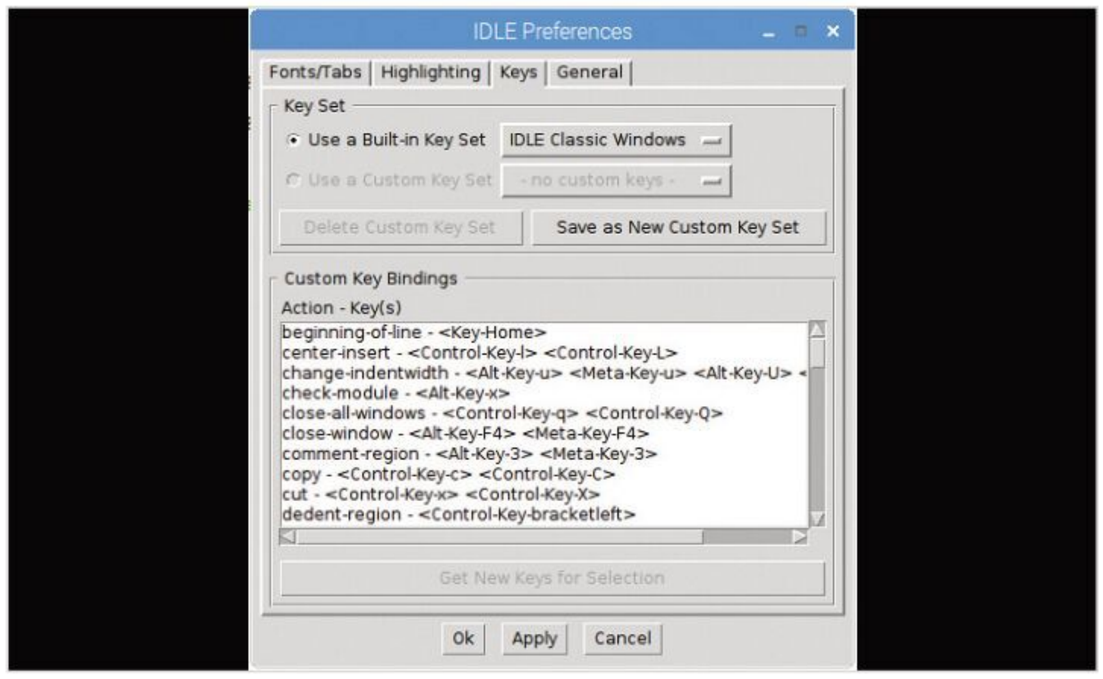
IDLE Colour Coding

Colour	Use for	Examples
Black	Data & variables	`23.6 area`
Green	Strings	`"Hello World"`
Purple	Functions	`len() print()`
Orange	Commands	`if for else`
Blue	User functions	`get_area()`
Dark red	Comments	`#Remember VAT`
Light red	Error messages	`SyntaxError:`

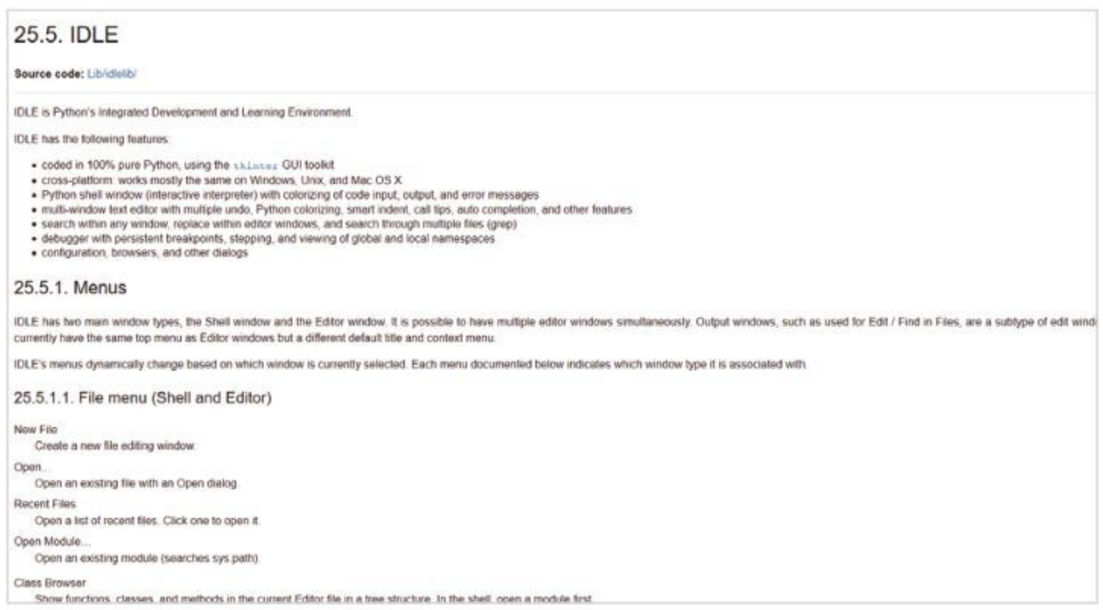
STEP 8 The Python IDLE is a configurable environment. If you don't like the way the colours are represented, then you can always change them via Options > Configure IDLE and clicking on the Highlighting tab. However, we don't recommend that, as you won't be seeing the same as our screenshots.



STEP 9 Just like most programs available, regardless of the operating system, there are numerous shortcut keys available. We don't have room for them all here but within the Options > Configure IDLE and under the Keys tab, you can see a list of the current bindings.



STEP 10 The Python IDLE is a power interface and one that's actually been written in Python using one of the available GUI toolkits. If you want to know the many ins and outs of the Shell, we recommend you take a few moments to view **www.docs.python.org/3/library/idle.html**, which details many of the IDLE's features.



25.5. IDLE

Source code: Lib/idlelib/

IDLE is Python's Integrated Development and Learning Environment.

IDLE has the following features:

- coded in 100% pure Python, using the tkinter GUI toolkit
- cross-platform: works mostly the same on Windows, Unix, and Mac OS X
- Python shell window (interactive interpreter) with colorizing of code input, output, and error messages
- multi-window text editor with multiple undo, Python colorizing, smart indent, call tips, auto completion, and other features
- search within any window, replace within editor windows, and search through multiple files (grep)
- debugger with persistent breakpoints, stepping, and viewing of global and local namespaces
- configuration, browsers, and other dialogs

25.5.1. Menus

IDLE has two main window types, the Shell window and the Editor window. It is possible to have multiple editor windows simultaneously. Output windows, such as used for Edit / Find in Files, are a subtype of edit windo currently have the same top menu as Editor windows but a different default title and context menu.

IDLE's menus dynamically change based on which window is currently selected. Each menu documented below indicates which window type it is associated with.

25.5.1.1. File menu (Shell and Editor)

New File
Create a new file editing window.

Open...
Open an existing file with an Open dialog.

Recent Files
Open a list of recent files. Click one to open it.

Open Module...
Open an existing module (searches sys.path).

Class Browser
Show functions, classes, and methods in the current Editor file in a tree structure. In the shell, open a module first.

STEP 10 The Python IDLE is a power interface and one that's actually been written in Python using one of the available GUI toolkits. If you want to know the many ins and outs of the Shell, we recommend you take a few moments to view **www.docs.python.org/3/library/idle.html**, which details many of the IDLE's features.

Your First Code

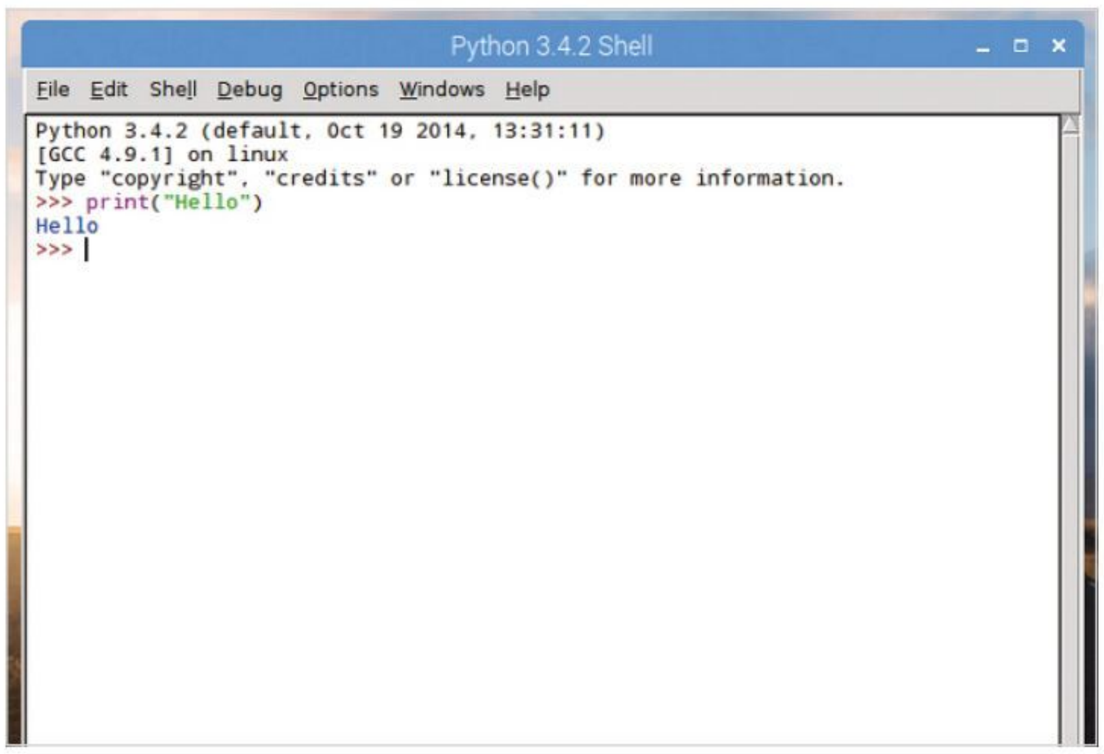
Essentially, you've already written your first piece of code with the 'print("Hello everyone!")' function from the previous tutorial. However, let's expand that and look at entering your code and playing around with some other Python examples.

PLAYING WITH PYTHON

With most languages, computer or human, it's all about remembering and applying the right words to the right situation. You're not born knowing these words, so you need to learn them.

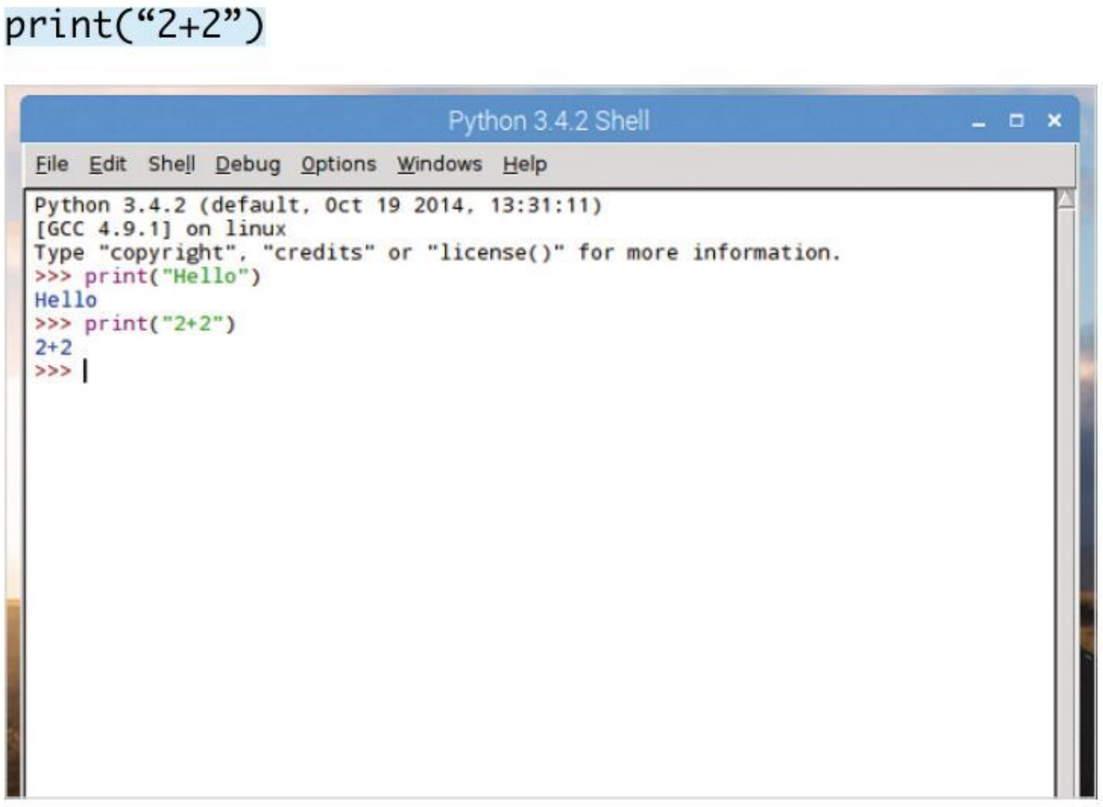
STEP 1 If you've closed Python 3 IDLE, reopen it in whichever operating system version you prefer. In the Shell, enter the familiar following:

```
print("Hello")
```



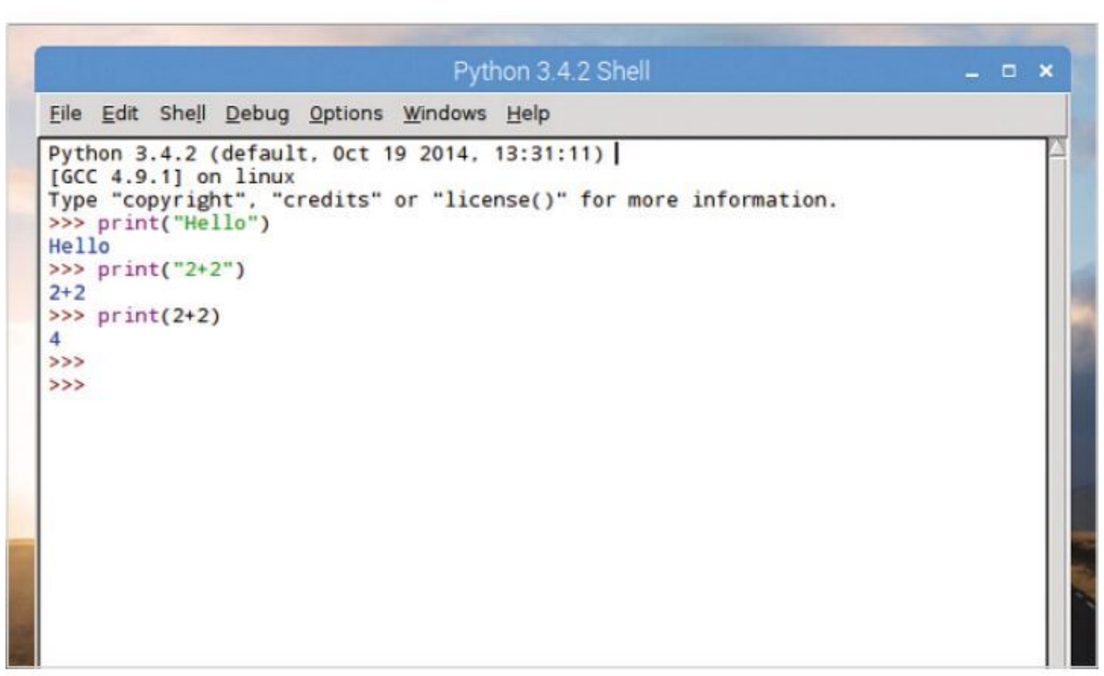
STEP 2 Just as predicted, the word Hello appears in the Shell as blue text, indicating output from a string. It's fairly straightforward and doesn't require too much explanation. Now try:

```
print("2+2")
```



STEP 3 You can see that instead of the number 4, the output is the 2+2 you asked to be printed to the screen. The quotation marks are defining what's being outputted to the IDLE Shell; to print the total of 2+2 you need to remove the quotes:

```
print(2+2)
```

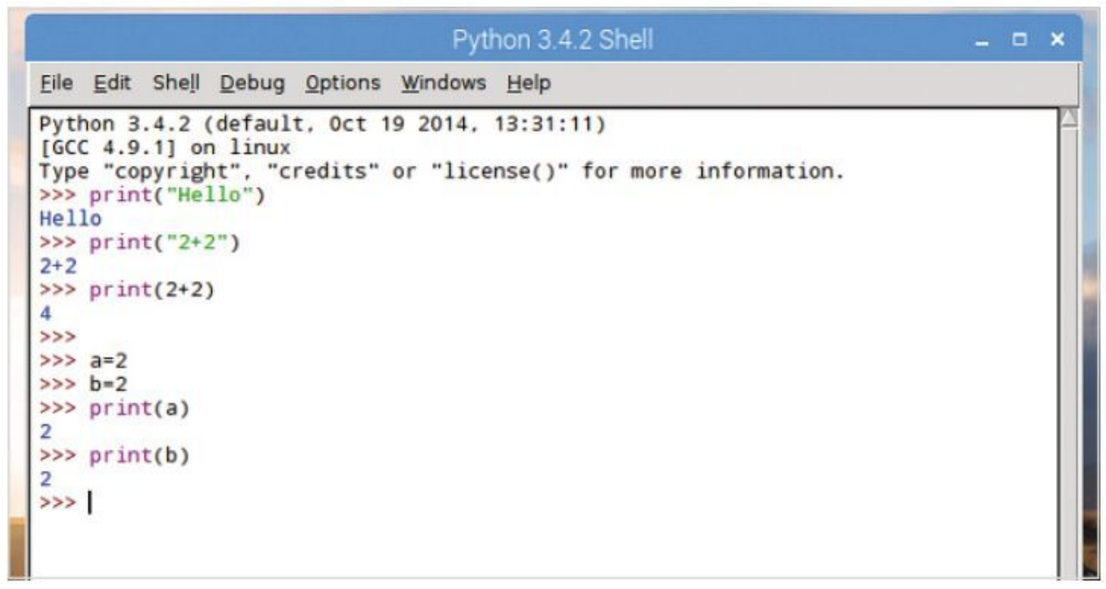


STEP 4 You can continue as such, printing 2+2, 464+2343 and so on to the Shell. An easier way is to use a variable, which is something we will cover in more depth later. For now, enter:

```
a=2
b=2
```

STEP 5 What you have done here is assign the letters a and b two values: 2 and 2. These are now variables, which can be called upon by Python to output, add, subtract, divide and so on for as long as their numbers stay the same. Try this:

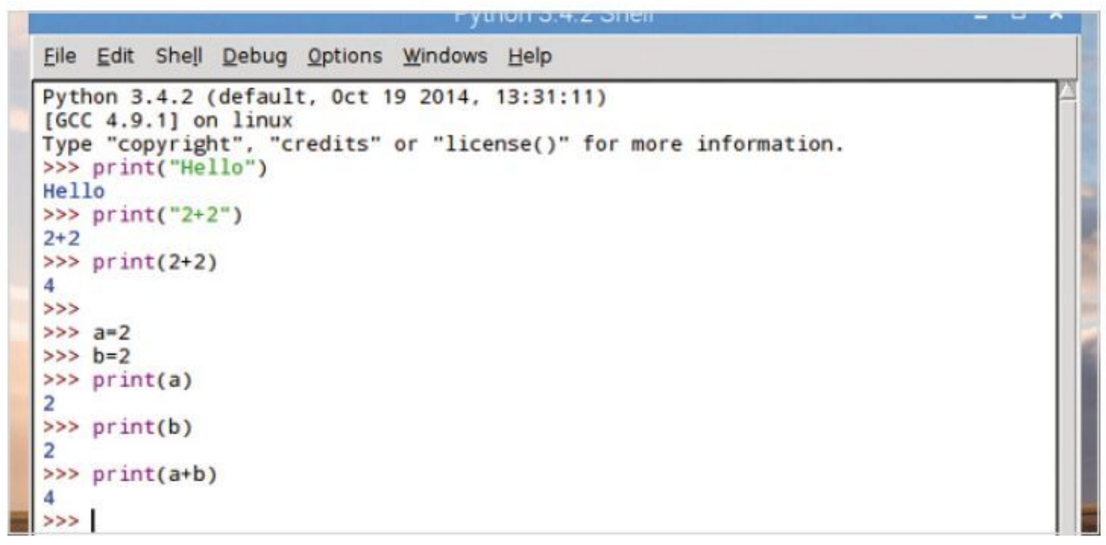
```
print(a)
print(b)
```



STEP 6 The output of the last step displays the current values of both a and b individually, as you've asked them to be printed separately. If you want to add them up, you can use the following:

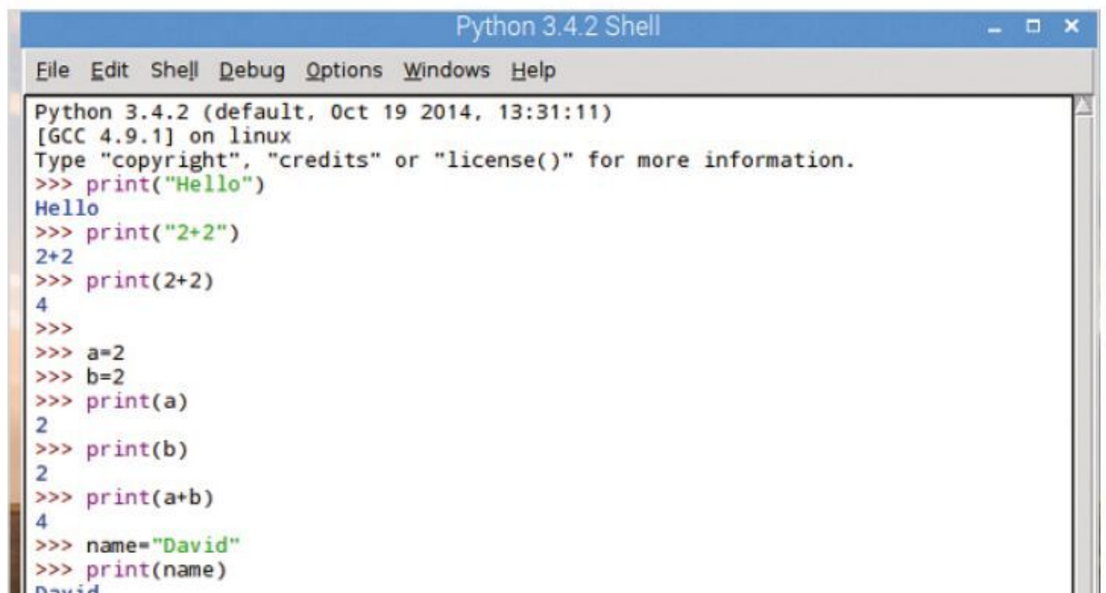
```
print(a+b)
```

This code simply takes the values of a and b, adds them together and outputs the result.



STEP 7 You can play around with different kinds of variables and the Print function. For example, you could assign variables for someone's name:

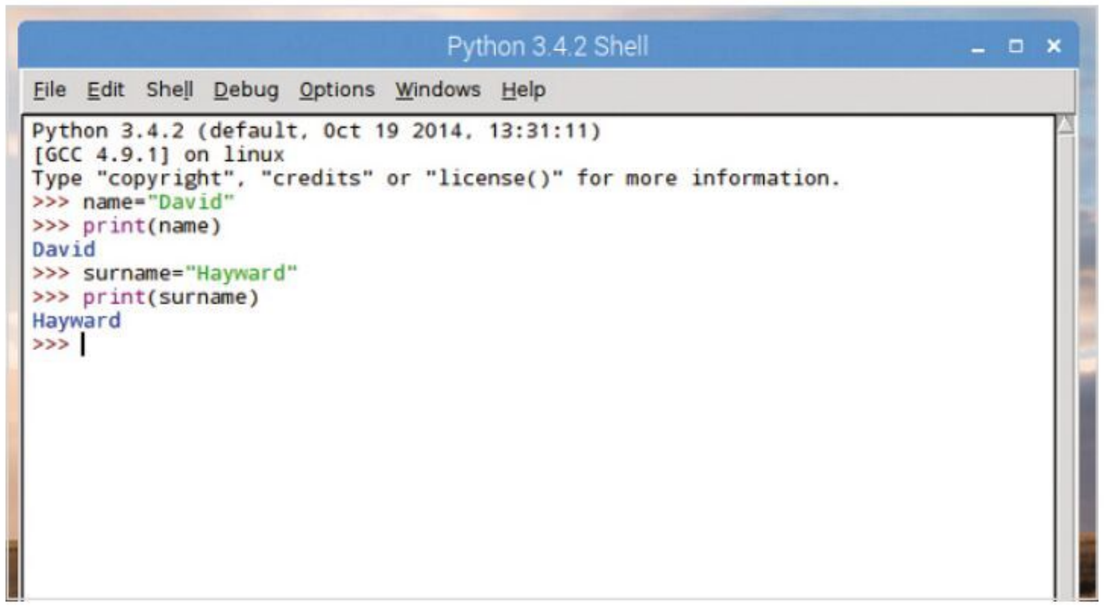
```
name=”David”
print(name)
```



STEP 8 Now let's add a surname:

```
surname=”Hayward”
print(surname)
```

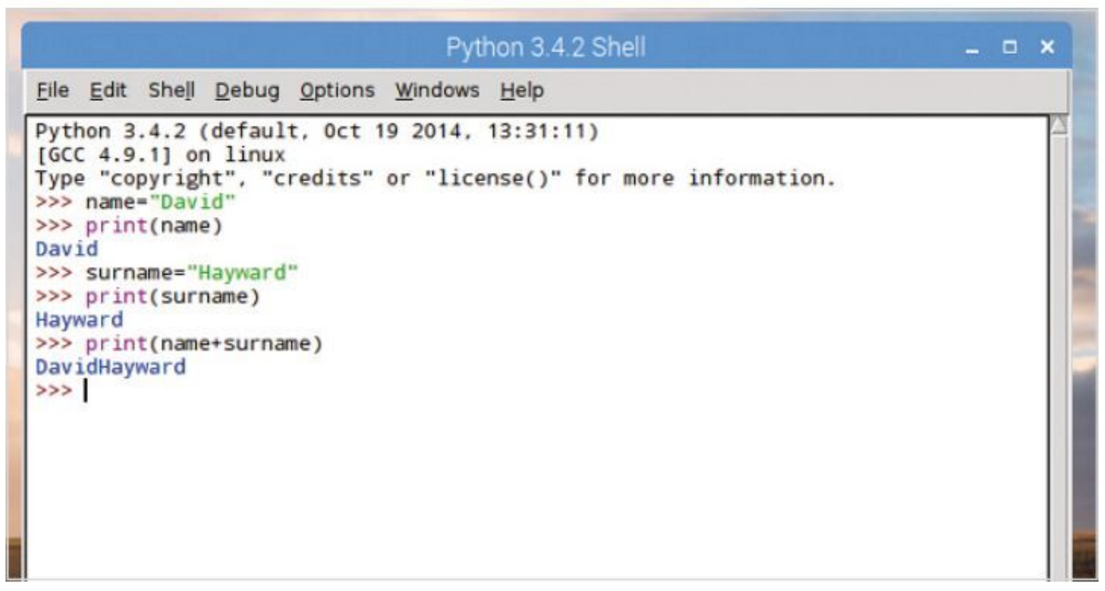
You now have two variables containing a first name and a surname and you can print them independently.



STEP 9 If we were to apply the same routine as before, using the + symbol, the name wouldn't appear correctly in the output in the Shell. Try it:

```
print(name+surname)
```

You need a space between the two, defining them as two separate values and not something you mathematically play around with.



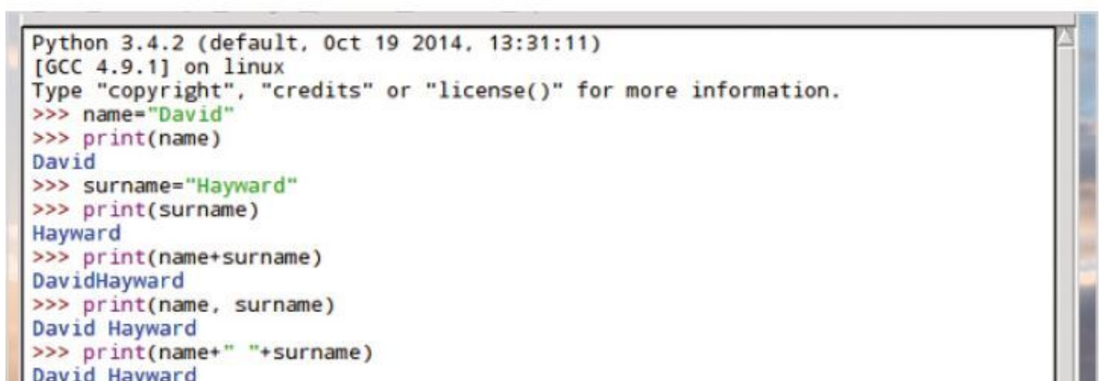
STEP 10 In Python 3 you can separate the two variables with a space using a comma:

```
print(name, surname)
```

Alternatively, you can add the space yourself:

```
print(name+” “+surname)
```

The use of the comma is much neater, as you can see. Congratulations, you've just taken your first steps into the wide world of Python.



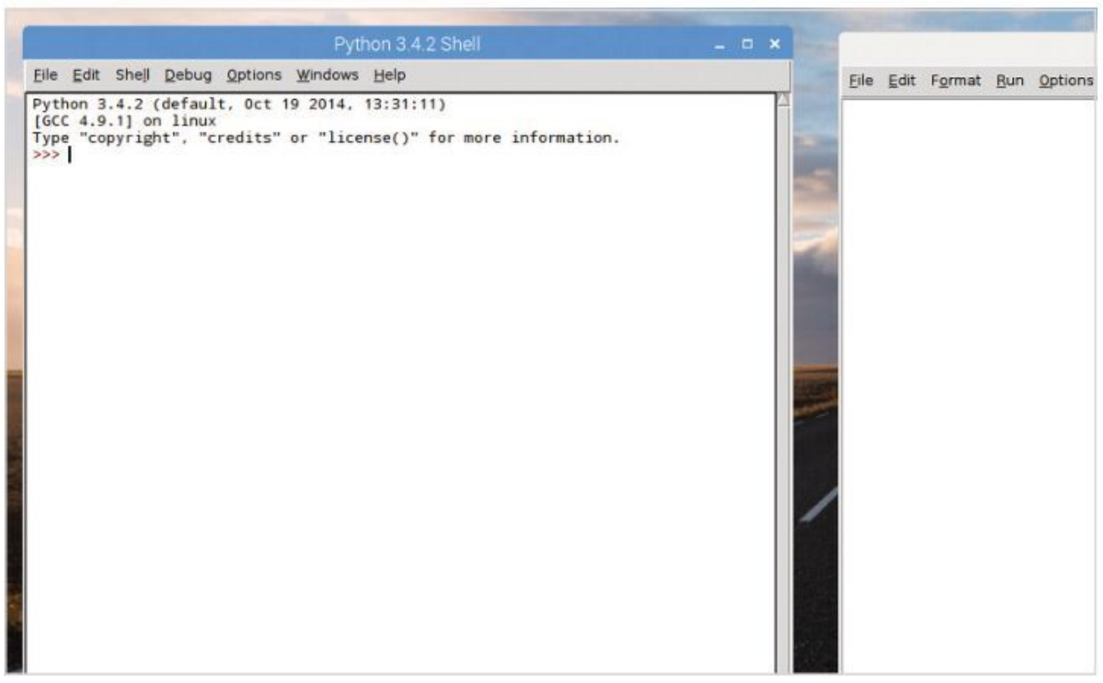
Saving and Executing Your Code

While working in the IDLE Shell is perfectly fine for small code snippets, it's not designed for entering longer program listings. In this section you're going to be introduced to the IDLE Editor, where you will be working from now on.

EDITING CODE

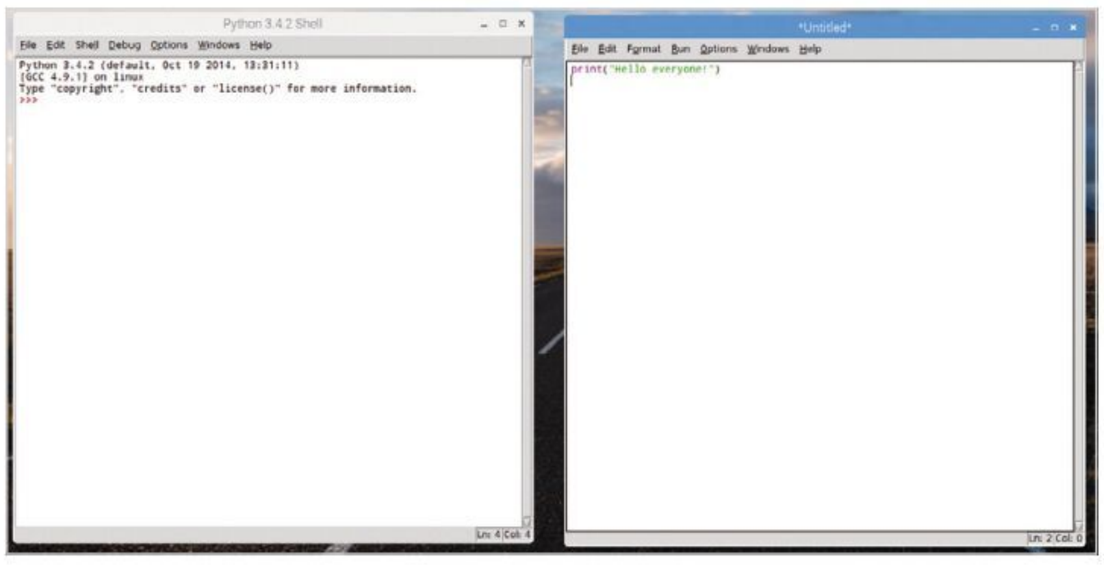
You will eventually reach a point where you have to move on from inputting single lines of code into the Shell. Instead, the IDLE Editor will allow you to save and execute your Python code.

STEP 1 First, open the Python IDLE Shell and when it's up, click on File > New File. This will open a new window with Untitled as its name. This is the Python IDLE Editor and within it you can enter the code needed to create your future programs.

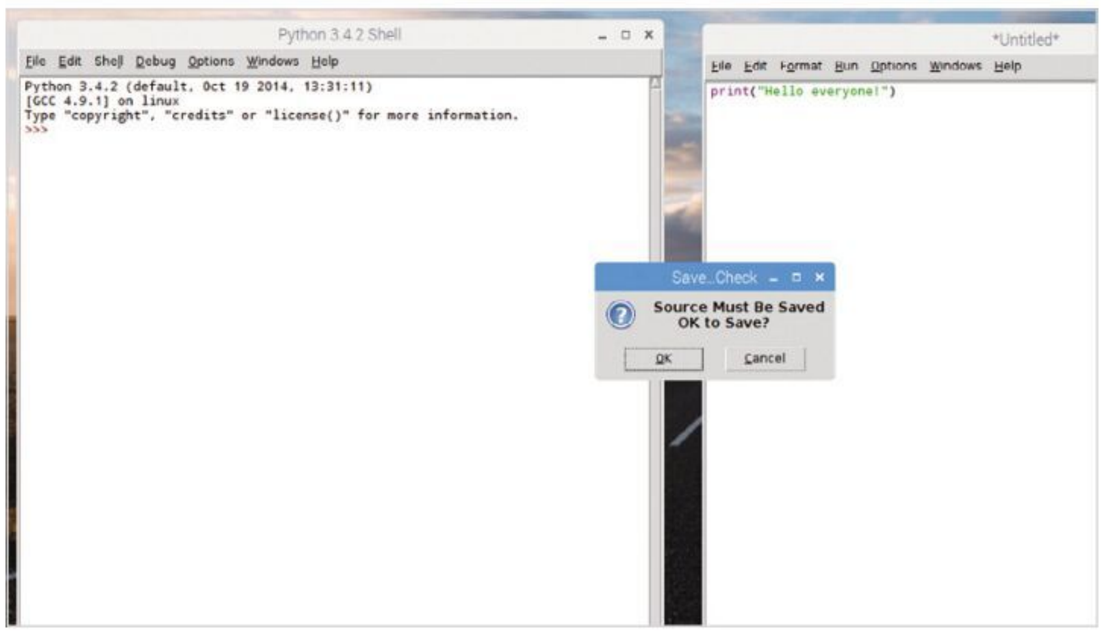


STEP 2 The IDLE Editor is, for all intents and purposes, a simple text editor with Python features, colour coding and so on; much in the same vein as Sublime. You enter code as you would within the Shell, so taking an example from the previous tutorial, enter:

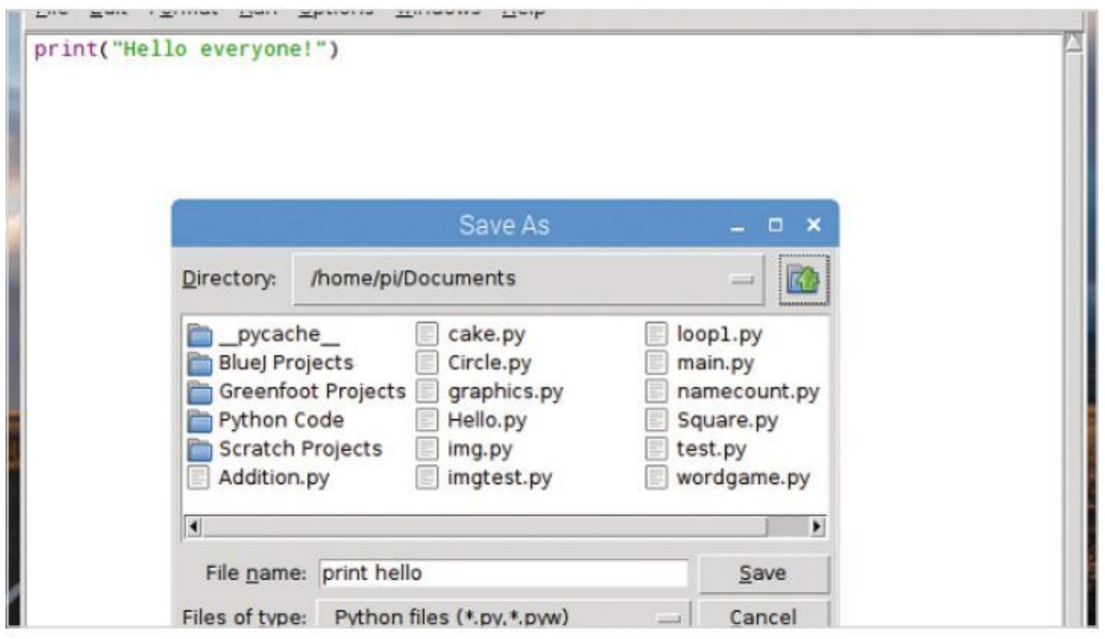
```
print("Hello everyone!")
```



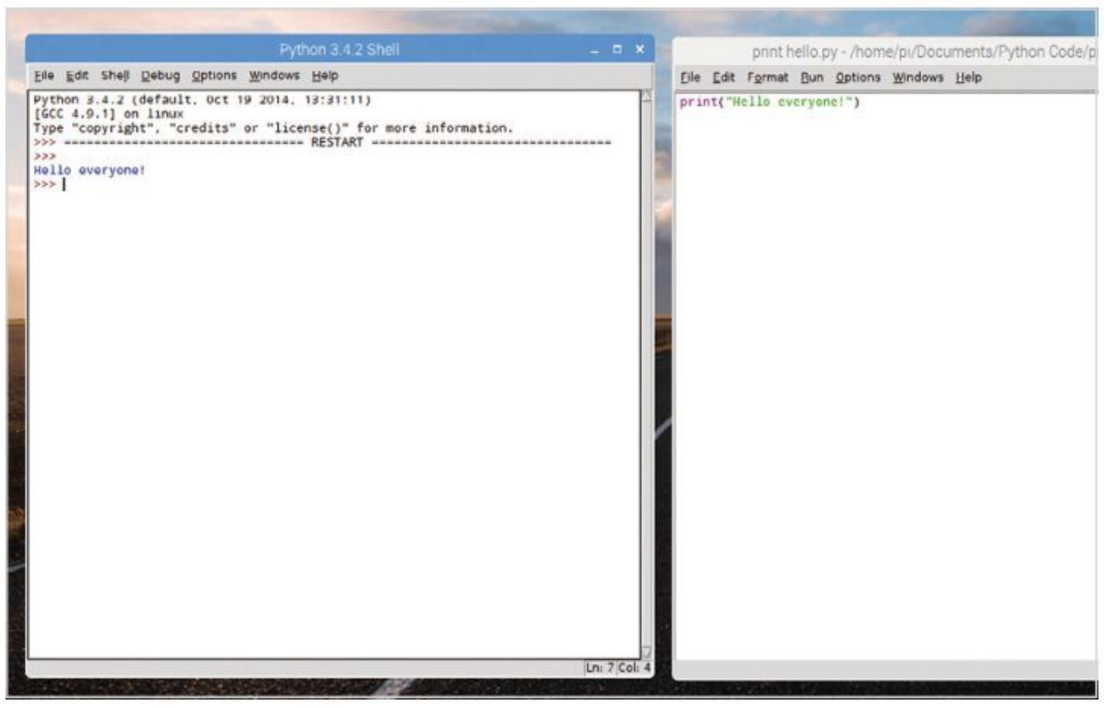
STEP 3 You can see that the same colour coding is in place in the IDLE Editor as it is in the Shell, enabling you to better understand what's going on with your code. However, to execute the code you need to first save it. Press F5 and you get a Save…Check box open.



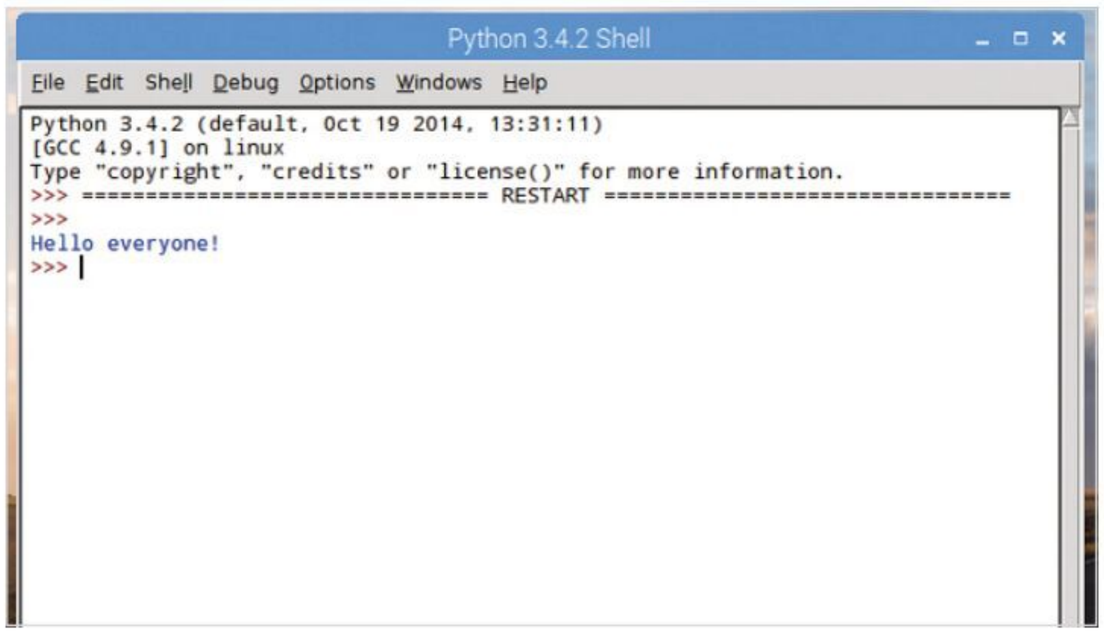
STEP 4 Click on the OK button in the Save box and select a destination where you'll save all your Python code. The destination can be a dedicated folder called Python or you can just dump it wherever you like. Remember to keep a tidy drive though, to help you out in the future.



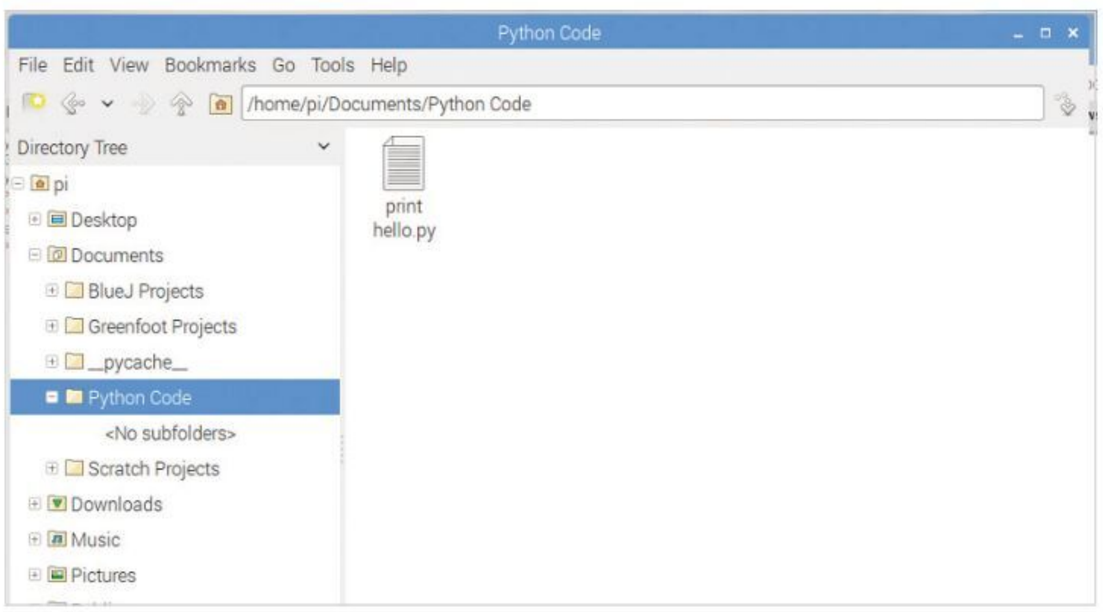
STEP 5 Enter a name for your code, 'print hello' for example, and click on the Save button. Once the Python code is saved it's executed and the output will be detailed in the IDLE Shell. In this case, the words 'Hello everyone!'.



STEP 6 This is how the vast majority of your Python code will be conducted. Enter it into the Editor, hit F5, save the code and look at the output in the Shell. Sometimes things will differ, depending on whether you've requested a separate window, but essentially that's the process. It's the process we will use throughout this book, unless otherwise stated.



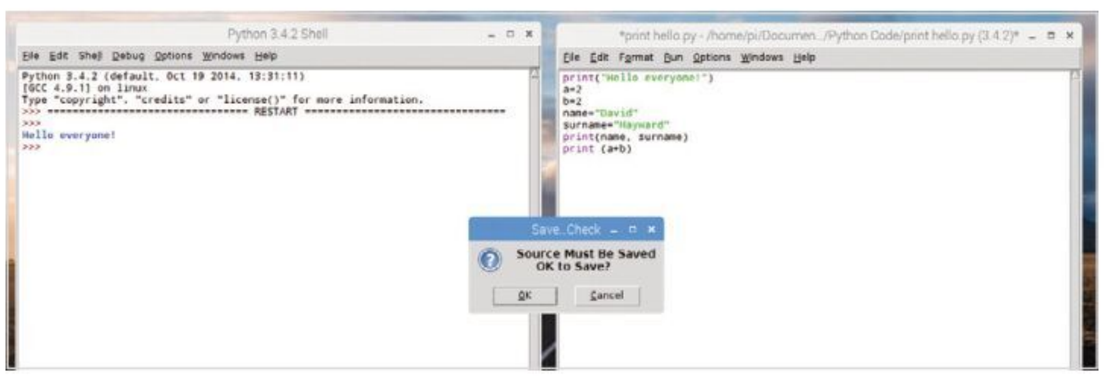
STEP 7 If you open the file location of the saved Python code, you can see that it ends in a .py extension. This is the default Python file name. Any code you create will be whatever.py and any code downloaded from the many Internet Python resource sites will be .py. Just ensure that the code is written for Python 3.



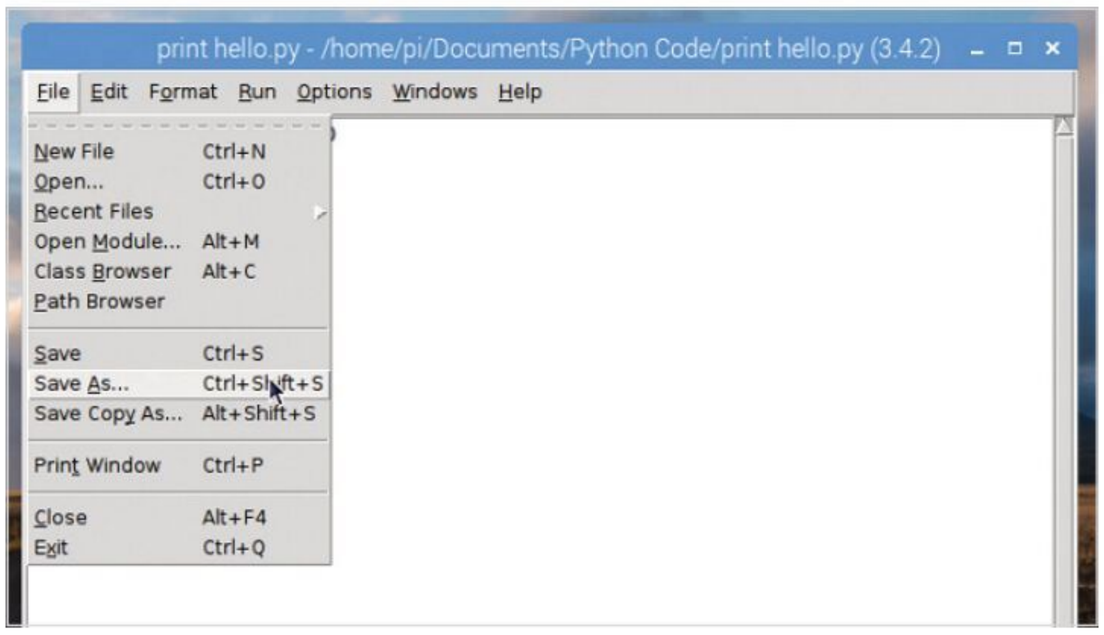
STEP 8 Let's extend the code and enter a few examples from the previous tutorial:

```
a=2
b=2
name="David"
surname="Hayward"
print(name, surname)
print (a+b)
```

If you press F5 now you'll be asked to save the file, again, as it's been modified from before.



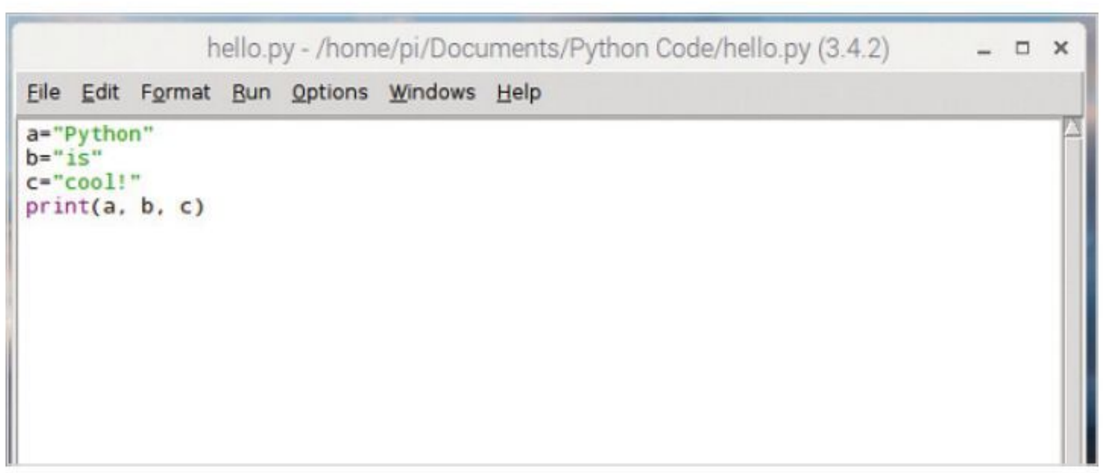
STEP 9 If you click the OK button, the file will be overwritten with the new code entries, and executed, with the output in the Shell. It's not a problem with just these few lines but if you were to edit a larger file, overwriting can become an issue. Instead, use File > Save As from within the Editor to create a backup.



STEP 10 Now create a new file. Close the Editor, and open a new instance (File > New File from the Shell). Enter the following and save it as hello.py:

```
a="Python"
b="is"
c="cool!"
print(a, b, c)
```

You will use this code in the next tutorial.



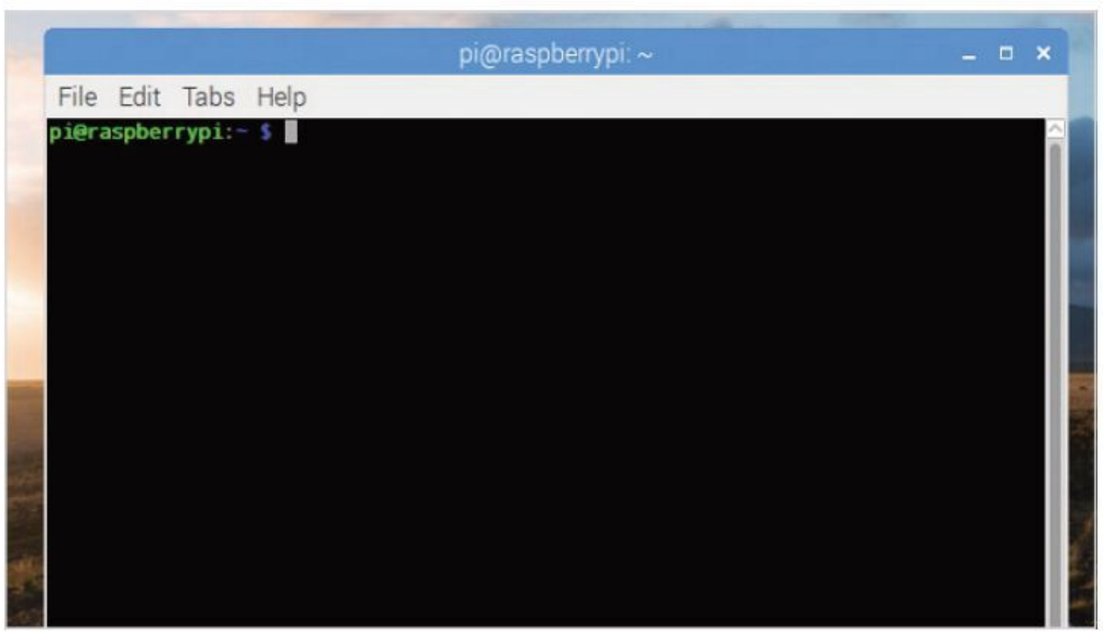
Executing Code from the Terminal

Although we're working from the GUI IDLE throughout this book, it's worth taking a look at Python's command line handling. We already know there's a command line version of Python but it's also used to execute code.

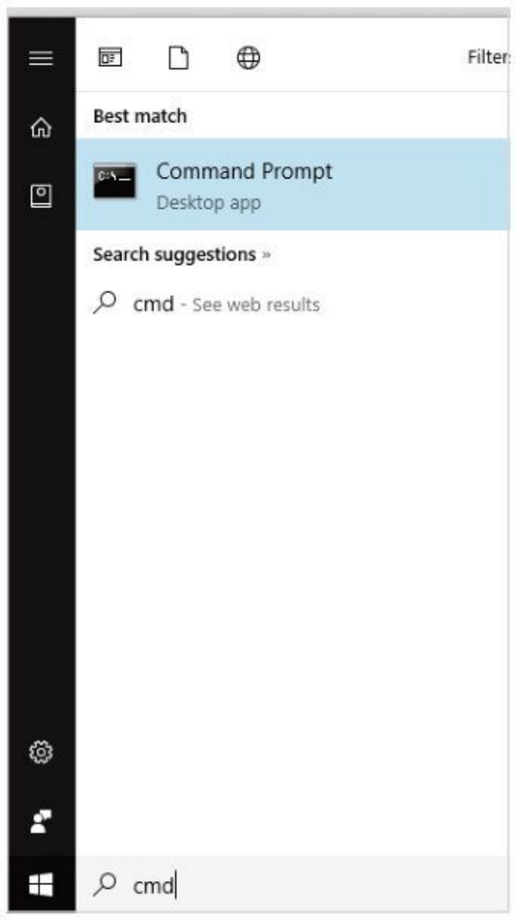
COMMAND THE CODE

Using the code we created in the previous tutorial, the one we named hello.py, let's see how you can run code that was made in the GUI at the command line level.

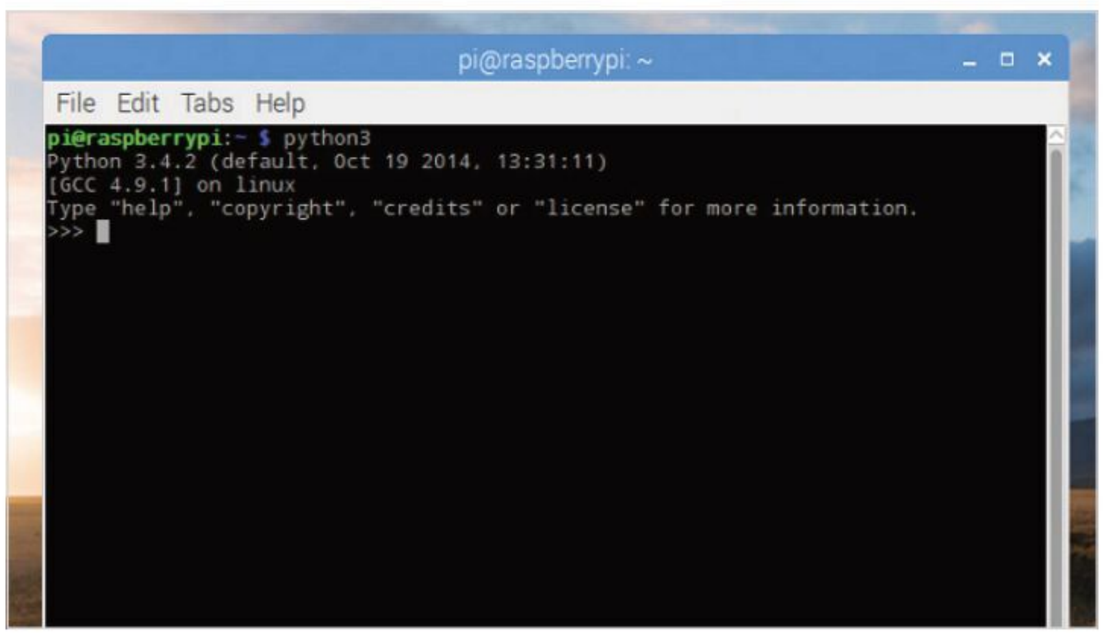
STEP 1 Python, in Linux, comes with two possible ways of executing code via the command line. One of the ways is with Python 2, whilst the other uses the Python 3 libraries and so on. First though, drop into the command line or Terminal on your operating system.



STEP 2 Just as before, we're using a Raspberry Pi: Windows users will need to click the Start button and search for CMD, then click the Command Line returned search; and macOS users can get access to their command line by clicking Go > Utilities > Terminal.



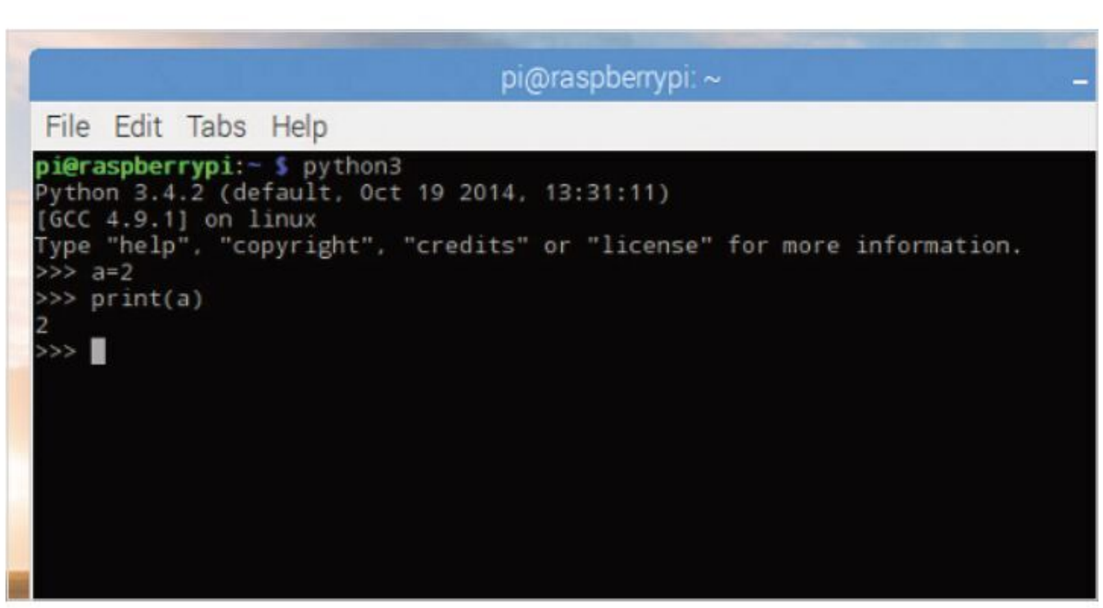
STEP 3 Now you're at the command line we can start Python. For Python 3 you need to enter the command **python3** and press Enter. This will put you into the command line version of the Shell, with the familiar three right-facing arrows as the cursor (>>>).



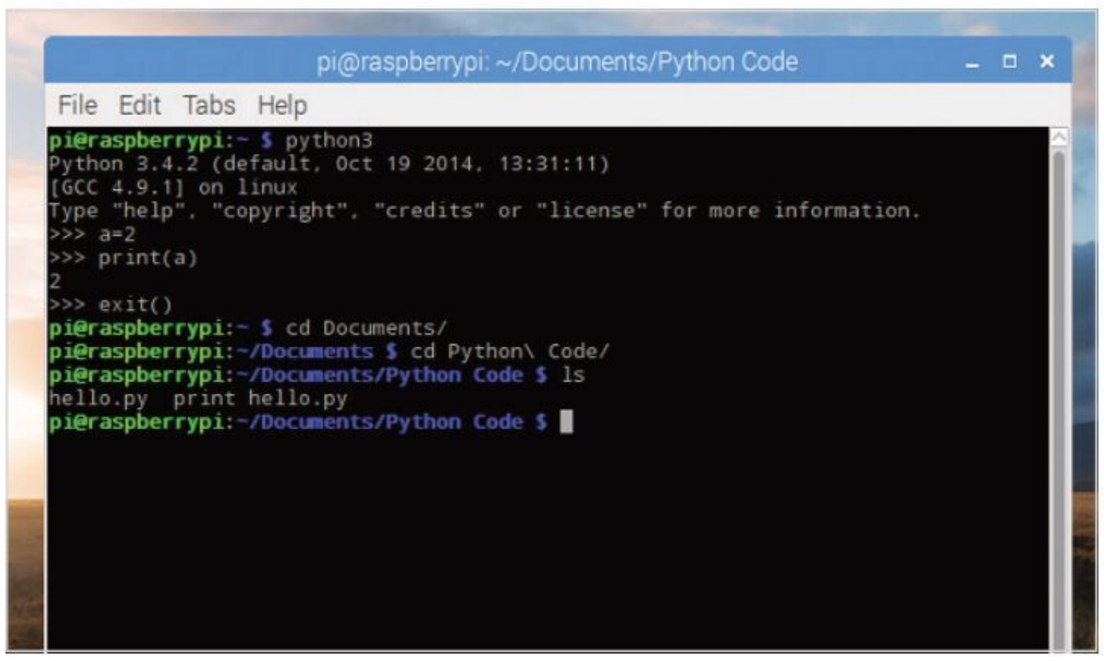
STEP 4 From here you're able to enter the code you've looked at previously, such as:

```
a=2
print(a)
```

You can see that it works exactly the same.



STEP 5 Now enter: **exit()** to leave the command line Python session and return you back to the command prompt. Enter the folder where you saved the code from the previous tutorial and list the available files within; hopefully you should see the hello.py file.

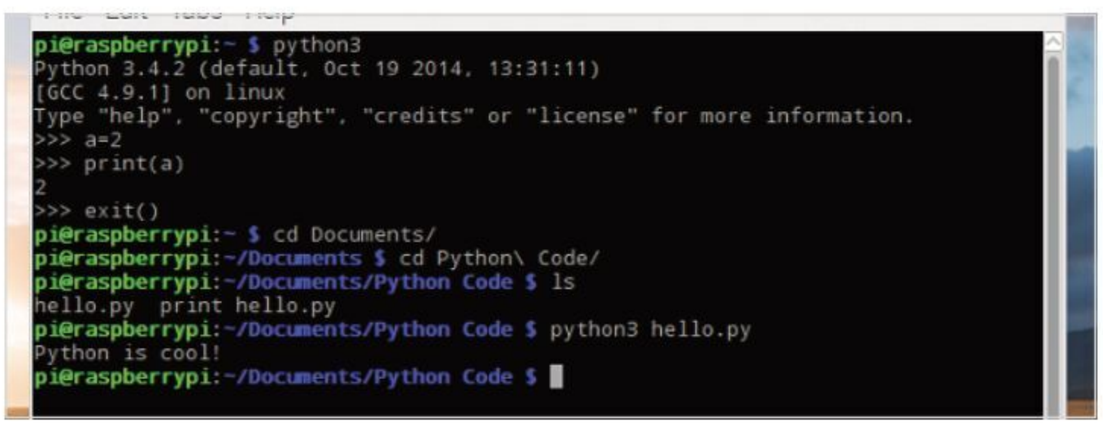


STEP 6 From within the same folder as the code you're going to run, enter the following into the command line:

```
python3 hello.py
```

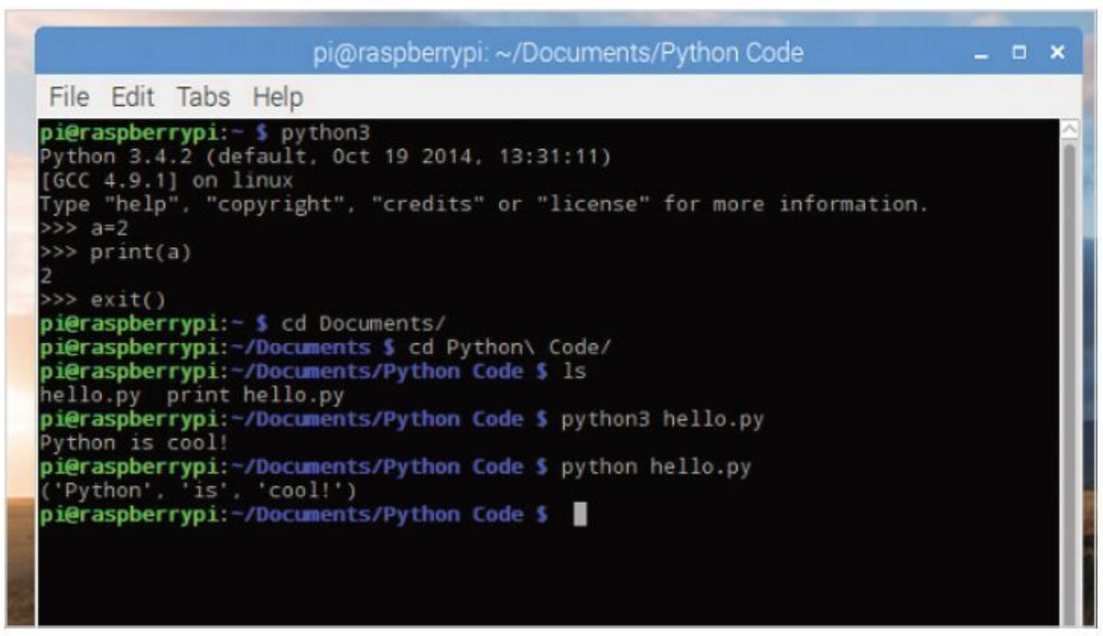
This will execute the code we created, which to remind you is:

```
a="Python"
b="is"
c="cool!"
print(a, b, c)
```

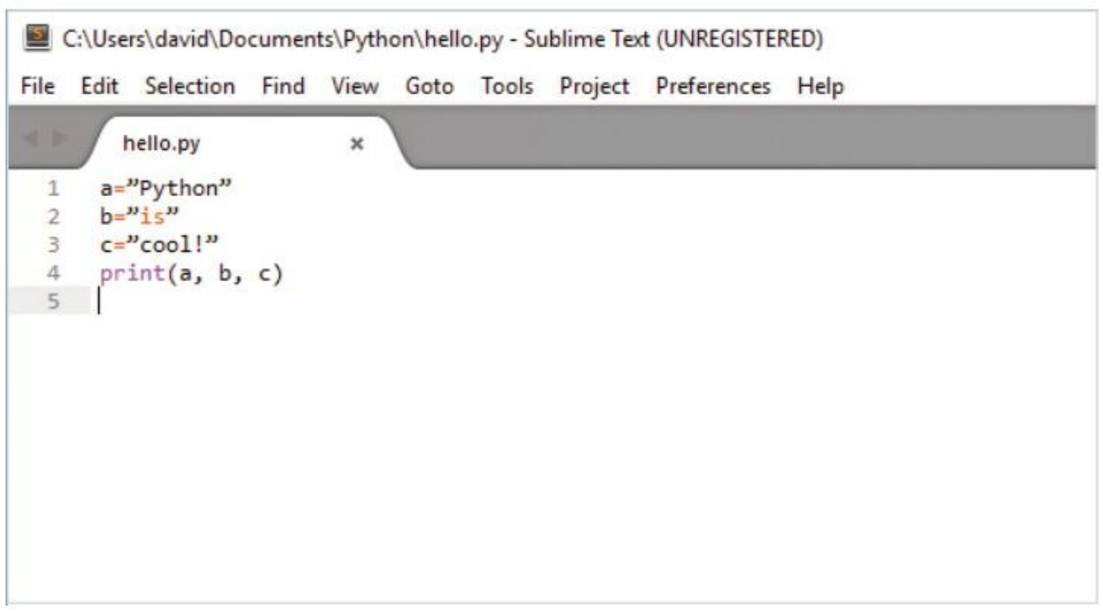


STEP 7 Naturally, since this is Python 3 code, using the syntax and layout that's unique to Python 3, it only works when you use the python3 command. If you like, try the same with Python 2 by entering:

```
python hello.py
```

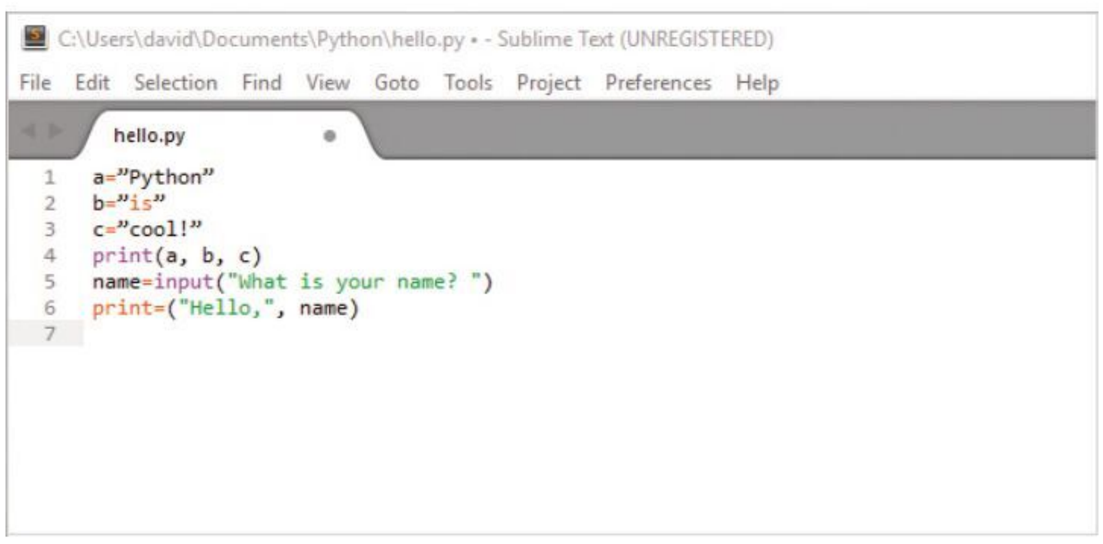


STEP 8 The result of running Python 3 code from the Python 2 command line is quite obvious. Whilst it doesn't error out in any way, due to the differences between the way Python 3 handles the Print command over Python 2, the result isn't as we expected. Using Sublime for the moment, open the hello.py file.



STEP 9 Since Sublime Text isn't available for the Raspberry Pi, you're going to temporarily leave the Pi for the moment and use Sublime as an example that you don't necessarily need to use the Python IDLE. With the hello.py file open, alter it to include the following:

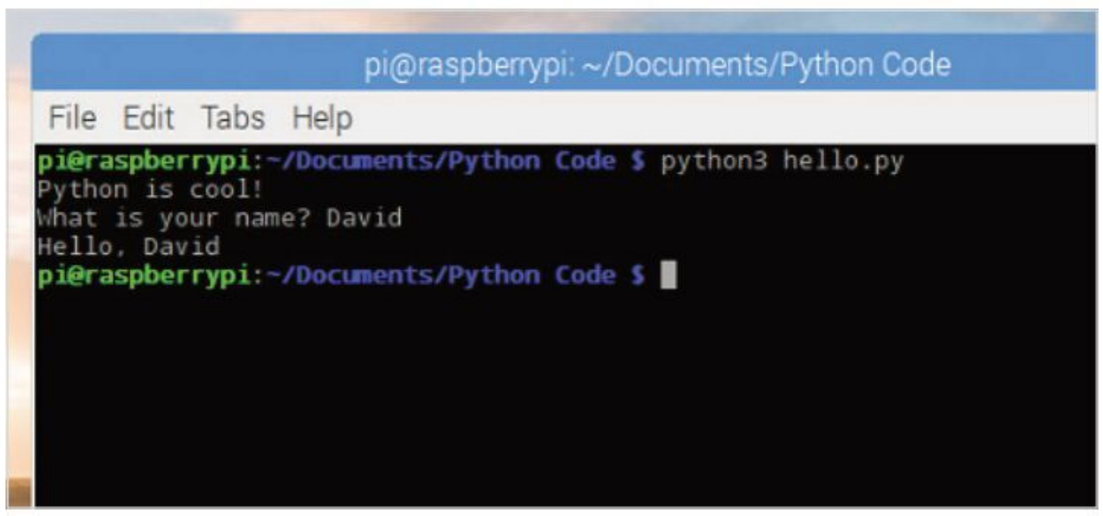
```
name=input("What is your name? ")
print("Hello,", name)
```



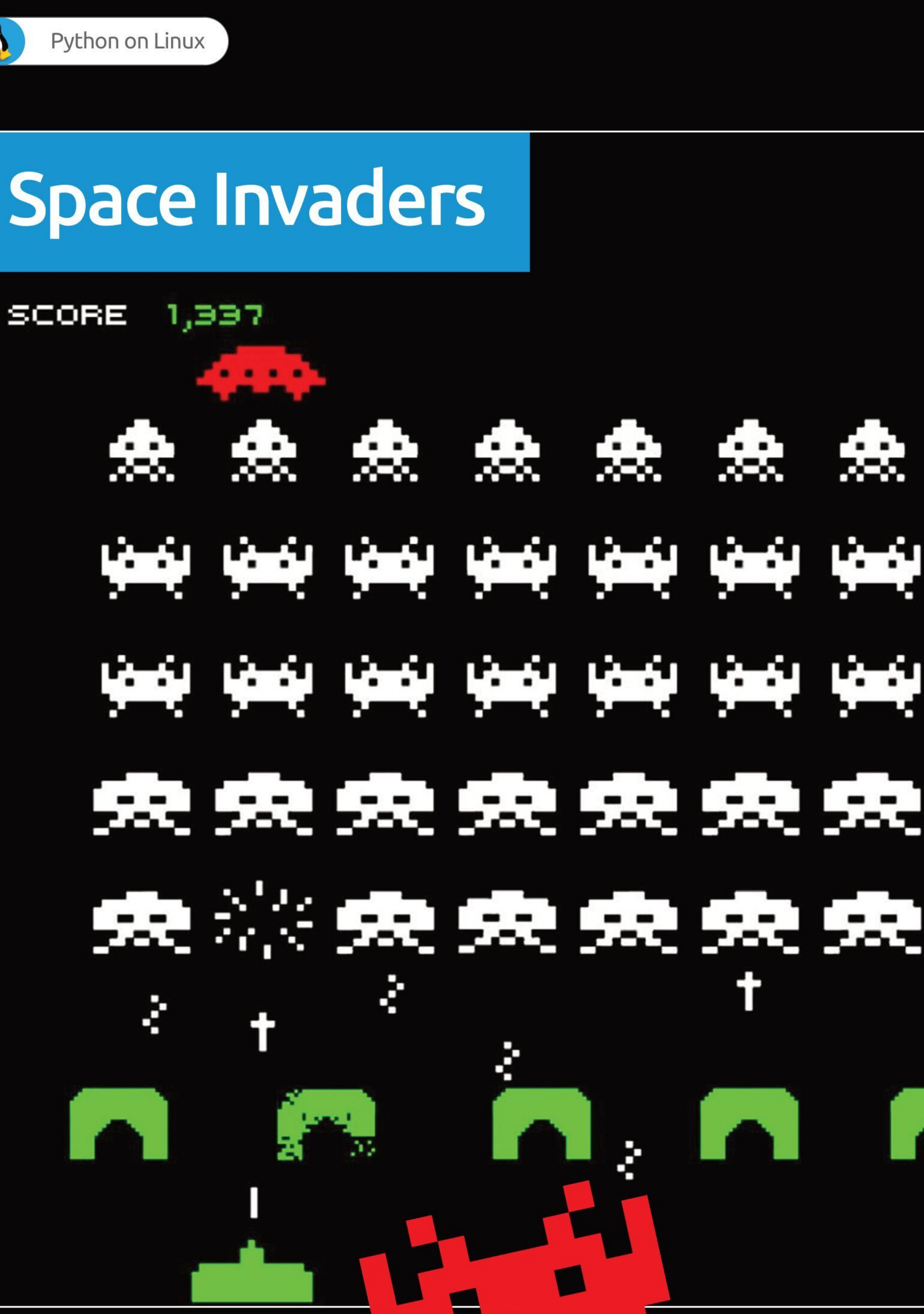
STEP 10 Save the hello.py file and drop back to the command line. Now execute the newly saved code with:

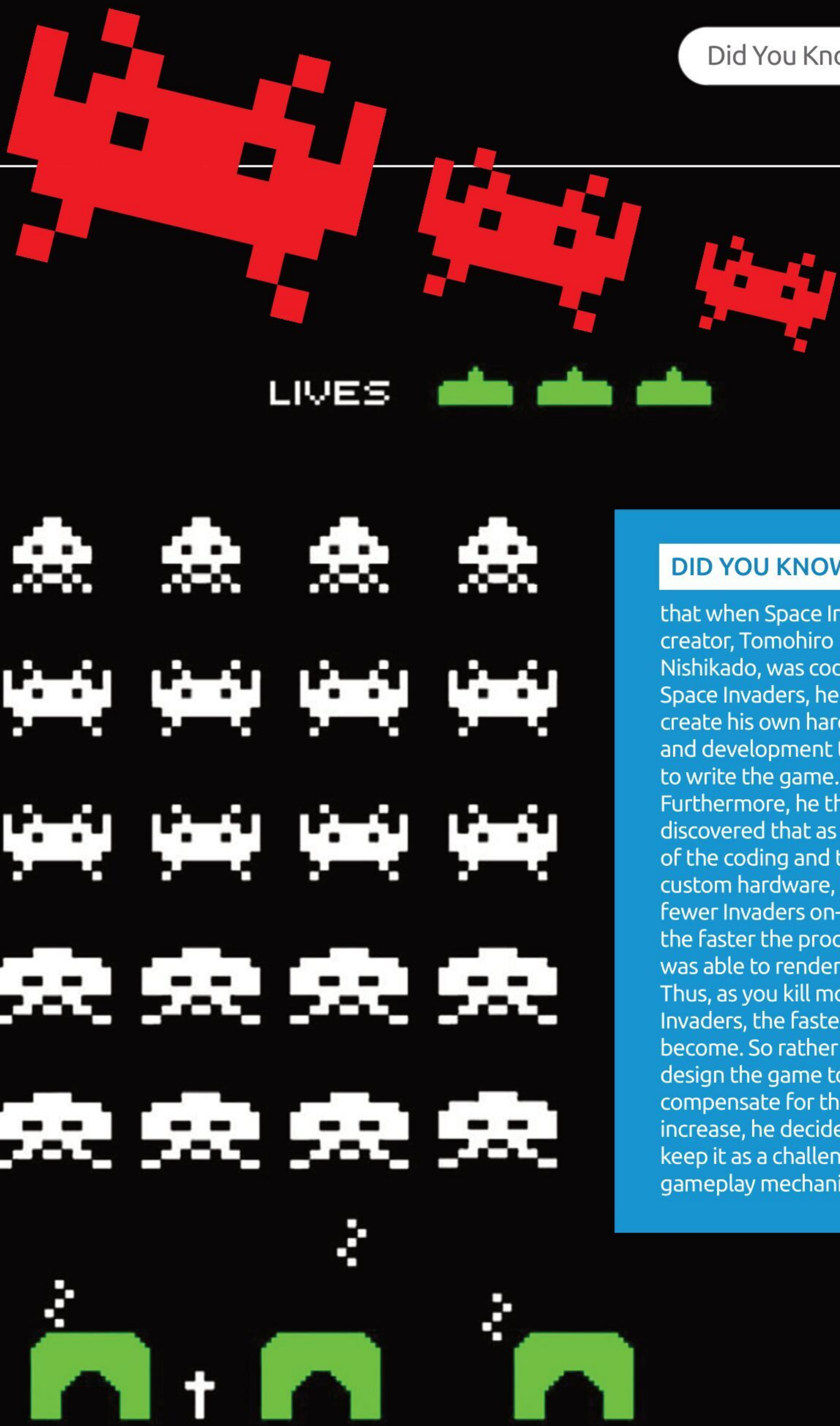
```
python3 hello.py
```

The result will be the original Python is cool! statement, together with the added input command asking you for your name, and displaying it in the command window.



Space Invaders





DID YOU KNOW...

that when Space Invaders creator, Tomohiro Nishikado, was coding Space Invaders, he had to create his own hardware and development tools to write the game. Furthermore, he then discovered that as a result of the coding and the custom hardware, the fewer Invaders on-screen the faster the processor was able to render them. Thus, as you kill more Invaders, the faster they become. So rather than design the game to compensate for the speed increase, he decided to keep it as a challenging gameplay mechanism.

It was pure accident that the Invaders got faster the less of them there were.

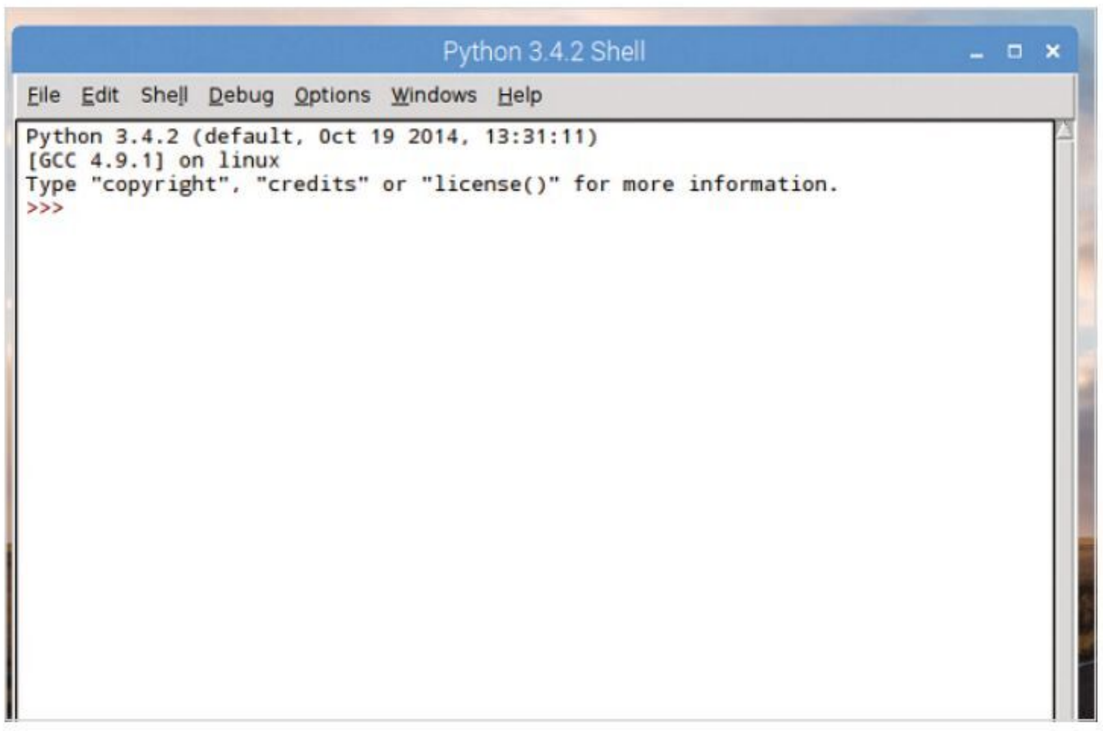
Numbers and Expressions

We've seen some basic mathematical expressions with Python, simple addition and the like. Let's expand on that now and see just how powerful Python is as a calculator. You can work within the IDLE Shell or in the Editor, whichever you like.

IT'S ALL MATHS, MAN

You can get some really impressive results with the mathematical powers of Python; as with most, if not all, programming languages, maths is the driving force behind the code.

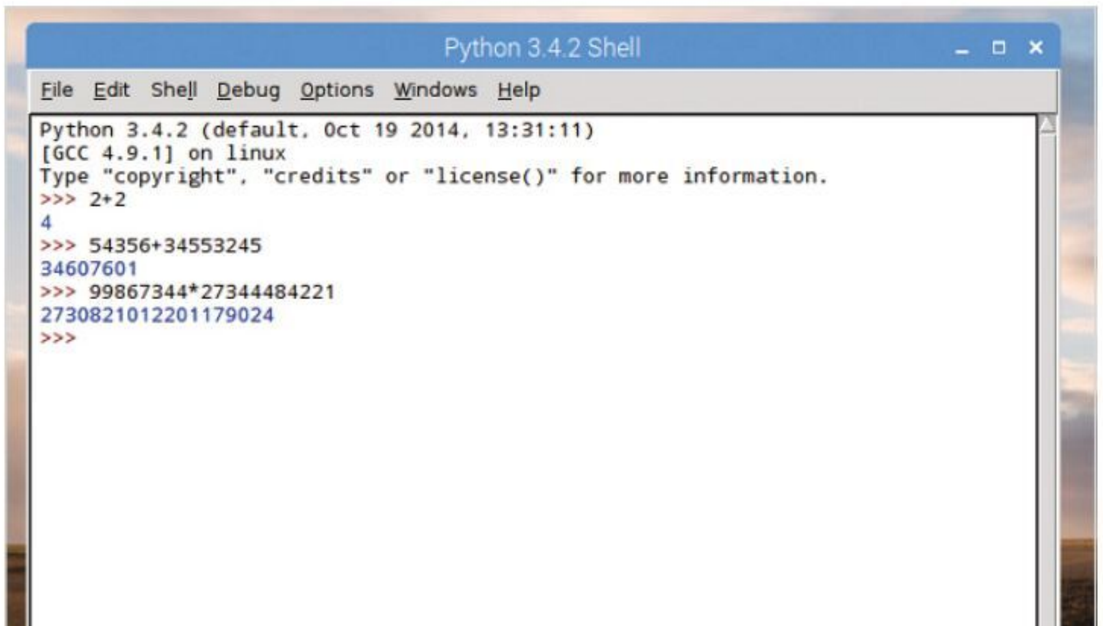
STEP 1 Open up the GUI version of Python 3, as mentioned you can use either the Shell or the Editor. For the time being, you're going to use the Shell just to warm our maths muscle, which we believe is a small gland located at the back of the brain (or not).



STEP 2 In the Shell enter the following:

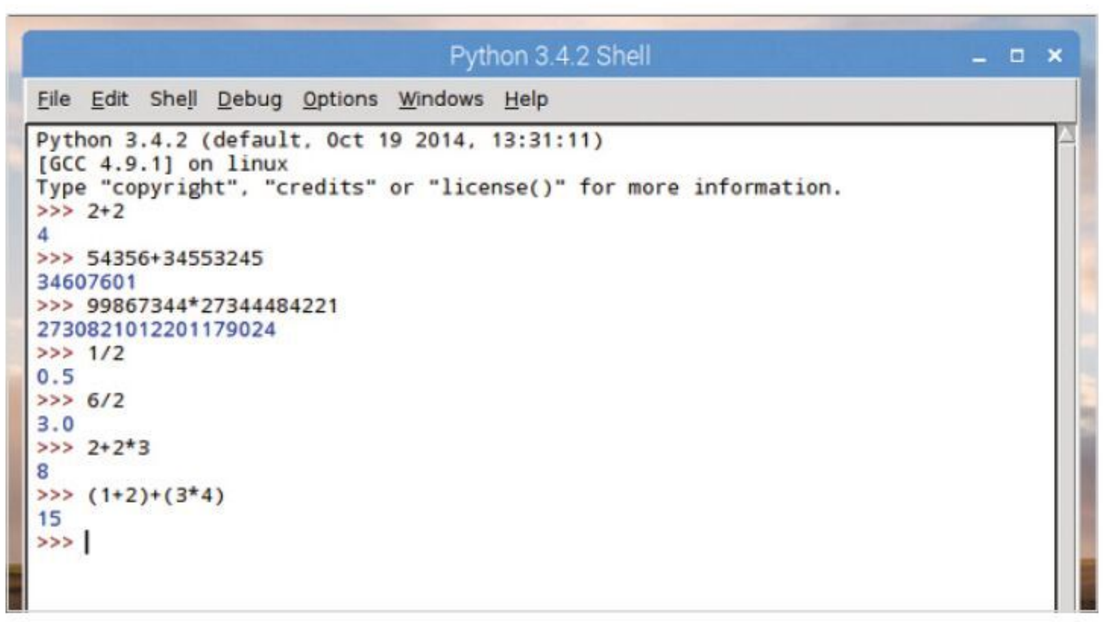
```
2+2
54356+34553245
99867344*27344484221
```

You can see that Python can handle some quite large numbers.



STEP 3 You can use all the usual mathematical operations: divide, multiply, brackets and so on. Practise with a few, for example:

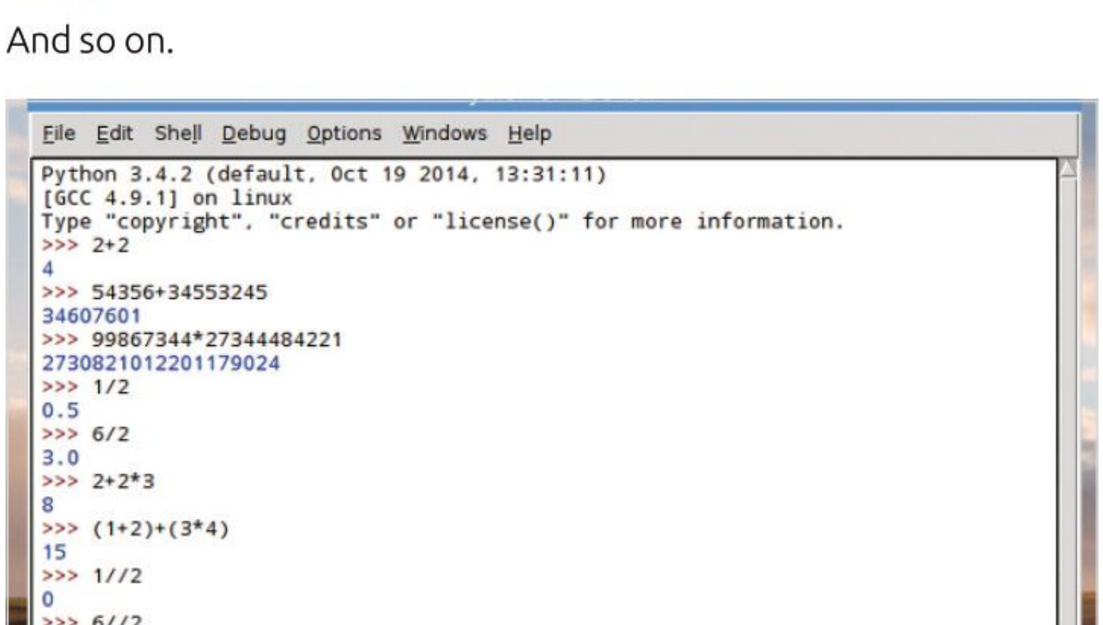
```
1/2
6/2
2+2*3
(1+2)+(3*4)
```



STEP 4 You've no doubt noticed, division produces a decimal number. In Python these are called floats, or floating point arithmetic. However, if you need an integer as opposed to a decimal answer, then you can use a double slash:

```
1//2
6//2
```

And so on.



STEP 5

You can also use an operation to see the remainder left over from division. For example:

```
10/3
```

Will display 3.333333333, which is of course 3.3-recurring. If you now enter:

```
10%3
```

This will display 1, which is the remainder left over from dividing 10 into 3.

```
2730821012201179024
>>> 1/2
0.5
>>> 6/2
3.0
>>> 2+2*3
8
>>> (1+2)+(3*4)
15
>>> 1//2
0
>>> 6//2
3
>>> 10/3
3.3333333333333335
>>> 10%3
1
```

STEP 6

Next up we have the power operator, or exponentiation if you want to be technical. To work out the power of something you can use a double multiplication symbol or double-star on the keyboard:

```
2**3
10**10
```

Essentially, it's 2x2x2 but we're sure you already know the basics behind maths operators. This is how you would work it out in Python.

```
>>> 6/2
3.0
>>> 2+2*3
8
>>> (1+2)+(3*4)
15
>>> 1//2
0
>>> 6//2
3
>>> 10/3
3.3333333333333335
>>> 10%3
1
>>> 2**3
8
>>> 10**10
10000000000
>>>
```

STEP 7

Numbers and expressions don't stop there. Python has numerous built-in functions to work out sets of numbers, absolute values, complex numbers and a host of mathematical expressions and Pythagorean tongue-twisters. For example, to convert a number to binary, use:

```
bin(3)
```

```
>>> 1/2
0.5
>>> 6/2
3.0
>>> 2+2*3
8
>>> (1+2)+(3*4)
15
>>> 1//2
0
>>> 6//2
3
>>> 10/3
3.3333333333333335
>>> 10%3
1
>>> 2**3
8
>>> 10**10
10000000000
>>> bin(3)
'0b11'
>>>
```

STEP 8

This will be displayed as '0b11', converting the integer into binary and adding the prefix 0b to the front. If you want to remove the 0b prefix, then you can use:

```
format(3, 'b')
```

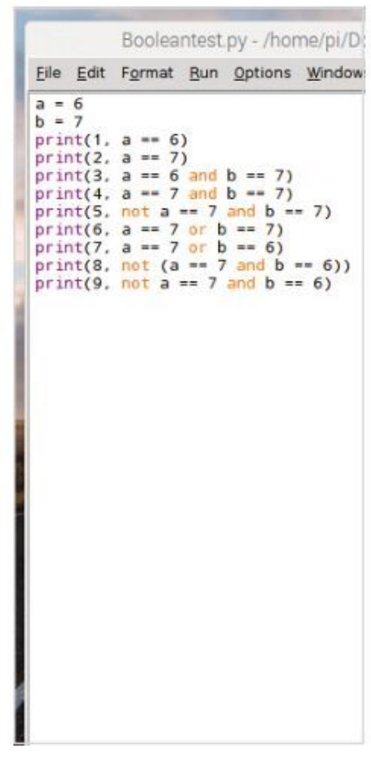
The Format command converts a value, the number 3, to a formatted representation as controlled by the format specification, the 'b' part.

```
>>> 2+2*3
8
>>> (1+2)+(3*4)
15
>>> 1//2
0
>>> 6//2
3
>>> 10/3
3.3333333333333335
>>> 10%3
1
>>> 2**3
8
>>> 10**10
10000000000
>>> bin(3)
'0b11'
>>> format(3,'b')
'11'
>>>
```

STEP 9

A Boolean Expression is a logical statement that will either be true or false. We can use these to compare data and test to see if it's equal to, less than or greater than. Try this in a New File:

```
a = 6
b = 7
print(1, a == 6)
print(2, a == 7)
print(3, a == 6 and b == 7)
print(4, a == 7 and b == 7)
print(5, not a == 7 and b == 7)
print(6, a == 7 or b == 7)
print(7, a == 7 or b == 6)
print(8, not (a == 7 and b == 6))
print(9, not a == 7 and b == 6)
```



STEP 10

Execute the code from Step 9, and you can see a series of True or False statements, depending on the result of the two defining values: 6 and 7. It's an extension of what you've looked at, and an important part of programming.

```
Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
1 True
2 False
3 True
4 False
5 True
6 True
7 False
8 True
9 False
>>>
```

Using Comments

When writing your code, the flow, what each variable does, how the overall program will operate and so on is all inside your head. Another programmer could follow the code line by line but over time, it can become difficult to read.

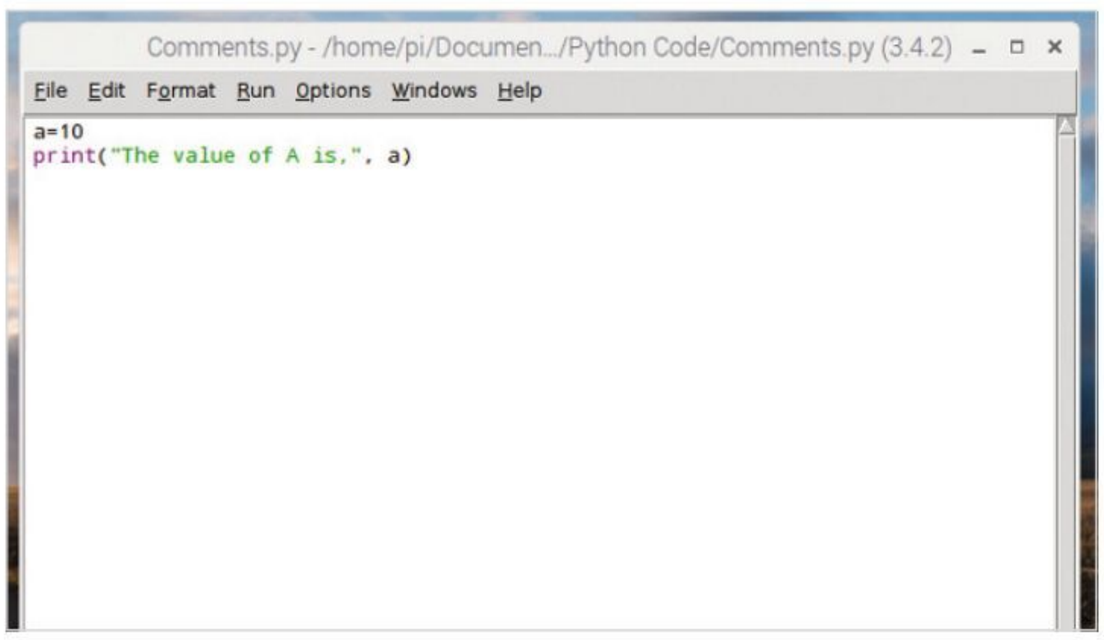
#COMMENTS!

Programmers use a method of keeping their code readable by commenting on certain sections. If a variable is used, the programmer comments on what it's supposed to do, for example. It's just good practise.

STEP 1 Start by creating a new instance of the IDLE Editor (File > New File) and create a simple variable and print command:

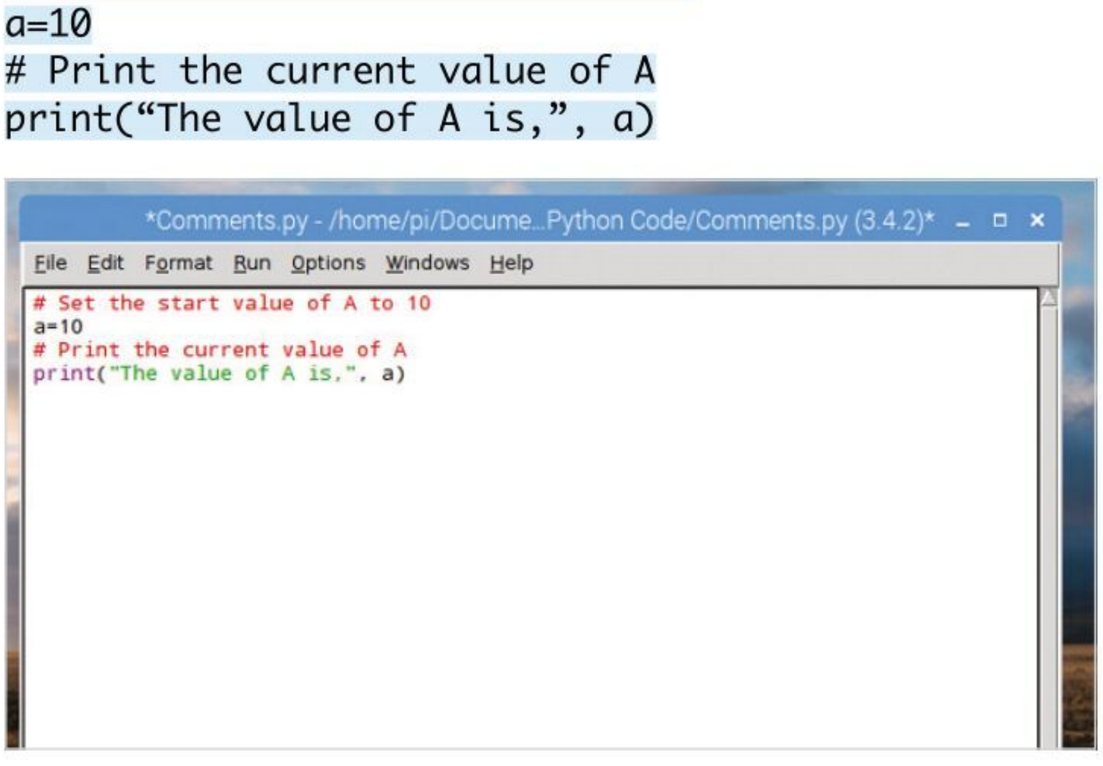
```
a=10
print("The value of A is,", a)
```

Save the file and execute the code.

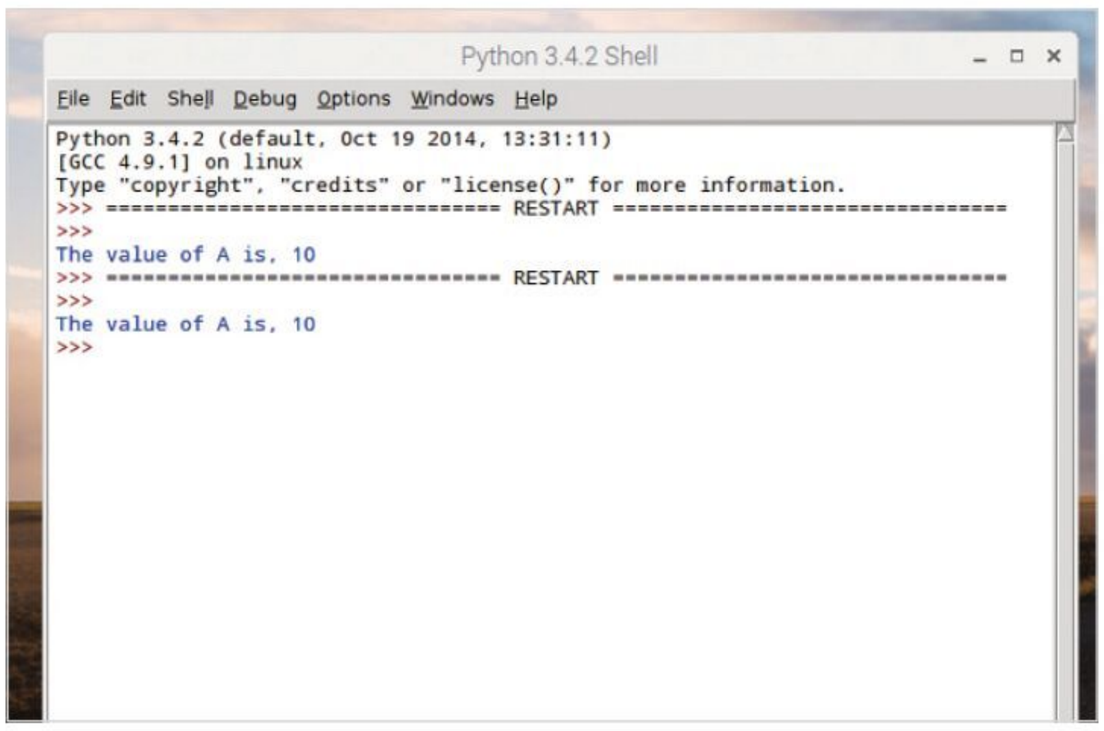


STEP 2 Running the code will return the line: The value of A is, 10 into the IDLE Shell window, which is what we expected. Now, add some of the types of comments you'd normally see within code:

```
# Set the start value of A to 10
a=10
# Print the current value of A
print("The value of A is,", a)
```

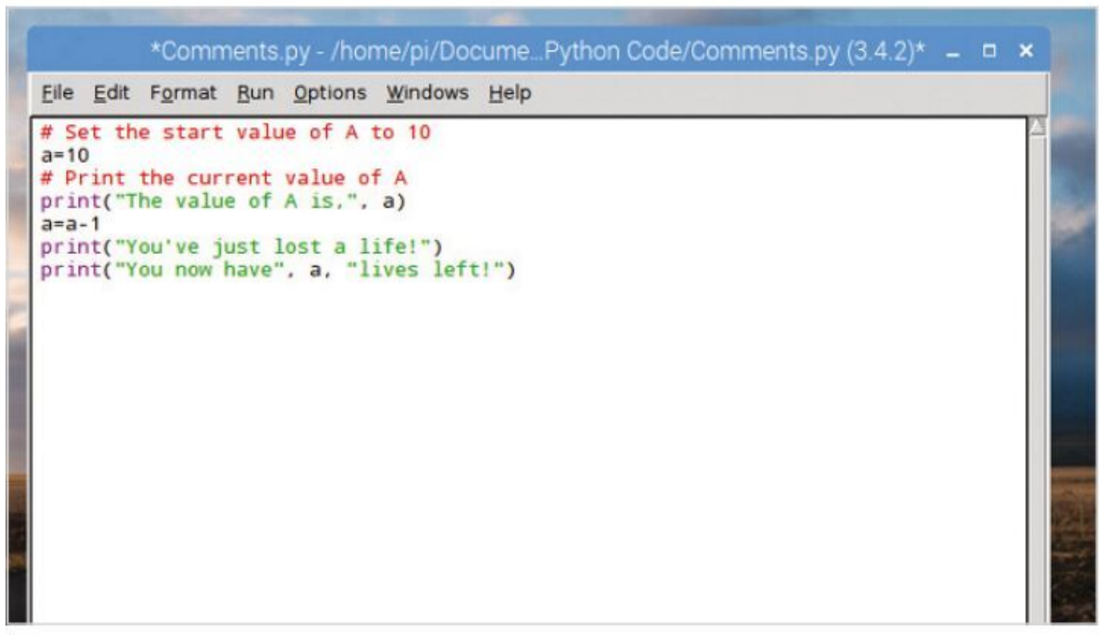


STEP 3 Resave the code and execute it. You can see that the output in the IDLE Shell is still the same as before, despite the extra lines being added. Simply put, the hash symbol (#) denotes a line of text the programmer can insert to inform them, and others, of what's going on without the user being aware.

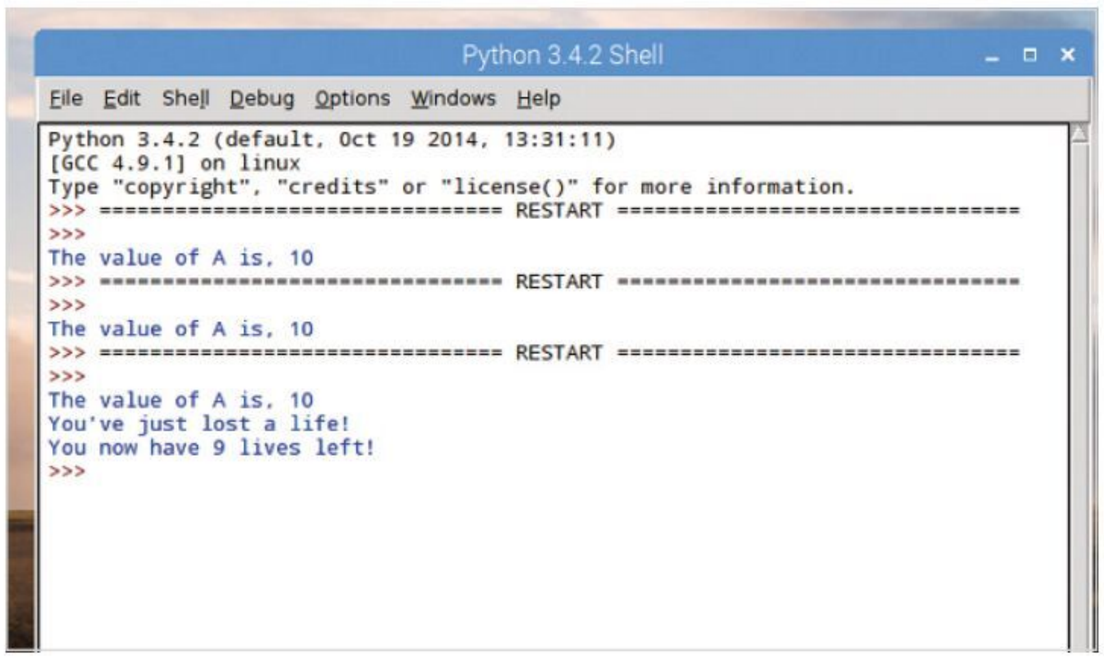


STEP 4 Let's assume that the variable A that we've created is the number of lives in a game. Every time the player dies, the value is decreased by 1. The programmer could insert a routine along the lines of:

```
a=a-1
print("You've just lost a life!")
print("You now have", a, "lives left!")
```

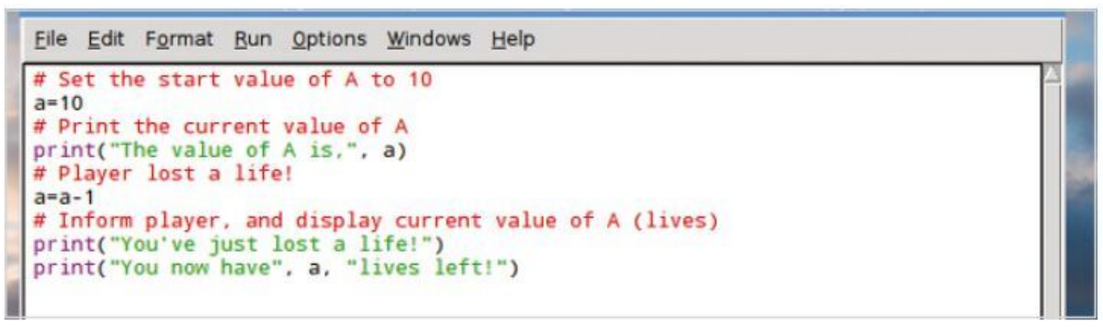


STEP 5 Whilst we know that the variable A is lives, and that the player has just lost one, a casual viewer or someone checking the code may not know. Imagine for a moment that the code is twenty thousand lines long, instead of just our seven. You can see how handy comments are.



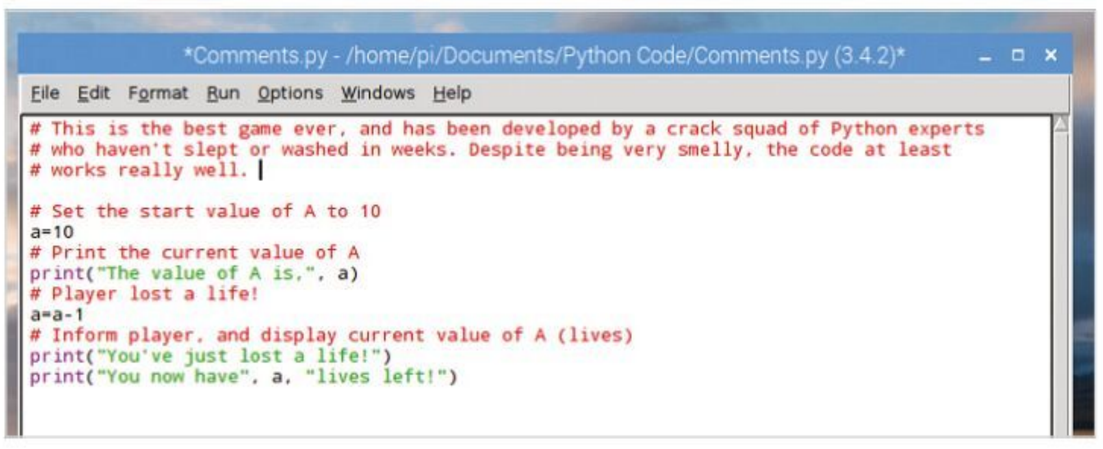
STEP 6 Essentially, the new code together with comments could look like:

```
# Set the start value of A to 10
a=10
# Print the current value of A
print(“The value of A is,”, a)
# Player lost a life!
a=a-1
# Inform player, and display current value of A (lives)
print(“You’ve just lost a life!”)
print(“You now have”, a, “lives left!”)
```



STEP 7 You can use comments in different ways. For example, Block Comments are a large section of text that details what’s going on in the code, such as telling the code reader what variables you’re planning on using:

```
# This is the best game ever, and has been
developed by a crack squad of Python experts
# who haven’t slept or washed in weeks. Despite
being very smelly, the code at least
# works really well.
```

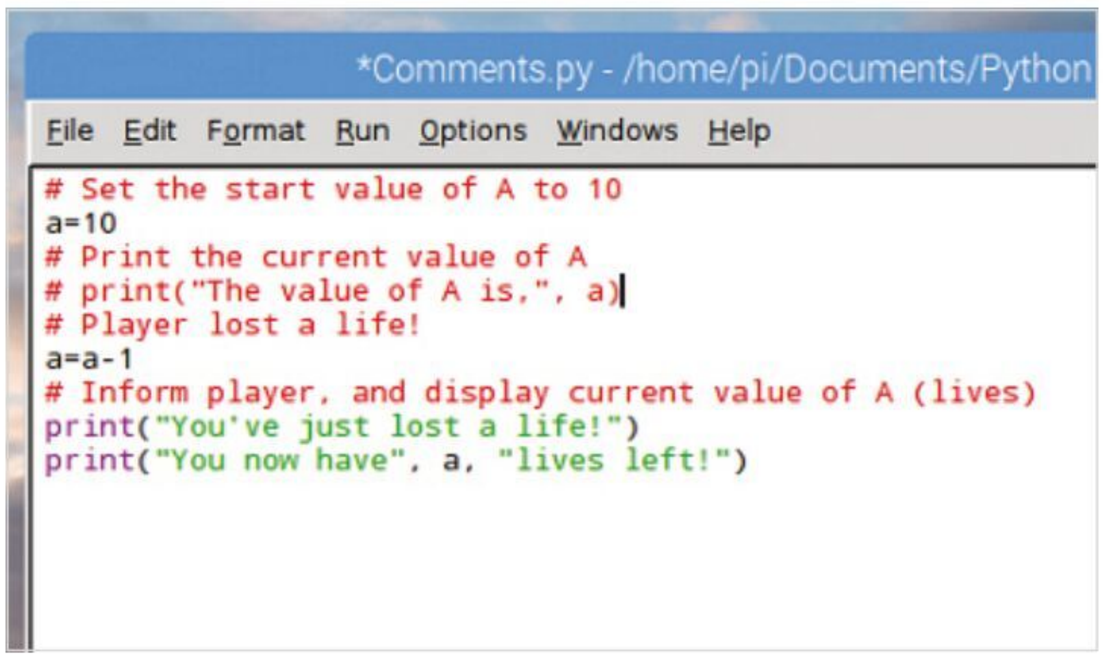


STEP 8 Inline comments are comments that follow a section of code. Take our examples from above, instead of inserting the code on a separate line, we could use:

```
a=10 # Set the start value of A to 10
print(“The value of A is,”, a) # Print the current
value of A
a=a-1 # Player lost a life!
print(“You’ve just lost a life!”)
print(“You now have”, a, “lives left!”) # Inform
player, and display current value of A (lives)
```

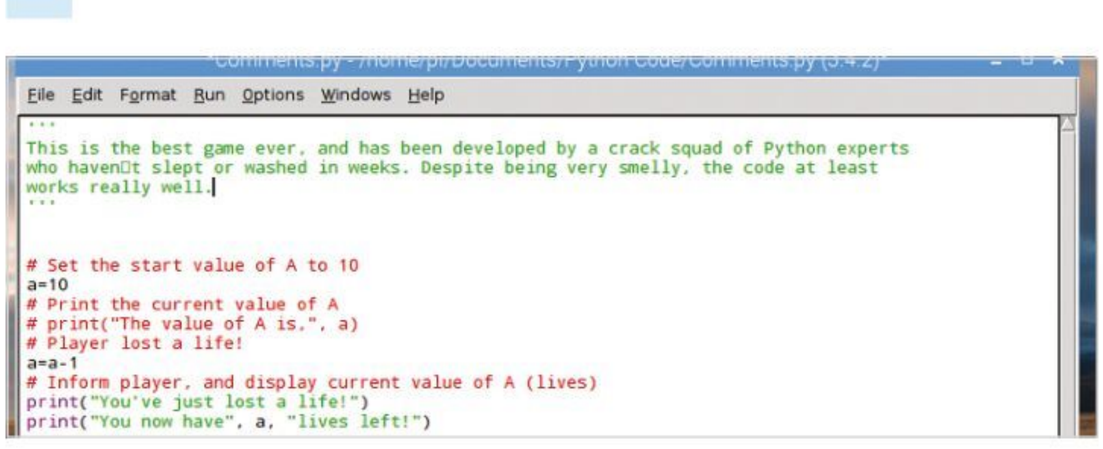
STEP 9 The comment, the hash symbol, can also be used to comment out sections of code you don’t want to be executed in your program. For instance, if you wanted to remove the first print statement, you would use:

```
# print(“The value of A is,”, a)
```



STEP 10 You also use three single quotes to comment out a Block Comment or multi-line section of comments. Place them before and after the areas you want to comment for them to work:

```
’’’
This is the best game ever, and has been developed
by a crack squad of Python experts who haven’t
slept or washed in weeks. Despite being very
smelly, the code at least works really well.
’’’
```

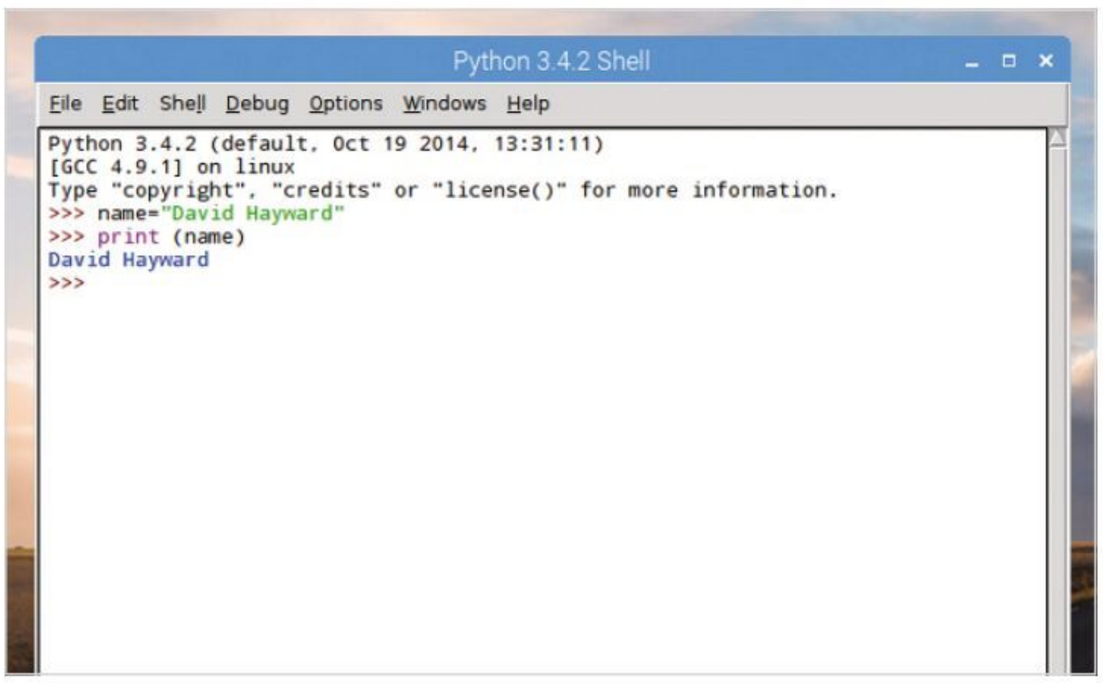
Working with Variables

We've seen some examples of variables in our Python code already but it's always worth going through the way they operate and how Python creates and assigns certain values to a variable.

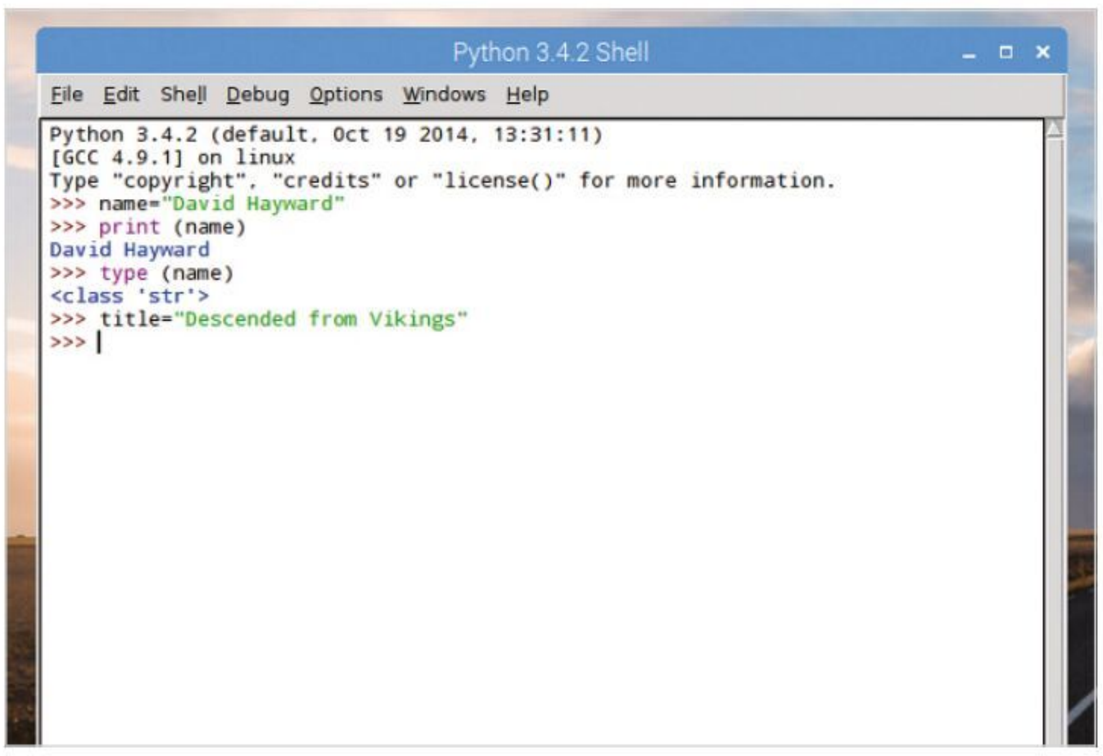
VARIOUS VARIABLES

You'll be working with the Python 3 IDLE Shell in this tutorial. If you haven't already, open Python 3 or close down the previous IDLE Shell to clear up any old code.

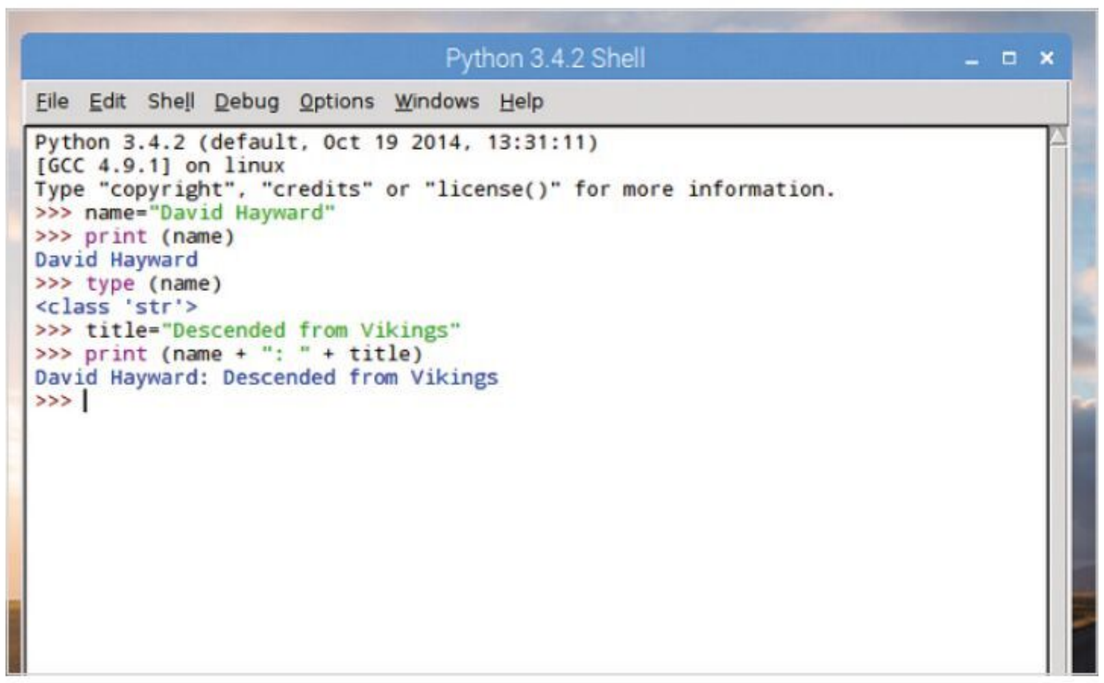
STEP 1 In some programming languages you're required to use a dollar sign to denote a string, which is a variable made up of multiple characters, such as a name of a person. In Python this isn't necessary. For example, in the Shell enter: `name="David Hayward"` (or use your own name, unless you're also called David Hayward).



STEP 2 You can check the type of variable in use by issuing the **type ()** command, placing the name of the variable inside the brackets. In our example, this would be: `type (name)`. Add a new string variable: `title="Descended from Vikings"`.



STEP 3 You've seen previously that variables can be concatenated using the plus symbol between the variable names. In our example we can use: `print (name + ": " + title)`. The middle part between the quotations allows us to add a colon and a space, as variables are connected without spaces, so we need to add them manually.



STEP 4 You can also combine variables within another variable. For example, to combine both name and title variables into a new variable we use:

```
character=name + ": " + title
```

Then output the content of the new variable as:

```
print (character)
```

Numbers are stored as different variables:

```
age=44
Type (age)
```

Which, as we know, are integers.

STEP 5 However, you can't combine both strings and integer type variables in the same command, as you would a set of similar variables. You need to either turn one into the other or vice versa. When you do try to combine both, you get an error message:

```
print (name + age)
```

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> name="David Hayward"
>>> print (name)
David Hayward
>>> type (name)
<class 'str'>
>>> title="Descended from Vikings"
>>> print (name + ": " + title)
David Hayward: Descended from Vikings
>>> character=name + ": " + title
>>> print (character)
David Hayward: Descended from Vikings
>>> age=44
>>> type (age)
<class 'int'>
>>> print (name+age)
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    print (name+age)
TypeError: Can't convert 'int' object to str implicitly
>>>
```

STEP 6 This is a process known as TypeCasting. The Python code is:

```
print (character + " is " + str(age) + " years old.")
```

or you can use:

```
print (character, "is", age, "years old.")
```

Notice again that in the last example, you don't need the spaces between the words in quotes as the commas treat each argument to print separately.

```
>>> print (name + age)
Traceback (most recent call last):
  File "<pyshell#18>", line 1, in <module>
    print (name + age)
TypeError: Can't convert 'int' object to str implicitly
>>> print (character + " is " + str(age) + " years old.")
David Hayward: Descended from Vikings is 44 years old.
>>> print (character, "is", age, "years old.")
David Hayward: Descended from Vikings is 44 years old.
>>>
>>>
```

STEP 7 Another example of TypeCasting is when you ask for input from the user, such as a name. for example, enter:

```
age= input ("How old are you? ")
```

All data stored from the Input command is stored as a string variable.

```
Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> age= input ("How old are you? ")
How old are you? 44
>>> type(age)
<class 'str'>
>>>
```

STEP 8 This presents a bit of a problem when you want to work with a number that's been inputted by the user, as age + 10 won't work due to being a string variable and an integer. Instead, you need to enter:

```
int(age) + 10
```

This will TypeCast the age string into an integer that can be worked with.

```
Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> age= input ("How old are you? ")
How old are you? 44
>>> type(age)
<class 'str'>
>>> age + 10
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    age + 10
TypeError: Can't convert 'int' object to str implicitly
>>> int(age) + 10
54
>>>
```

STEP 9 The use of TypeCasting is also important when dealing with floating point arithmetic; remember: numbers that have a decimal point in them. For example, enter:

```
shirt=19.99
```

Now enter `type(shirt)` and you'll see that Python has allocated the number as a 'float', because the value contains a decimal point.

```
Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> shirt=19.99
>>> type(shirt)
<class 'float'>
>>>
```

STEP 10 When combining integers and floats Python usually converts the integer to a float, but should the reverse ever be applied it's worth remembering that Python doesn't return the exact value. When converting a float to an integer, Python will always round down to the nearest integer, called truncating; in our case instead of 19.99 it becomes 19.

```
Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> shirt=19.99
>>> type(shirt)
<class 'float'>
>>> int(shirt)
19
>>>
```

User Input

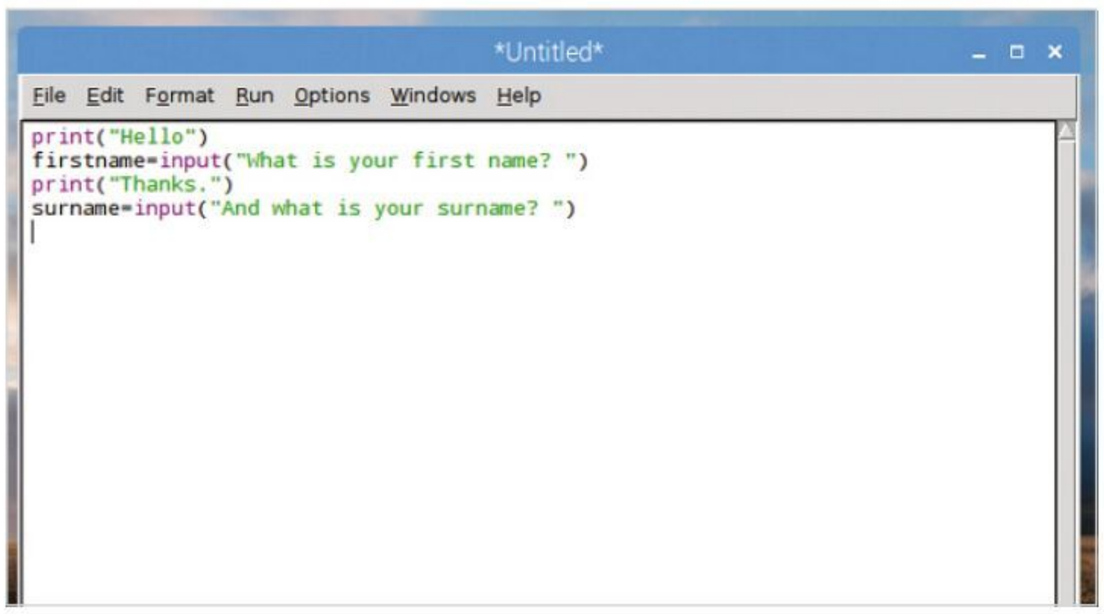
We've seen some basic user interaction with the code from a few of the examples earlier, so now would be a good time to focus solely on how you would get information from the user then store and present it.

USER FRIENDLY

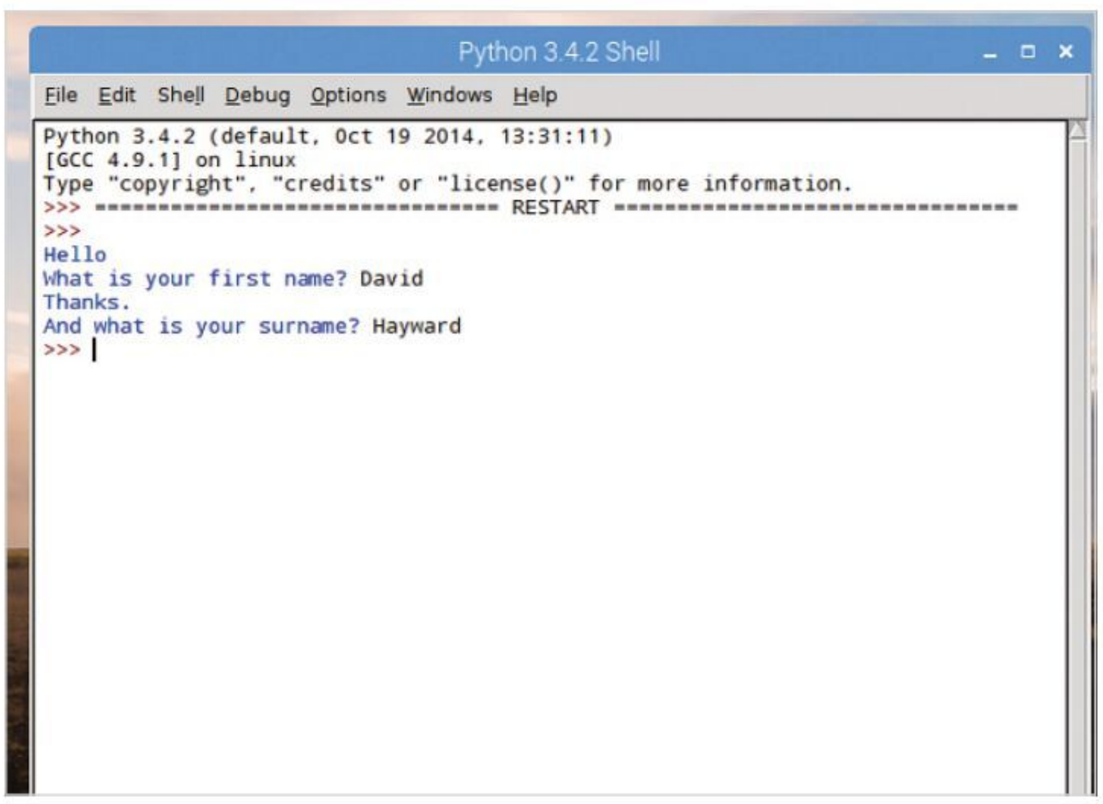
The type of input you want from the user will depend greatly on the type of program you're coding. For example, a game may ask for a character's name, whereas a database can ask for personal details.

STEP 1 If it's not already, open the Python 3 IDLE Shell, and start a New File in the Editor. Let's begin with something really simple, enter:

```
print("Hello")
firstname=input("What is your first name? ")
print("Thanks.")
surname=input("And what is your surname? ")
```

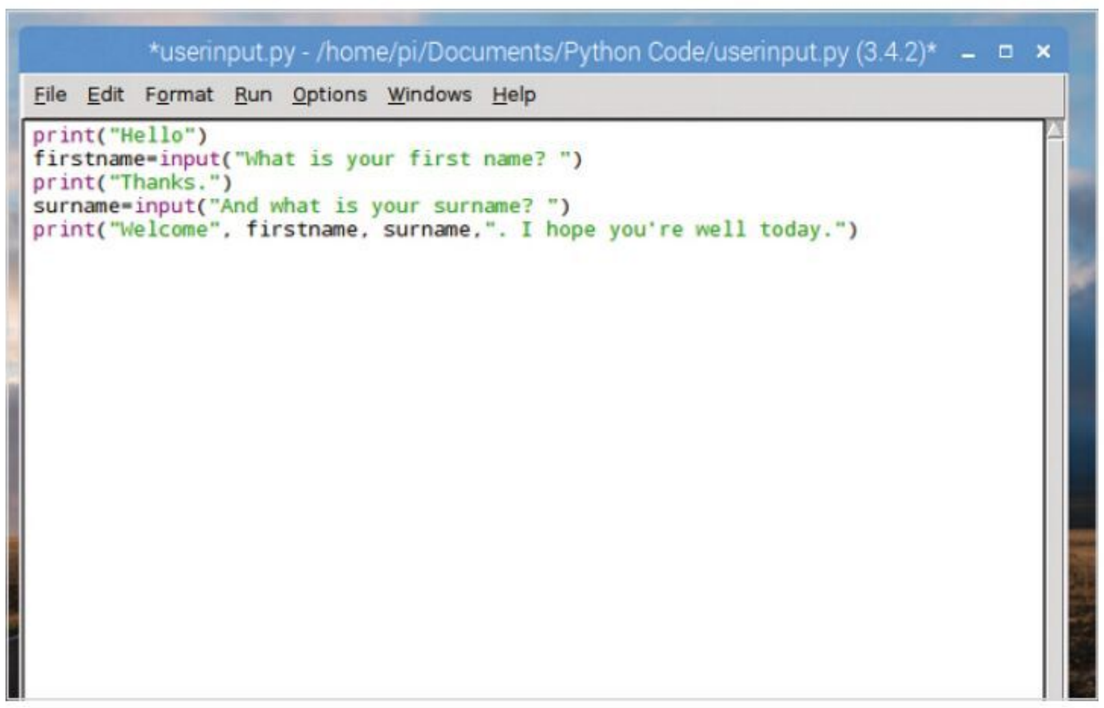


STEP 2 Save and execute the code, and as you already no doubt suspected, in the IDLE Shell the program will ask for your first name, storing it as the variable firstname, followed by your surname; also stored in its own variable (surname).



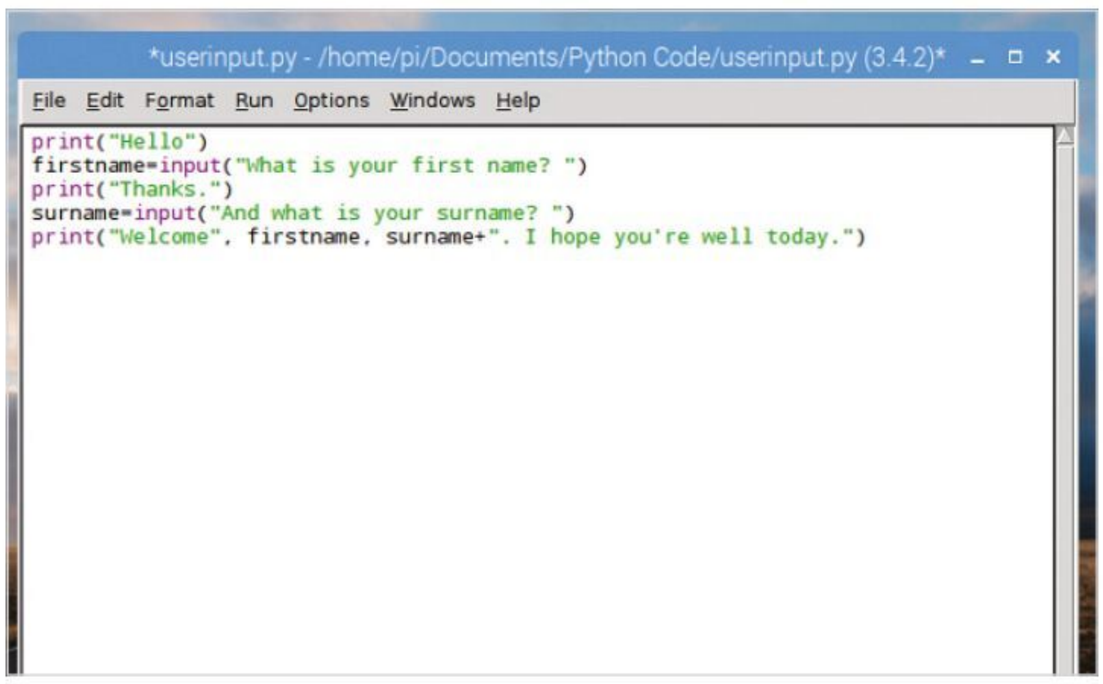
STEP 3 Now that we have the user's name stored in a couple of variables we can call them up whenever we want:

```
print("Welcome", firstname, surname, ". I hope
you're well today.")
```



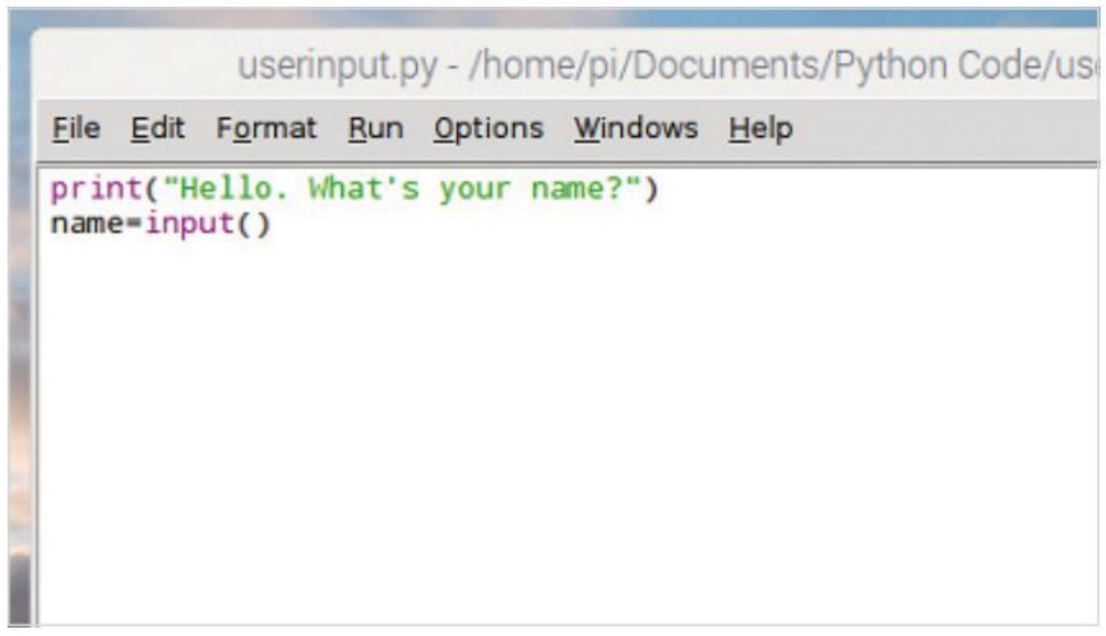
STEP 4 Run the code and you can see a slight issue, the full stop after the surname follows a blank space. To eliminate that we can add a plus sign instead of the comma in the code:

```
print("Welcome", firstname, surname+". I hope
you're well today.")
```



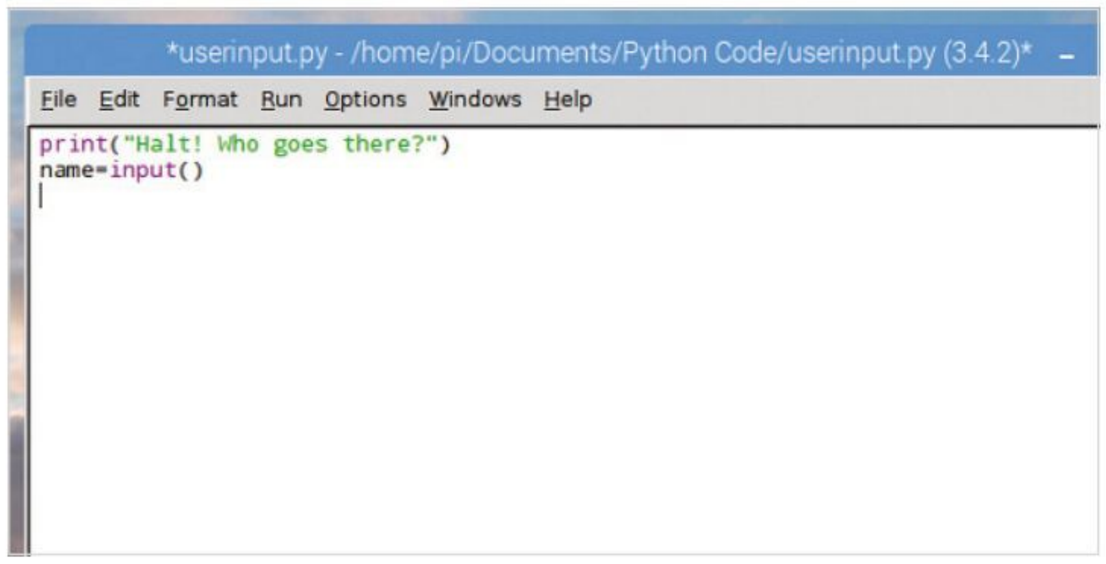
STEP 5 You don't always have to include quoted text within the input command. For example, you can ask the user their name, and have the input in the line below:

```
print("Hello. What's your name?")
name=input()
```



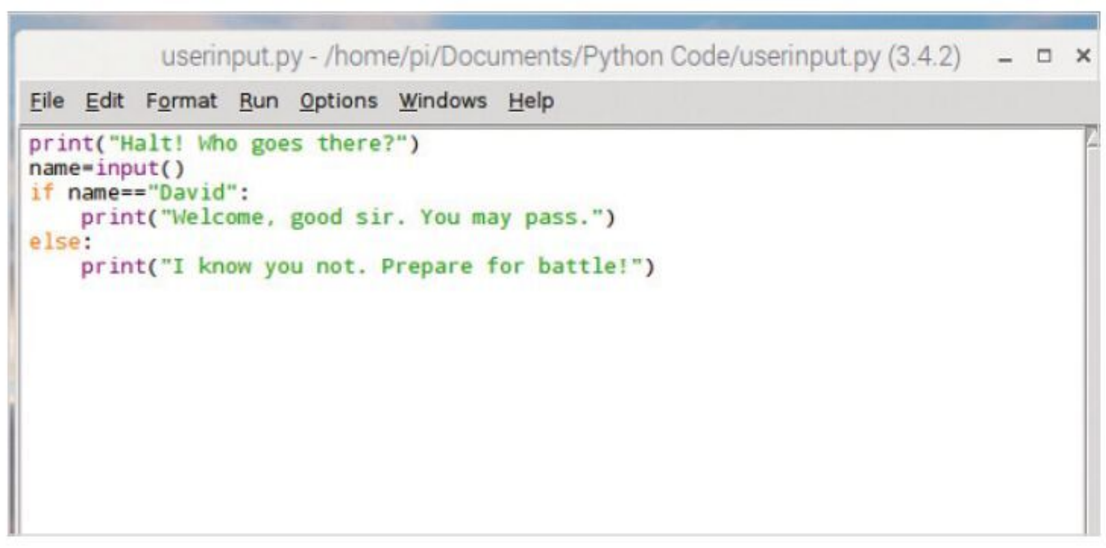
STEP 6 The code from the previous step is often regarded as being a little neater than having a lengthy amount of text in the input command, but it's not a rule that's set in stone, so do as you like in these situations. Expanding on the code, try this:

```
print("Halt! Who goes there?")
name=input()
```

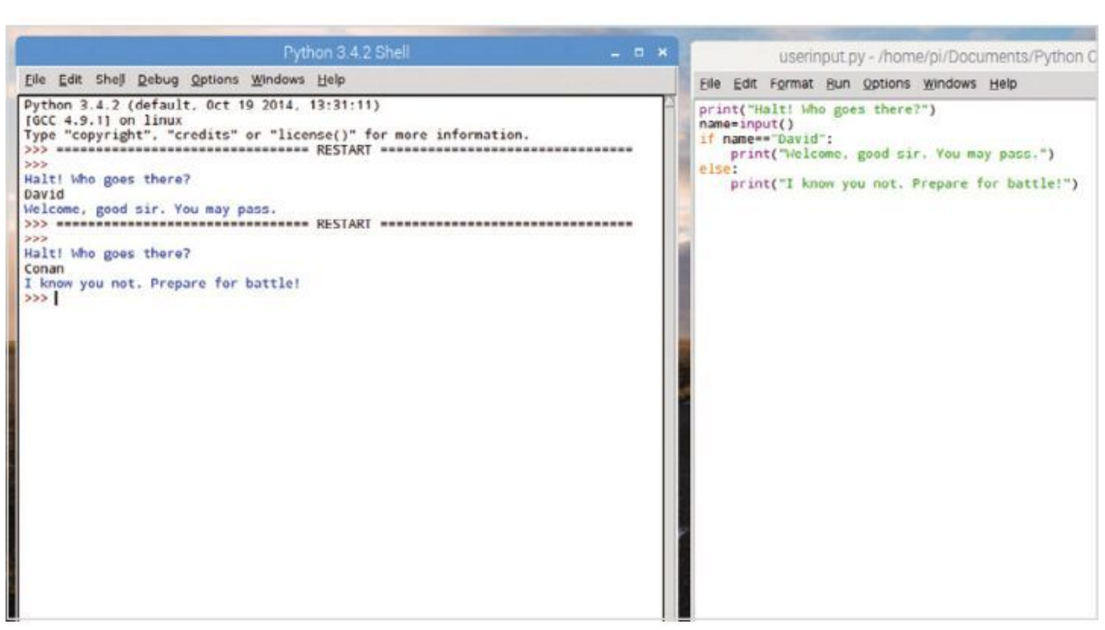


STEP 7 It's a good start to a text adventure game, perhaps? Now you can expand on it and use the raw input from the user to flesh out the game a little:

```
if name=="David":
  print("Welcome, good sir. You may pass.")
else:
  print("I know you not. Prepare for battle!")
```

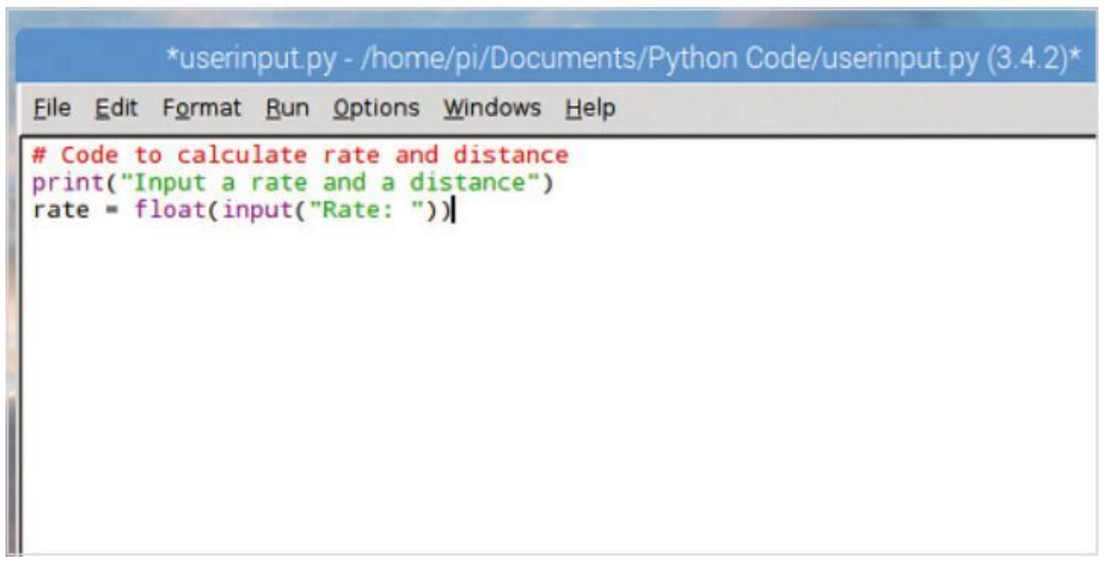


STEP 8 What you've created here is a condition, which we will cover soon. In short, we're using the input from the user and measuring it against a condition. So, if the user enters David as their name, the guard will allow them to pass unhindered. Else, if they enter a name other than David, the guard challenges them to a fight.



STEP 9 Just as you learned previously, any input from a user is automatically a string, so you need to apply a TypeCast in order to turn it into something else. This creates some interesting additions to the input command. For example:

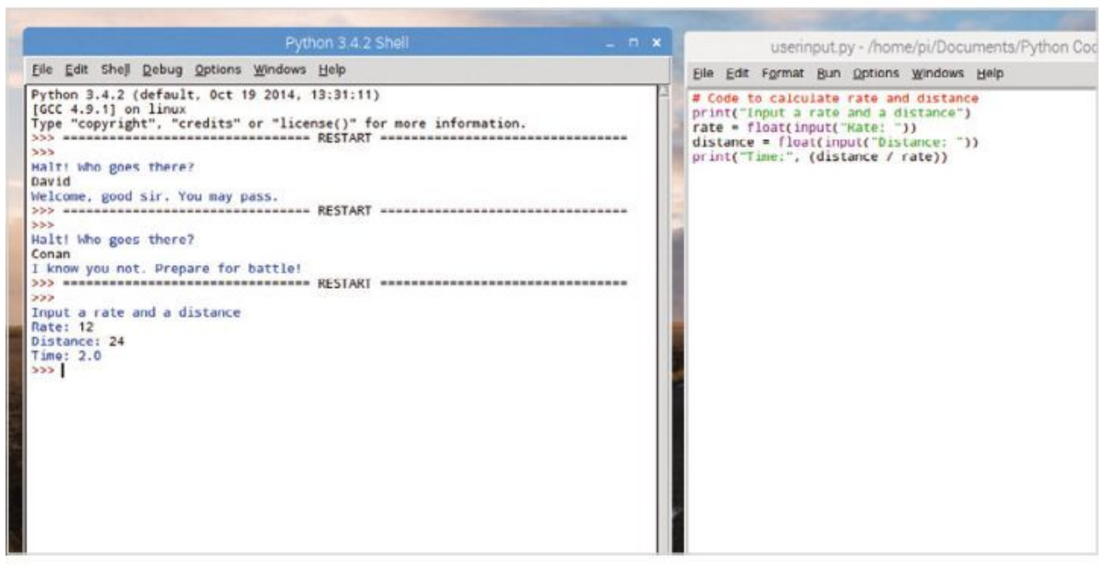
```
# Code to calculate rate and distance
print("Input a rate and a distance")
rate = float(input("Rate: "))
```



STEP 10 To finalise the rate and distance code, we can add:

```
distance = float(input("Distance: "))
print("Time:", (distance / rate))
```

Save and execute the code and enter some numbers. Using the float(input element, we've told Python that anything entered is a floating point number rather than a string.



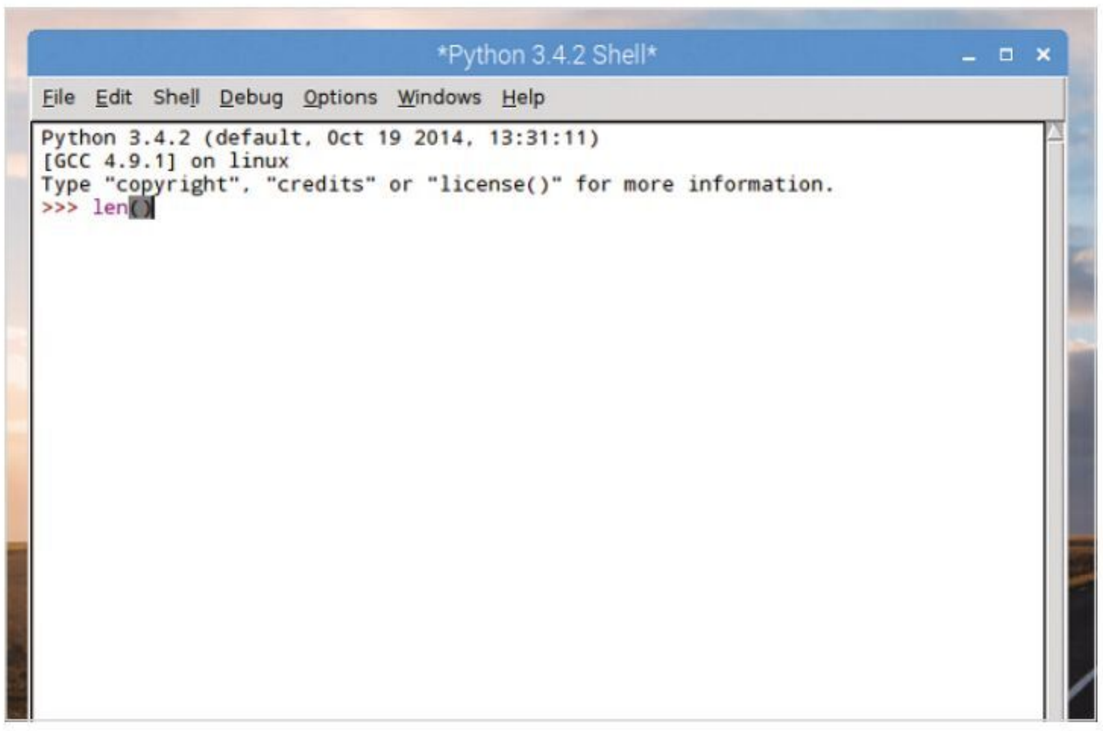
Creating Functions

Now that you've mastered the use of variables and user input, the next step is to tackle functions. You've already used a few functions, such as the print command but Python enables you to define your own functions.

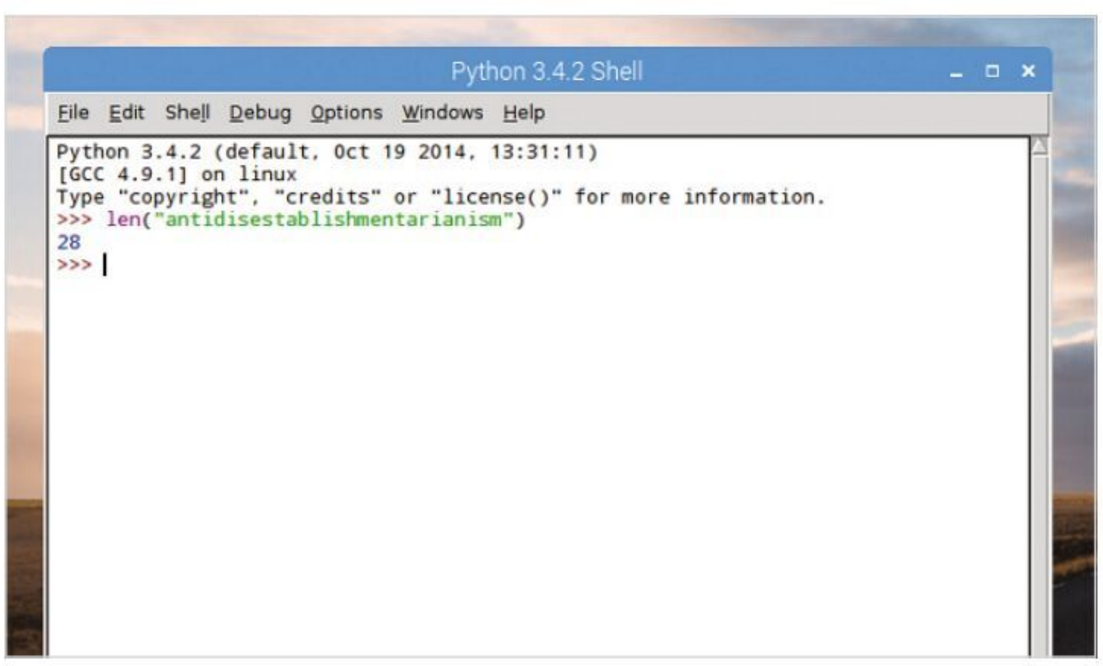
FUNKY FUNCTIONS

A function is a command that you enter into Python to do something. It's a little piece of self-contained code that takes data, works on it and then returns the result.

STEP 1 It's not just data that a function works on. They can do all manner of useful things in Python, such as sort data, change items from one format to another and check the length or type of items. Basically, a function is a short word that's followed by brackets. For example, **len()**, **list()** or **type()**.



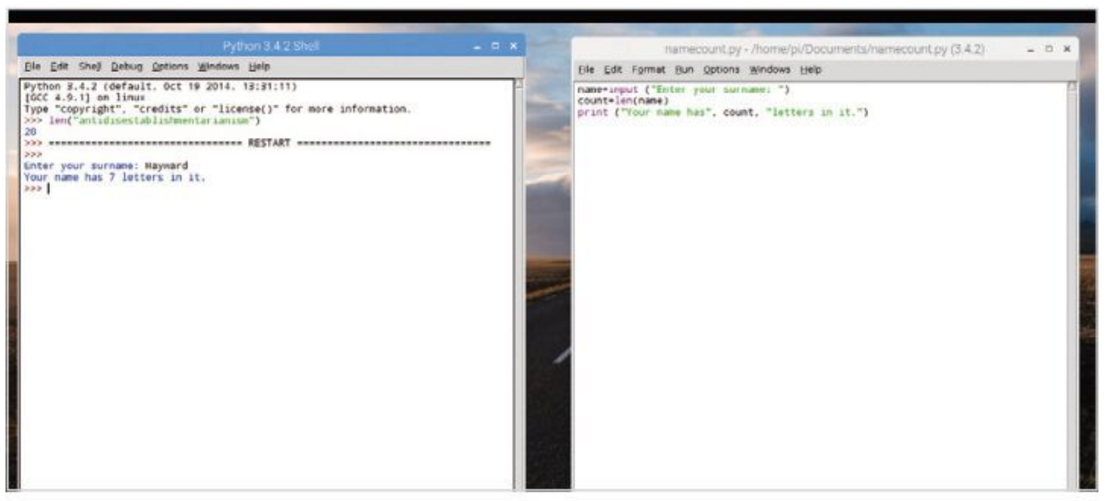
STEP 2 A function takes data, usually a variable, works on it depending on what the function is programmed to do and returns the end value. The data being worked on goes inside the brackets, so if you wanted to know how many letters are in the word antidisestablishmentarianism, then you'd enter: `len("antidisestablishmentarianism")` and the number 28 would return.



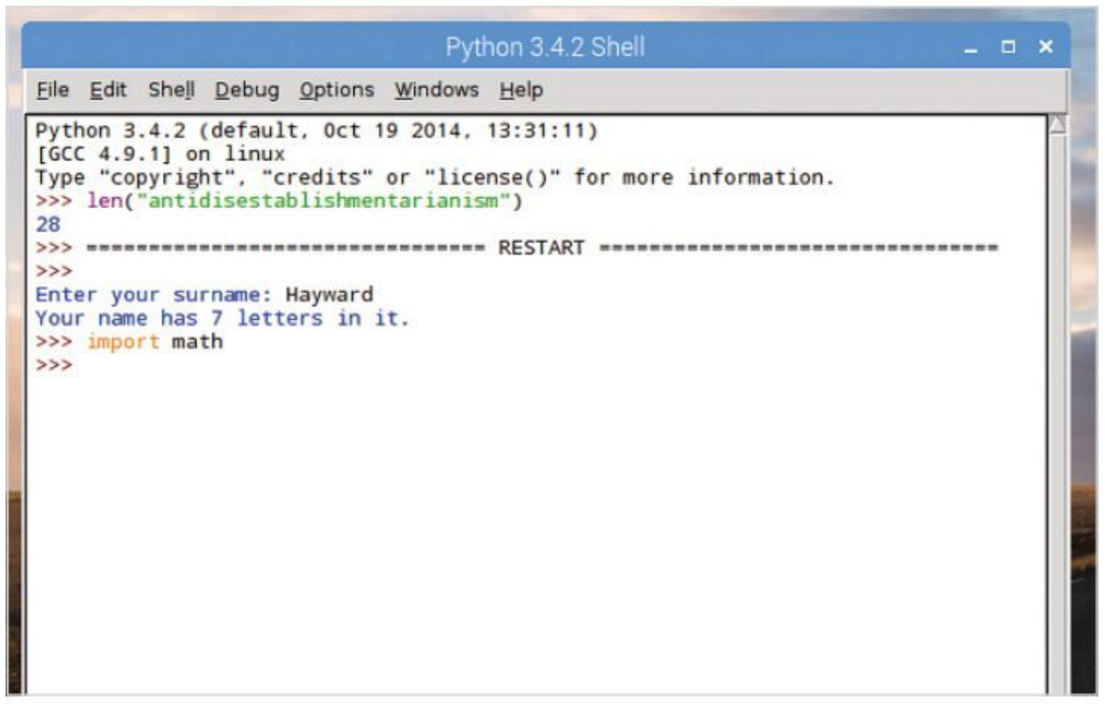
STEP 3 You can pass variables through functions in much the same manner. Let's assume you want the number of letters in a person's surname, you could use the following code (enter the text editor for this example):

```
name=input ("Enter your surname: ")
count=len(name)
print ("Your surname has", count, "letters in
it.")
```

Press F5 and save the code to execute it.



STEP 4 Python has tens of functions built into it, far too many to get into in the limited space available here. However, to view the list of built-in functions available to Python 3, navigate to www.docs.python.org/3/library/functions.html. These are the predefined functions, but since users have created many more, they're not the only ones available.



STEP 5 Additional functions can be added to Python through modules. Python has a vast range of modules available that can cover numerous programming duties. They add functions and can be imported as and when required. For example, to use advanced Mathematics functions enter:

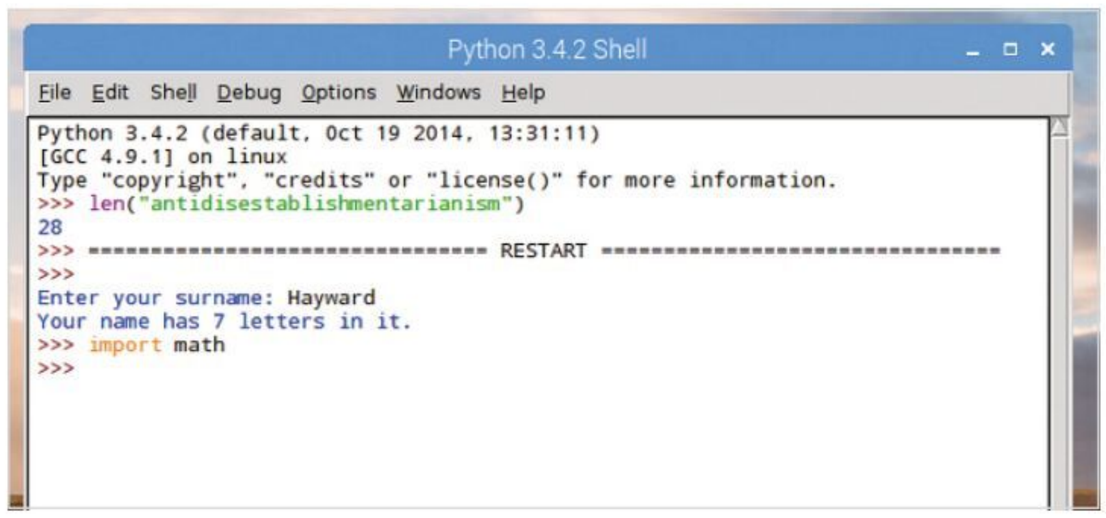
```
import math
```

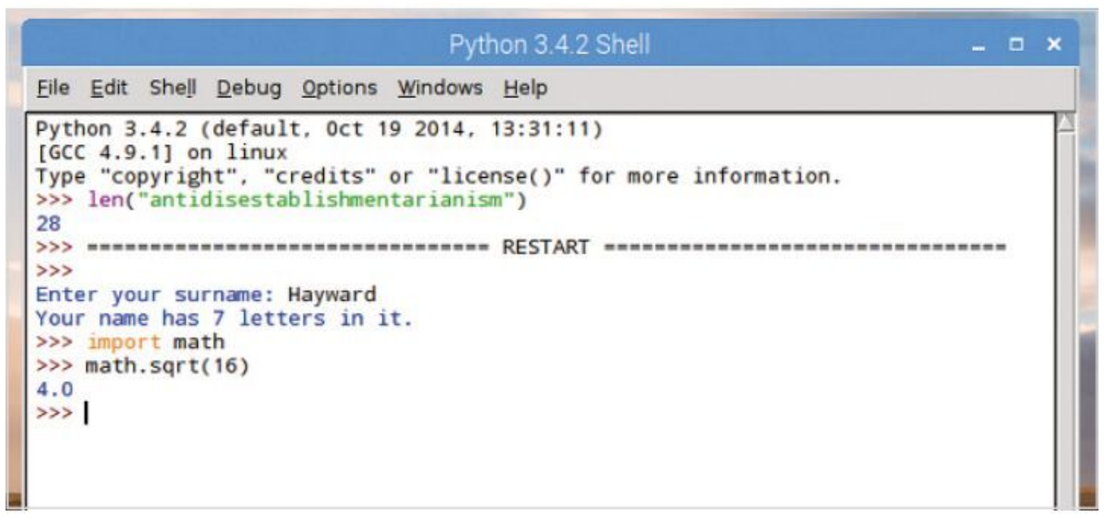
Once entered, you have access to all the Math module functions.

STEP 6 To use a function from a module enter the name of the module followed by a full stop, then the name of the function. For instance, using the Math module, since you've just imported it into Python, you can utilise the square root function. To do so, enter:

```
math.sqrt(16)
```

You can see that the code is presented as module.function(data).





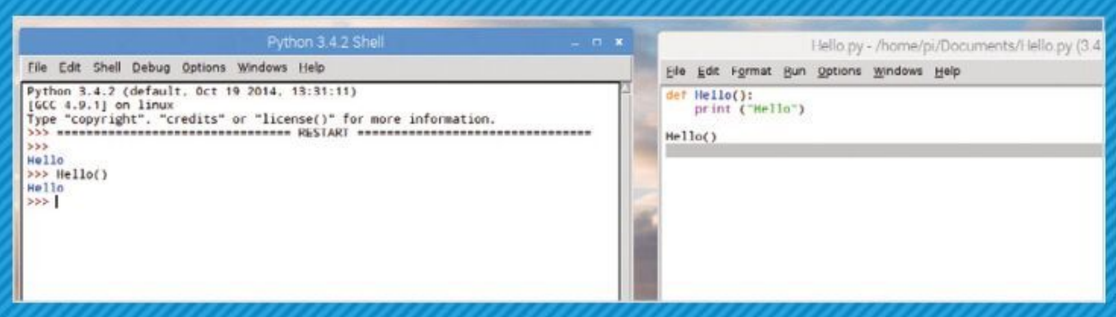
FORGING FUNCTIONS

There are many different functions you can import created by other Python programmers and you will undoubtedly come across some excellent examples in the future; you can also create your own with the def command.

STEP 1 Choose File > New File to enter the editor, let's create a function called Hello, that greets a user. Enter:

```
def Hello():
    print ("Hello")

Hello()
```

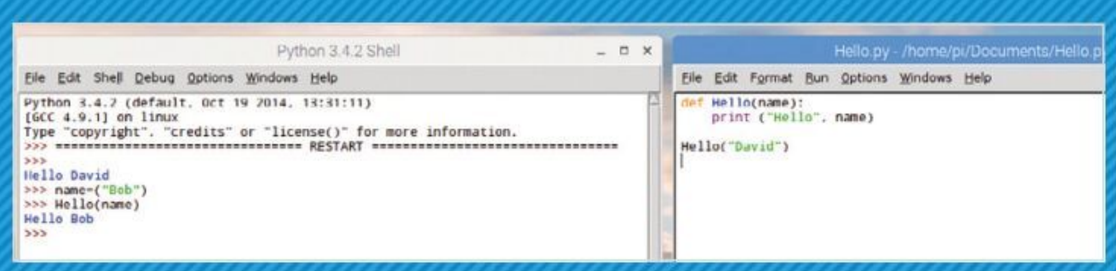
Press F5 to save and run the script. You can see Hello in the Shell, type in Hello() and it returns the new function.



STEP 2 Let's now expand the function to accept a variable, the user's name for example. Edit your script to read:

```
def Hello(name):
    print ("Hello", name)

Hello("David")
```

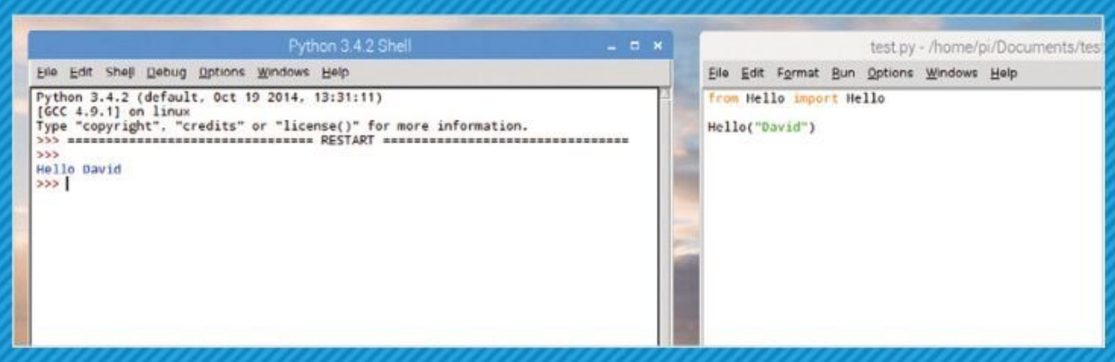
This will now accept the variable name, otherwise it prints Hello David. In the Shell, enter: name=("Bob"), then: Hello(name). Your function can now pass variables through it.



STEP 3 To modify it further, delete the Hello("David") line, the last line in the script and press Ctrl+S to save the new script. Close the Editor and create a new file (File > New File). Enter the following:

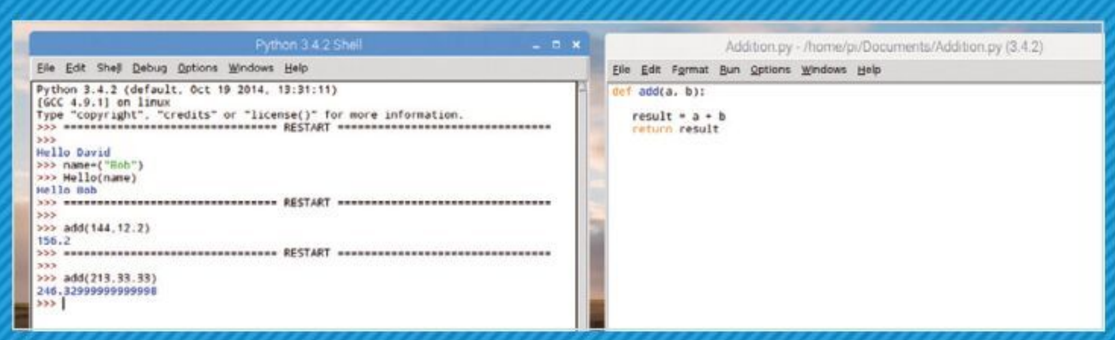
```
from Hello import Hello

Hello("David")
```

Press F5 to save and execute the code.



STEP 4 What you've just done is import the Hello function from the saved Hello.py program and then used it to say hello to David. This is how modules and functions work: you import the module then use the function. Try this one, and modify it for extra credit:

```
def add(a, b):
    result = a + b
    return result
```



Conditions and Loops

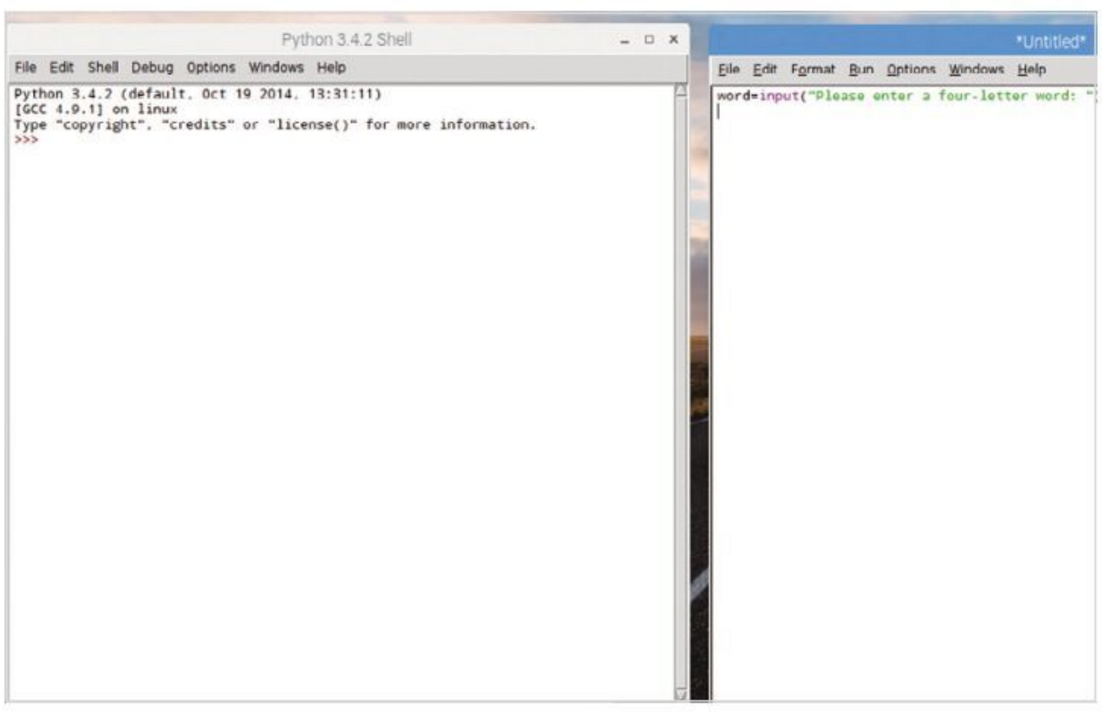
Conditions and loops are what make a program interesting; they can be simple or rather complex. How you use them depends greatly on what the program is trying to achieve; they could be the number of lives left in a game or just displaying a countdown.

TRUE CONDITIONS

Keeping conditions simple to begin with makes learning to program a more enjoyable experience. Let's start then by checking if something is TRUE, then doing something else if it isn't.

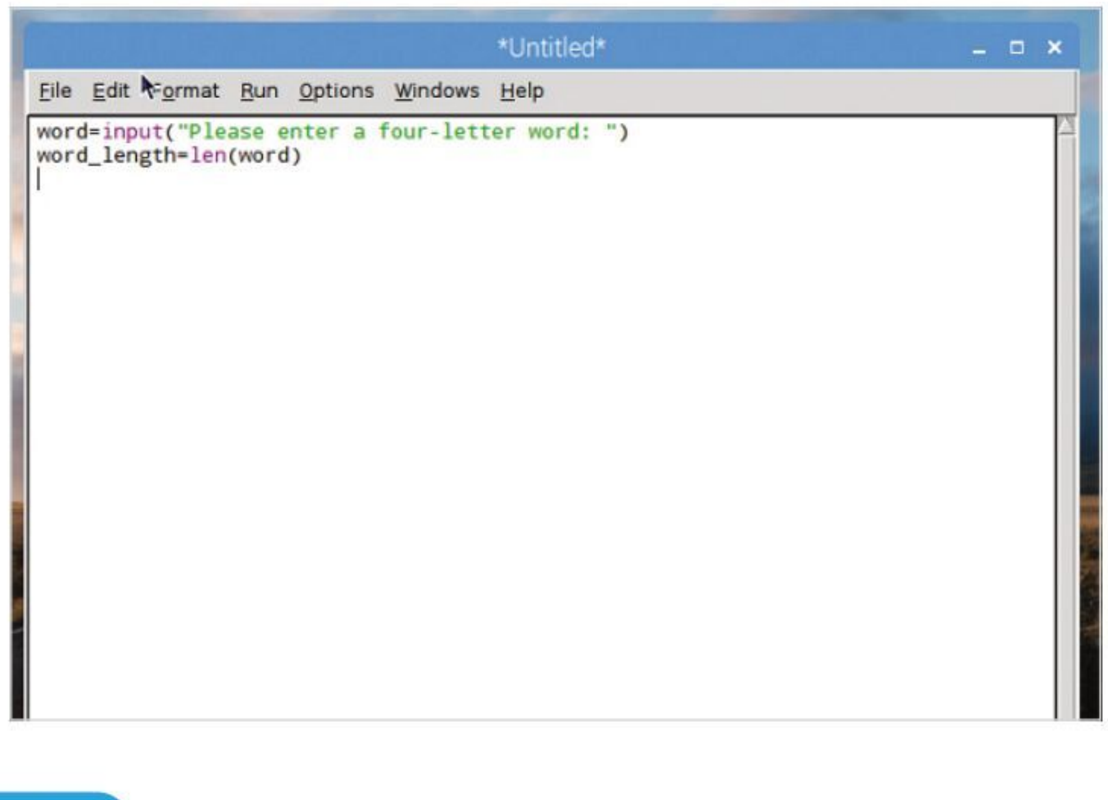
STEP 1 Let's create a new Python program that will ask the user to input a word, then check it to see if it's a four-letter word or not. Start with File > New File, and begin with the input variable:

```
word=input("Please enter a four-letter word: ")
```



STEP 2 Now we can create a new variable, then use the len function and pass the word variable through it to get the total number of letters the user has just entered:

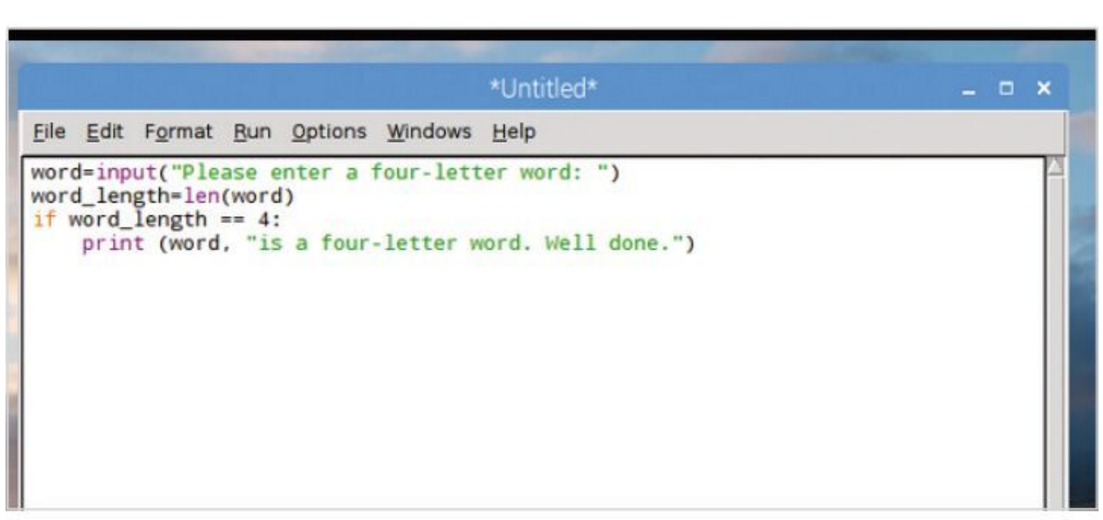
```
word=input("Please enter a four-letter word: ")
word_length=len(word)
```



STEP 3 Now you can use an if statement to check if the word_length variable is equal to four and print a friendly conformation if it applies to the rule:

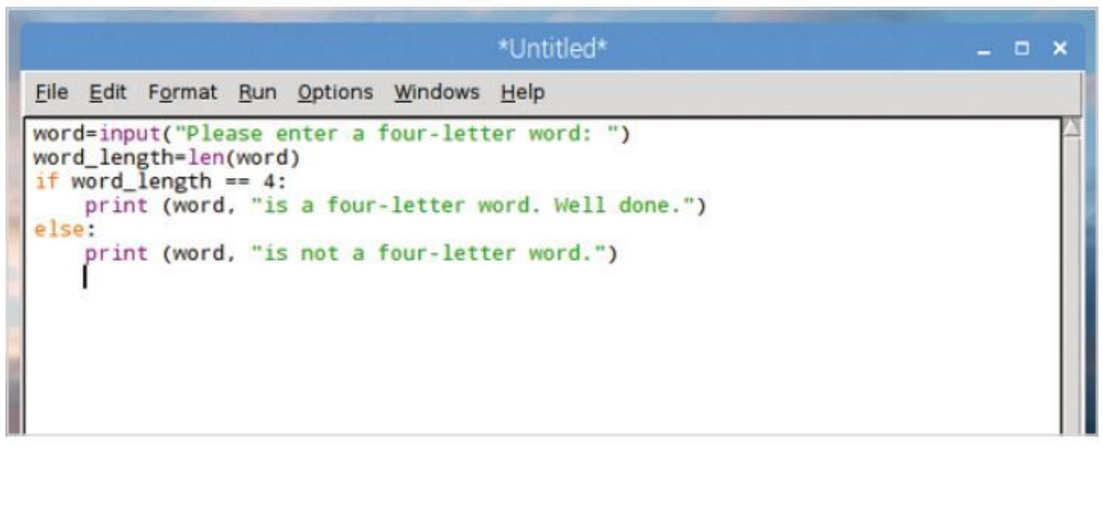
```
word=input("Please enter a four-letter word: ")
word_length=len(word)
if word_length == 4:
   print (word, "is a four-letter word. Well done.")
```

The double equal sign (==) means check if something is equal to something else.

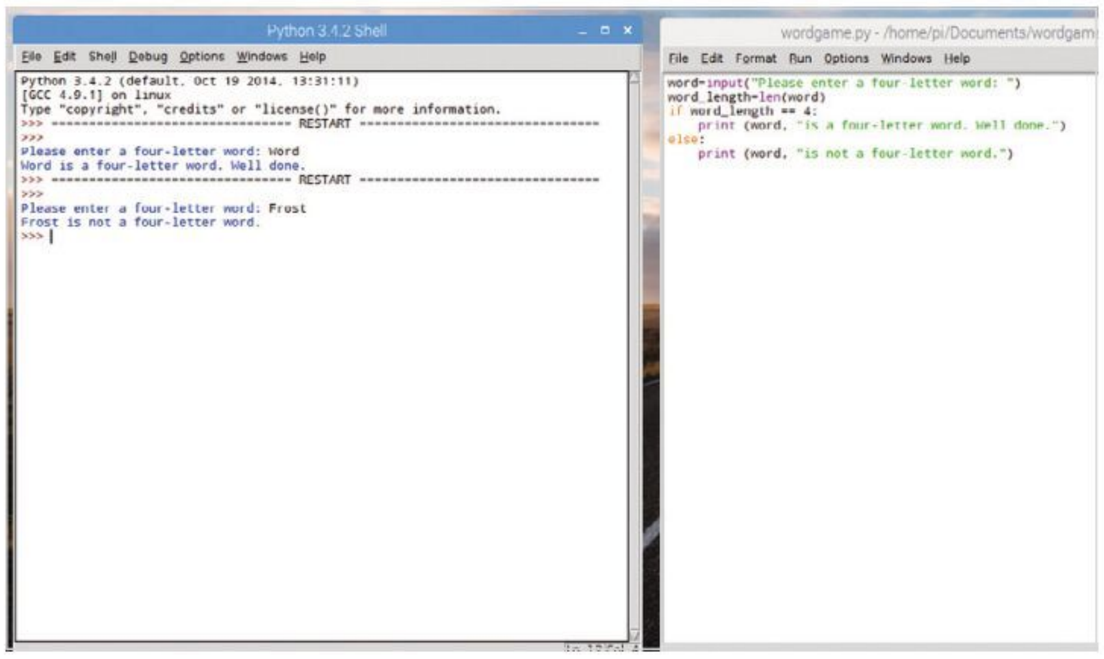


STEP 4 The colon at the end of IF tells Python that if this statement is true do everything after the colon that's indented. Next, move the cursor back to the beginning of the Editor:

```
word=input("Please enter a four-letter word: ")
word_length=len(word)
if word_length == 4:
   print (word, "is a four-letter word. Well
done.")
else:
   print (word, "is not a four-letter word.")
```

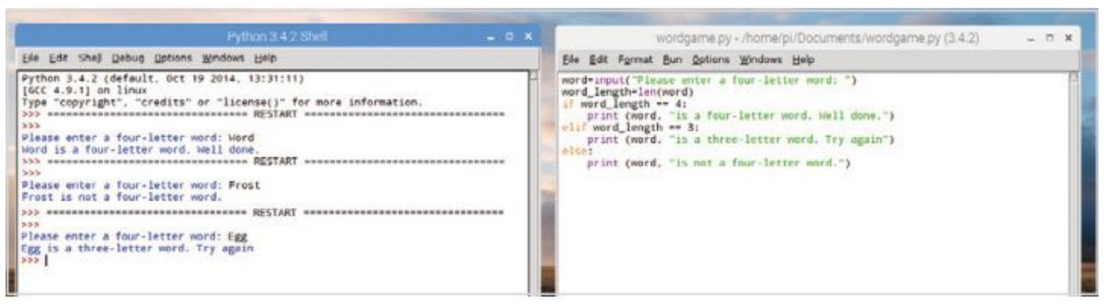


STEP 5 Press F5 and save the code to execute it. Enter a four-letter word in the Shell to begin with, you should have the returned message that it's the word is four letters. Now press F5 again and rerun the program but this time enter a five-letter word. The Shell will display that it's not a four-letter word.



STEP 6 Now expand the code to include another conditions. Eventually, it could become quite complex. We've added a condition for three-letter words:

```
word=input("Please enter a four-letter word: ")
word_length=len(word)
if word_length == 4:
   print (word, "is a four-letter word. Well
done.")
elif word_length == 3:
   print (word, "is a three-letter word. Try again.")
else:
   print (word, "is not a four-letter word.")
```

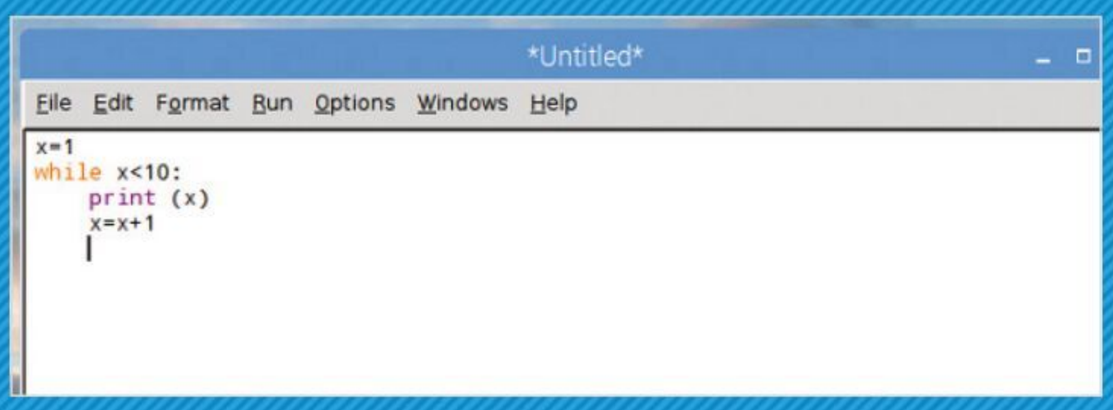


LOOPS

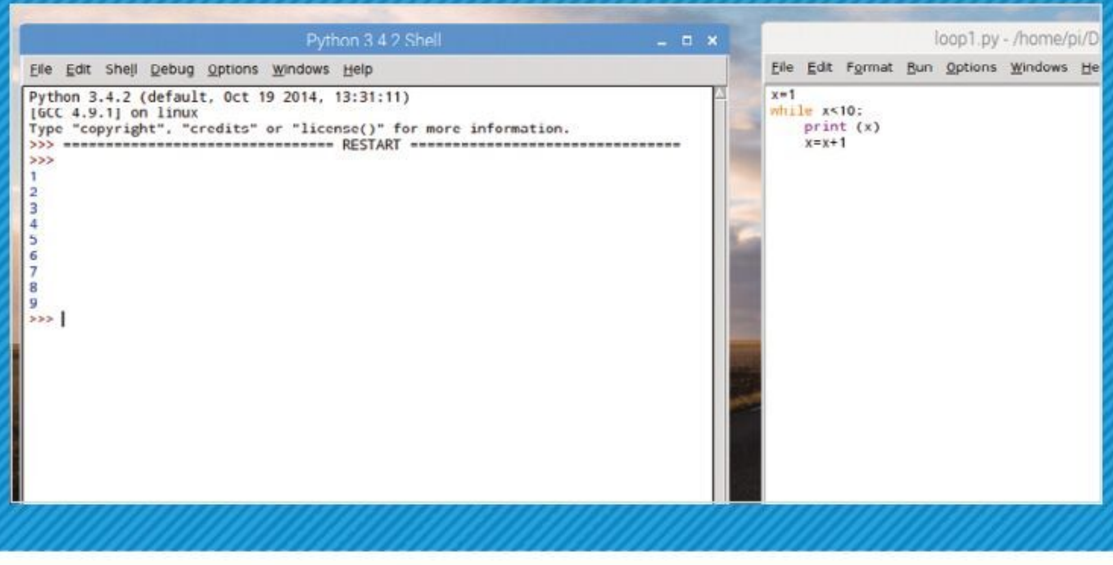
A loop looks quite similar to a condition but they are somewhat different in their operation. A loop will run through the same block of code a number of times, usually with the support of a condition.

STEP 1 Let's start with a simple While statement. Like IF, this will check to see if something is TRUE, then run the indented code:

```
x = 1
while x < 10:
   print (x)
   x = x + 1
```

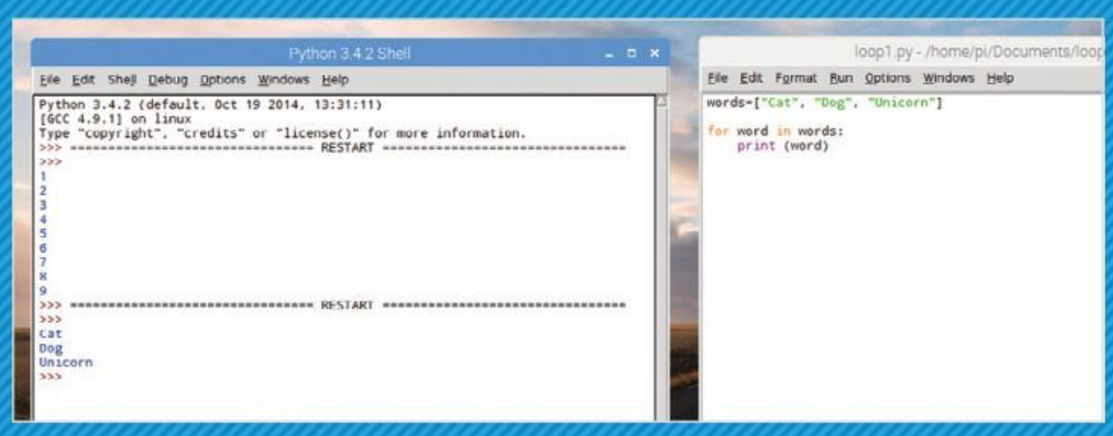


STEP 2 The difference between if and while is when while gets to the end of the indented code, it goes back and checks the statement is still true. In our example x is less than 10. With each loop it prints the current value of x, then adds one to that value. When x does eventually equal 10 it stops.



STEP 3 The For loop is another example. For is used to loop over a range of data, usually a list stored as variables inside square brackets. For example:

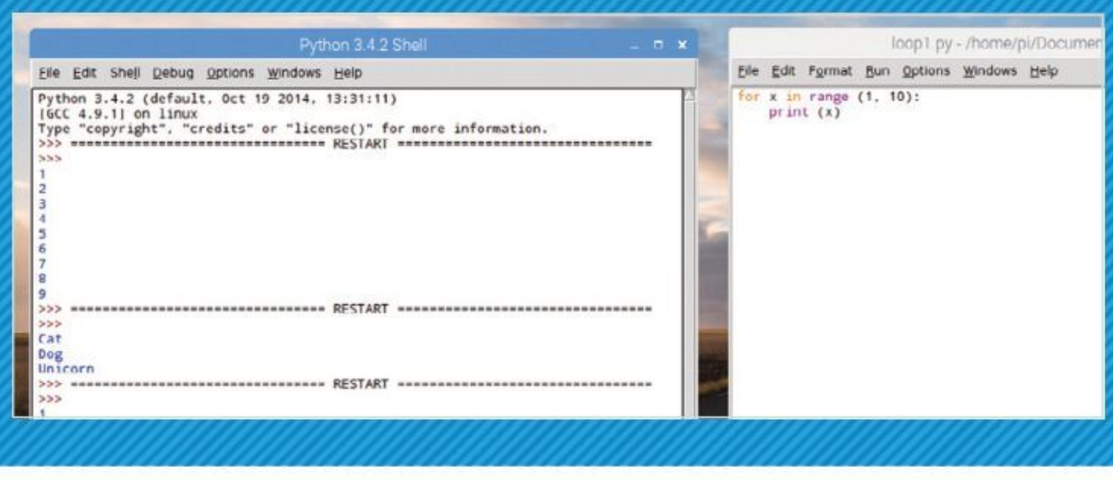
```
words=["Cat", "Dog", "Unicorn"]

for word in words:
   print (word)
```



STEP 4 The For loop can also be used in the countdown example by using the range function:

```
for x in range (1, 10):
  print (x)
```

The x=x+1 part isn't needed here because the range function creates a list between the first and last numbers used.



Python Modules

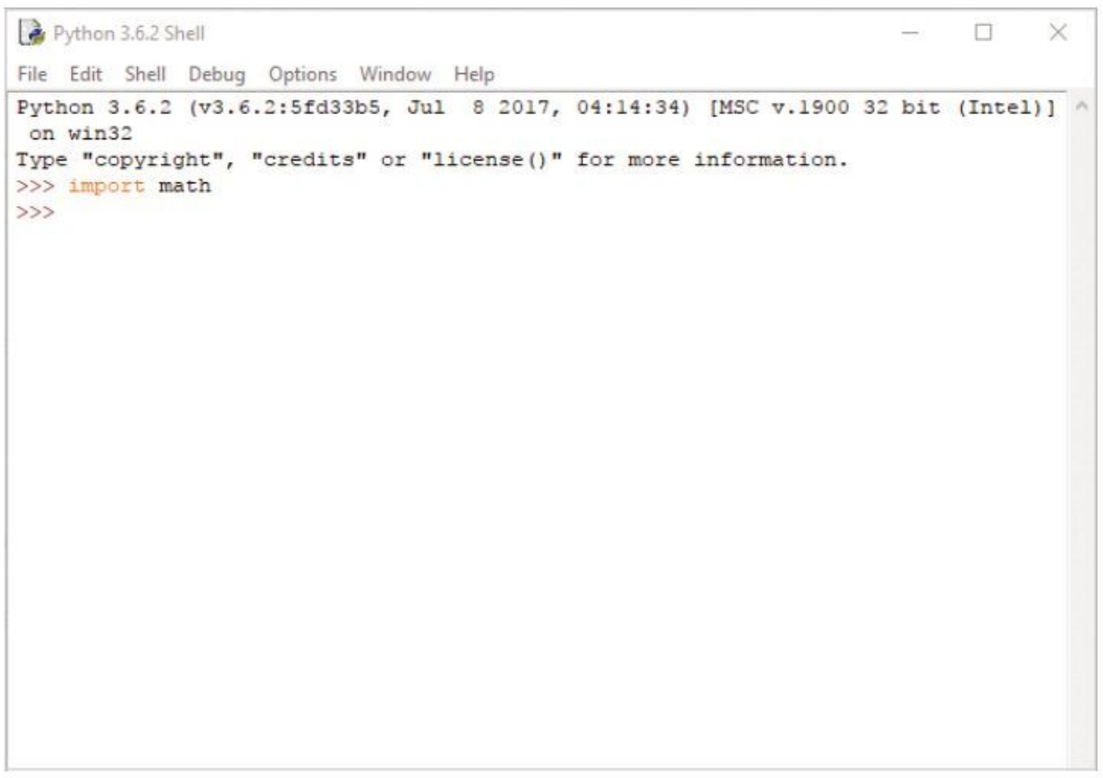
We've mentioned modules previously, (the Math module) but as modules are such a large part of getting the most from Python, it's worth dedicating a little more time to them. In this instance we're using the Windows version of Python 3.

MASTERING MODULES

Think of modules as an extension that's imported into your Python code to enhance and extend its capabilities. There are countless modules available and as we've seen, you can even make your own.

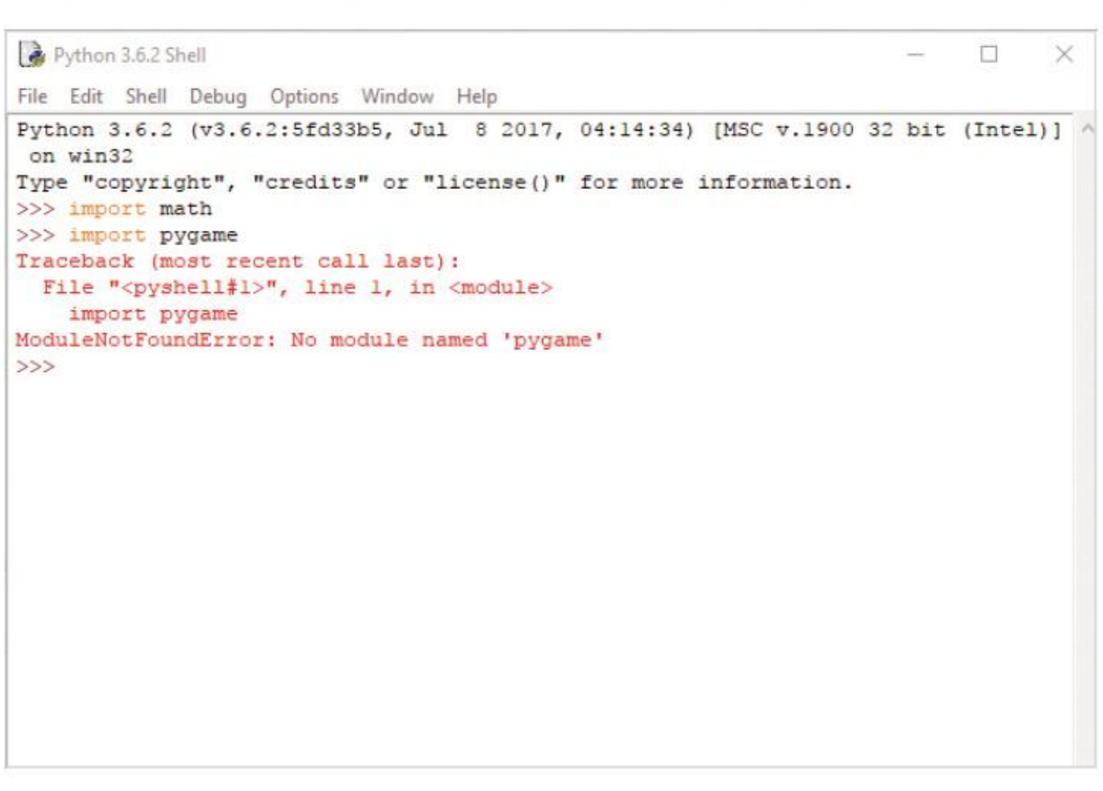
STEP 1

Although good, the built-in functions within Python are limited. The use of modules, however, allows us to make more sophisticated programs. As you are aware, modules are Python scripts that are imported, such as import math.



STEP 2

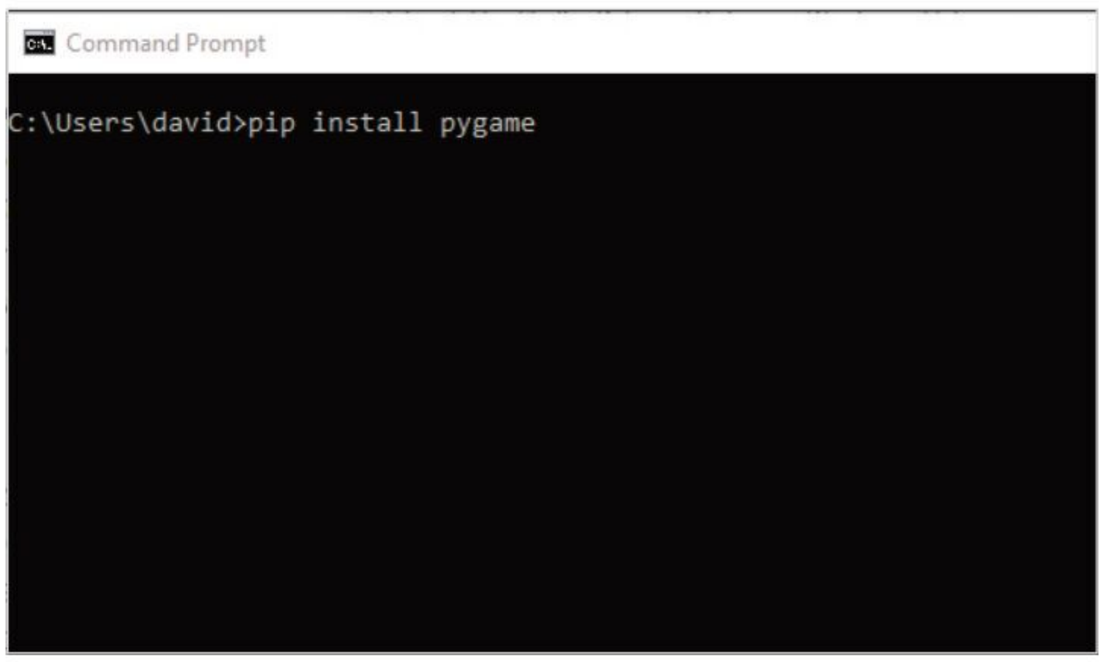
Some modules, especially on the Raspberry Pi, are included by default, the Math module being a prime example. Sadly, other modules aren't always available. A good example on non-Pi platforms is the Pygame module, which contains many functions to help create games. Try: **import pygame**.



STEP 3

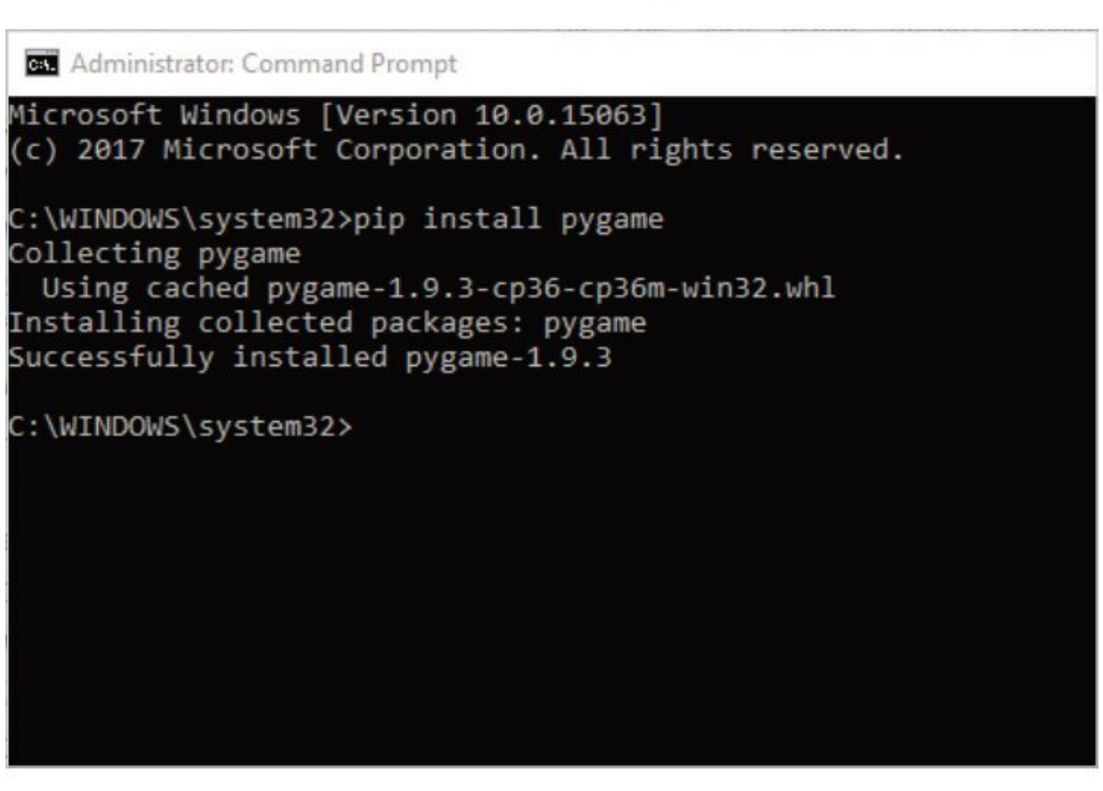
The result is an error in the IDLE Shell, as the Pygame module isn't recognised or installed in Python. To install a module we can use PIP (Pip Installs Packages). Close down the IDLE Shell and drop into a command prompt or Terminal session. At an elevated admin command prompt, enter:

```
pip install pygame
```

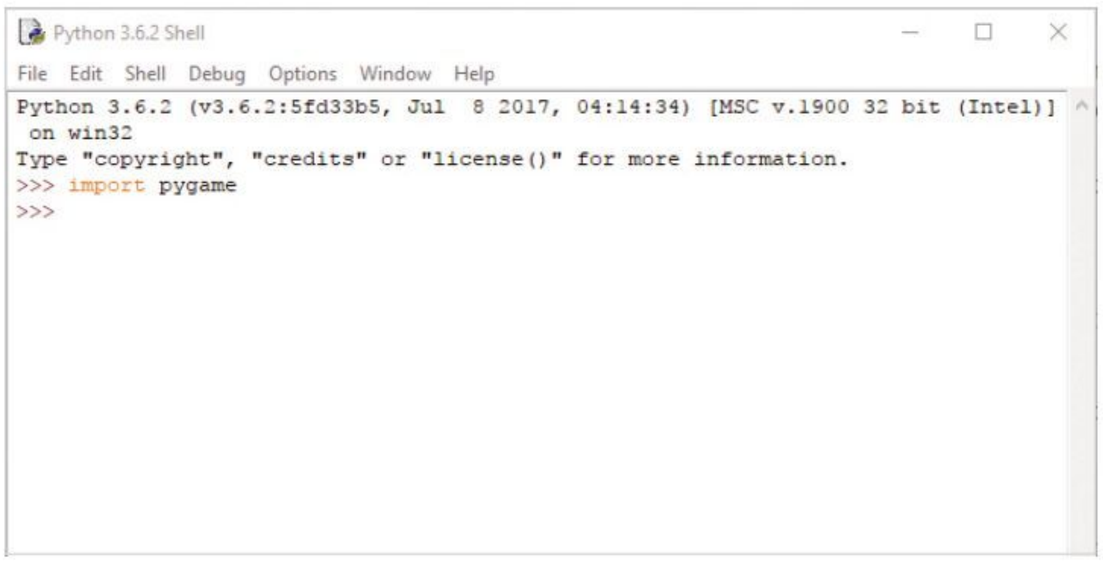


STEP 4

The PIP installation requires an elevated status due it installing components at different locations. Windows users can search for CMD via the Start button and right-click the result then click Run as Administrator. Linux and Mac users can use the Sudo command, with sudo pip install package.



STEP 5 Close the command prompt or Terminal and relaunch the IDLE Shell. When you now enter: `import pygame`, the module will be imported into the code without any problems. You'll find that most code downloaded or copied from the Internet will contain a module, mainstream of unique, these are usually the source of errors in execution due to them being missing.



STEP 6 The modules contain the extra code needed to achieve a certain result within your own code, as we've previously experimented with. For example:

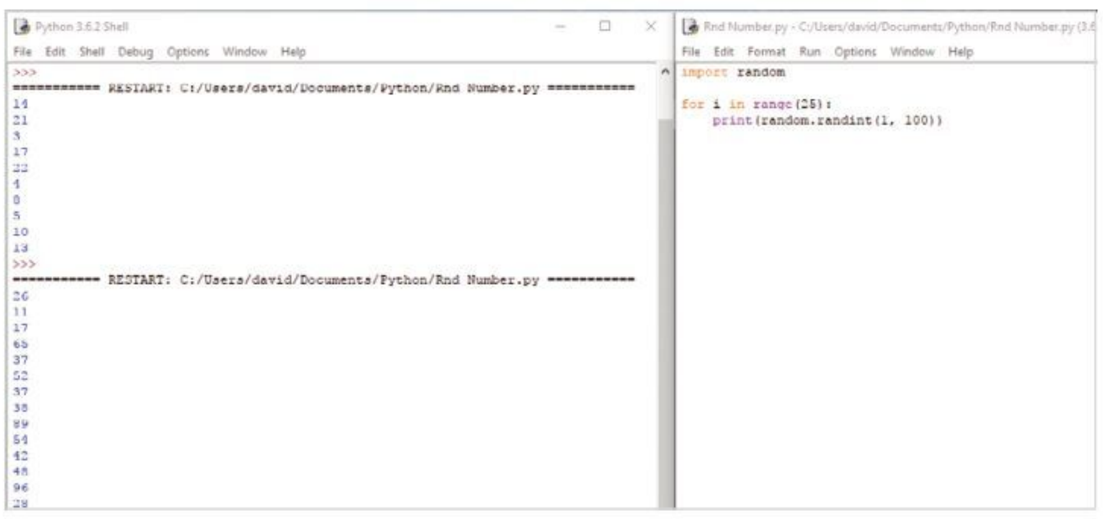
```
import random
```

Brings in the code from the Random Number Generator module. You can then use this module to create something like:

```
for i in range(10):
  print(random.randint(1, 25))
```

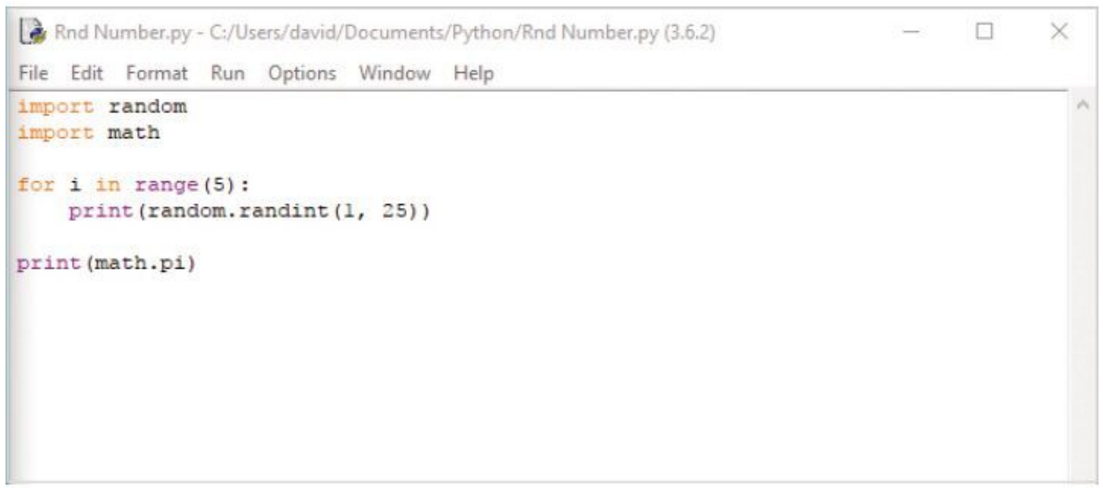
STEP 7 This code, when saved and executed, will display ten random numbers from 1 to 25. You can play around with the code to display more or less, and from a great or lesser range. For example:

```
import random

for i in range(25):
  print(random.randint(1, 100))
```



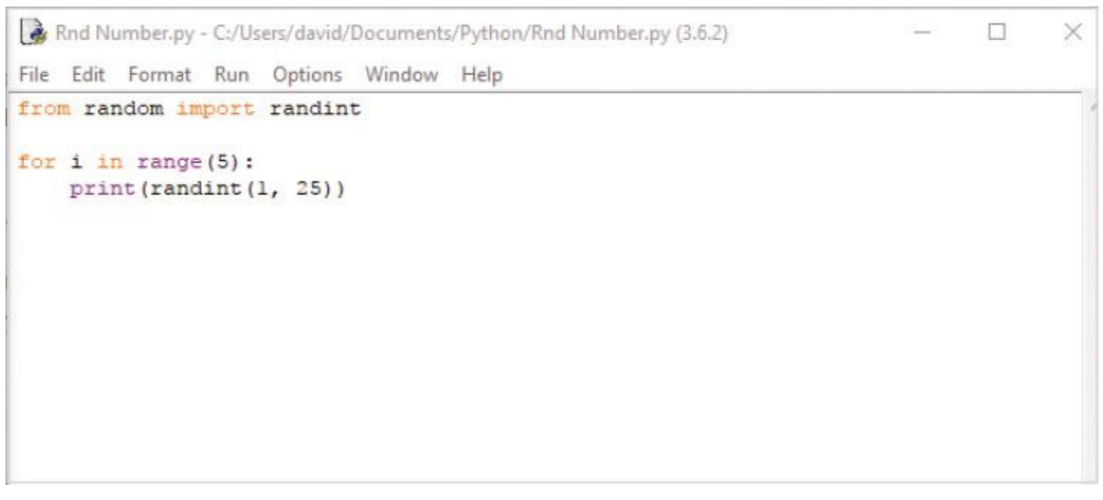
STEP 8 Multiple modules can be imported within your code. To extend our example, use:

```
import random
import math

for I in range(5):
  print(random.randint(1, 25))

print(math.pi)
```



STEP 9 The result is a string of random numbers followed by the value of Pi as pulled from the Math module using the print(math.pi) function. You can also pull in certain functions from a module by using the from and import commands, such as:

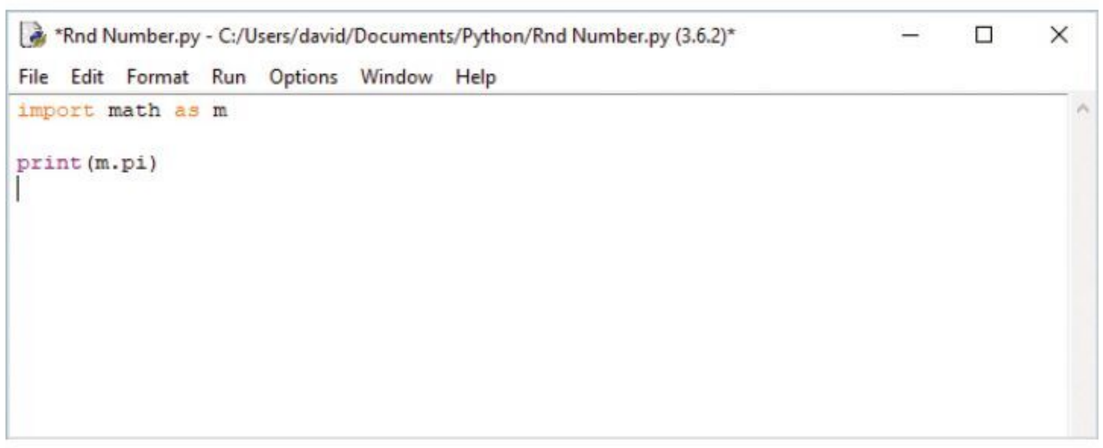
```
from random import randint

for i in range(5):
  print(randint(1, 25))
```



STEP 10 This helps create a more streamlined approach to programming. You can also use Import module*, which will import everything defined within the named module. However, it's often regarded as a waste of resources but it works nonetheless. Finally, modules can be imported as aliases:

```
import math as m

print(m.pi)
```

Of course, adding comments helps to tell others what's going on.



Admiral Grace Hopper, debugging computers since the ‘40s

Did You Know...

Debugging

DID YOU KNOW... that the word debugging in computing terms comes from Admiral Grace Hopper, who back in the '40s was working on a monolithic Harvard Mark II electromechanical computer. According to legend Hopper found a moth stuck in a relay, thus stopping the system from working. Removal of the moth was hence called debugging.

C++ on Linux

"The most important single aspect of software development is to be clear about what you are trying to build."

– *Bjarne Stroustrup (Developer and creator of C++)*

C++ is an excellent, high-level programming language that's used in a multitude of technologies. Everything from your favourite mobile app, console and PC game to entire operating systems are developed with C++ at the core, together with a collection of software development kits and custom libraries.

C++ is the driving force behind most of what you use on a daily basis, which makes it a complex and extraordinarily powerful language to get to grips with. It's the code behind Linux itself, as well as most of the behind the scenes drivers, library files and control elements of most, if not all, operating systems.

This section shows you how to start coding in C++ on Linux using a Raspberry Pi. From there, you can begin to create great things.

120	Why C++?
122	Your First C++ Program
124	Structure of a C++ Program
126	Compile and Execute
128	Did You Know…Virus!
132	Variables
136	Strings
138	C++ Maths
140	User Interaction
142	Did You Know…The Hobbit
144	Common Coding Mistakes

Why C++?

C++ is one of the most popular programming languages available today. Originally called C with Classes, the language was renamed C++ in 1983. It's an extension of the original C language and is a general purpose object-oriented (OOP) environment.

C EVERYTHING

Due to how complex the language can be, and its power and performance, C++ is often used to develop games, programs, device drivers and even entire operating systems.

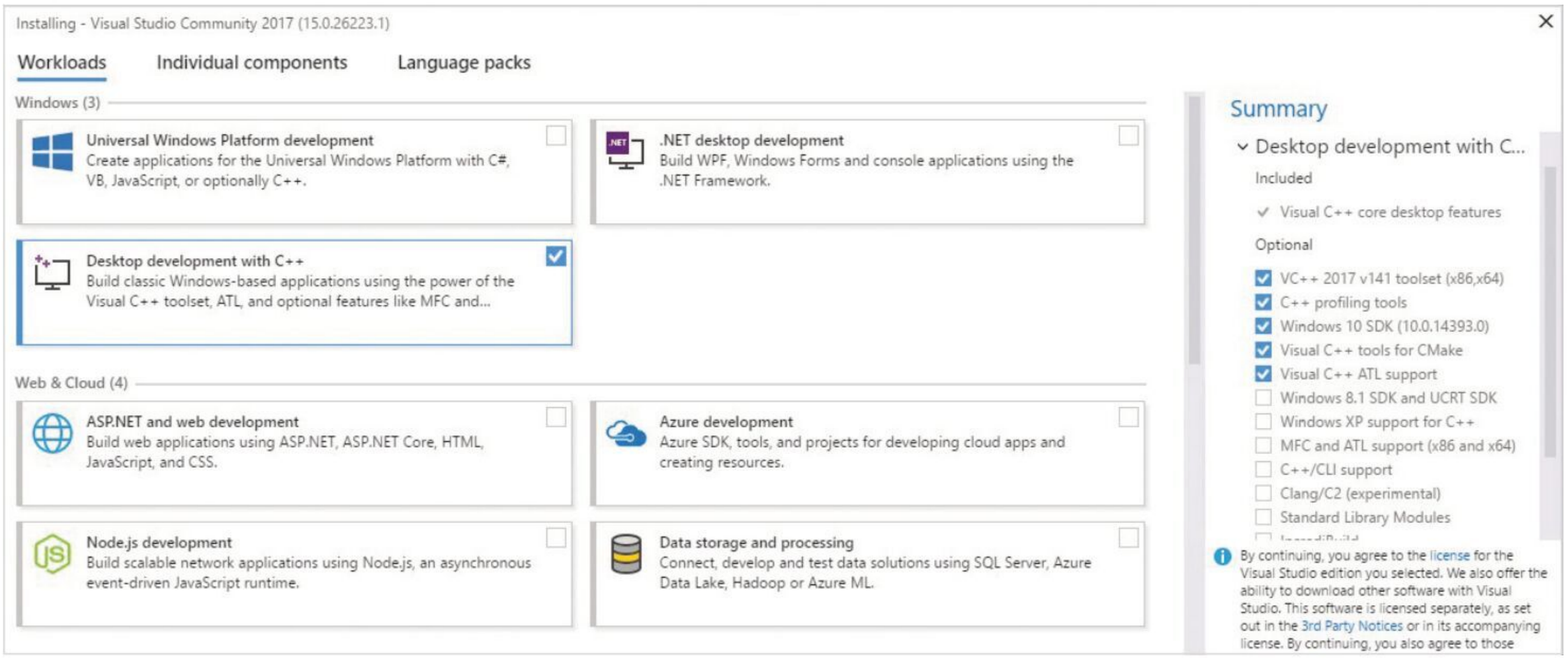
Dating back to 1979, the start of the golden era of home computing, C++, or rather C with Classes, was the brainchild of Danish computer scientist Bjarne Stroustrup while working on his PhD thesis. Stroustrup's plan was to further the original C language, which was widely used since the early seventies.

C++ proved to be popular among the developers of the '80s, since it was a much easier environment to get to grips with and more importantly, it was 99% compatible with the original C language. This meant that it could be used beyond the mainstream computing labs and by regular people who didn't have access to the mainframes and large computing data centres.

C++'s impact in the digital world is immense. Many of the programs, applications, games and even operating systems are coded using C++. For example, all of Adobe's major applications, such as Photoshop, InDesign and so on, are developed in C++. You will find that the browser you surf the Internet with is written in C++, as well as Windows 10, Microsoft Office and the backbone to Google's search engine. Apple's macOS is written largely in C++ (with some

C++ code is much faster than that of Python.

```
#include<iostream>
using namespace std;
void main()
{char ch;
 cout<<"Enter a charater to check it is vowel or not";
 cin>>ch;
     switch(ch)
             {
              case'a': case'A':
              cout<<ch<<" is a Vowel";
              break;
                  case 'e': case'E':
              cout<<ch<<" is a Vowel";
              break;
                  case 'i': case'I':
                  cout<<ch<<" is a Vowel";
                  break;
                      case 'o': case'O':
                      cout<<ch<<" is a Vowel";
                      break;
                          case 'u': case'U':
                          cout<<ch<<" is a Vowel";
                          break;
             default:
```

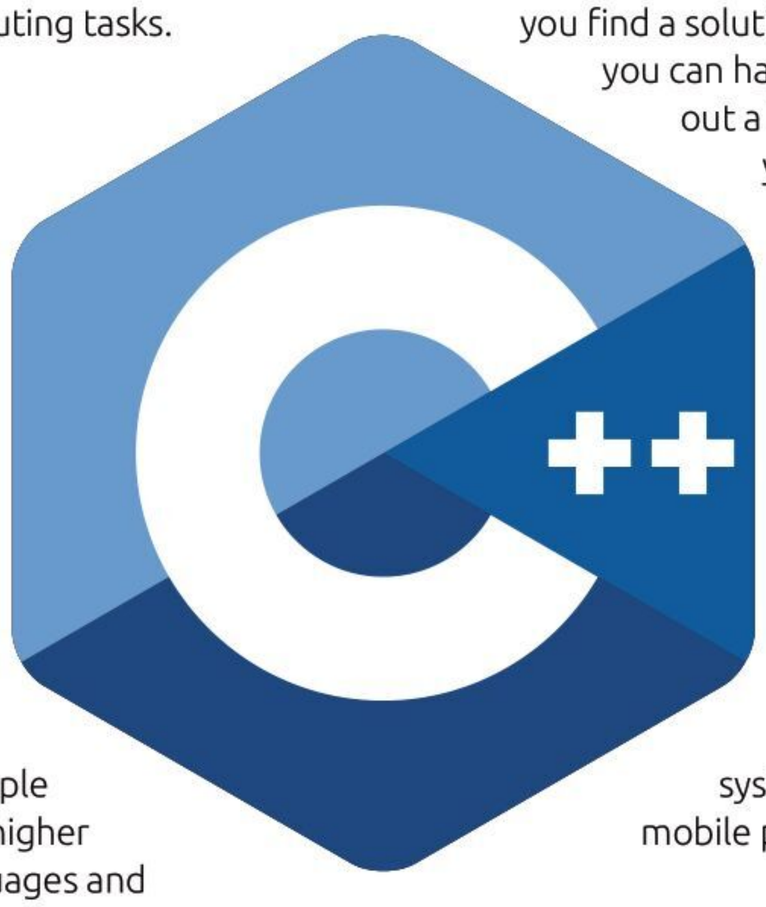


Microsoft's Visual Studio is a great, free environment to learn C++ in.

other languages mixed in depending on the function) and the likes of NASA, SpaceX and even CERN use C++ for various applications, programs, controls and umpteen other computing tasks.

C++ is also extremely efficient and performs well across the board as well as being an easier addition to the core C language. This higher level of performance over other languages, such as Python, BASIC and such, makes it an ideal development environment for modern computing, hence the aforementioned companies using it so widely.

While Python is a great programming language to learn, C++ puts the developer in a much wider world of coding. By mastering C++, you can find yourself developing code for the likes of Microsoft, Apple and so on. Generally, C++ developers enjoy a higher salary than programmers of some other languages and due to its versatility, the C++ programmer can move between jobs and companies without the need to relearn anything specific. However, Python is an easier language to begin with. If you're completely new to programming then we would recommend you begin with Python and spend some time getting to grips with programming structure and the many ways and means in which you find a solution to a problem through programming. Once you can happily power up your computer and whip out a Python program with one hand tied behind your back, then move on to C++. Of course, there's nothing stopping you from jumping straight into C++; if you feel up to the task, go for it.



Getting to use C++ is as easy as Python, all you need is the right set of tools in which to communicate with the computer in C++ and you can start your journey. A C++ IDE is free of charge, even the immensely powerful Visual Studio from Microsoft is freely available to download and use. You can get into C++ from any operating system, be it macOS, Linux, Windows or even mobile platforms.

Just like Python, to answer the question of Why C++ is the answer is because it's fast, efficient and developed by most of the applications you regularly use. It's cutting edge and a fantastic language to master.

Indeed, the operating system you're using is written in C++.

Your First C++ Program

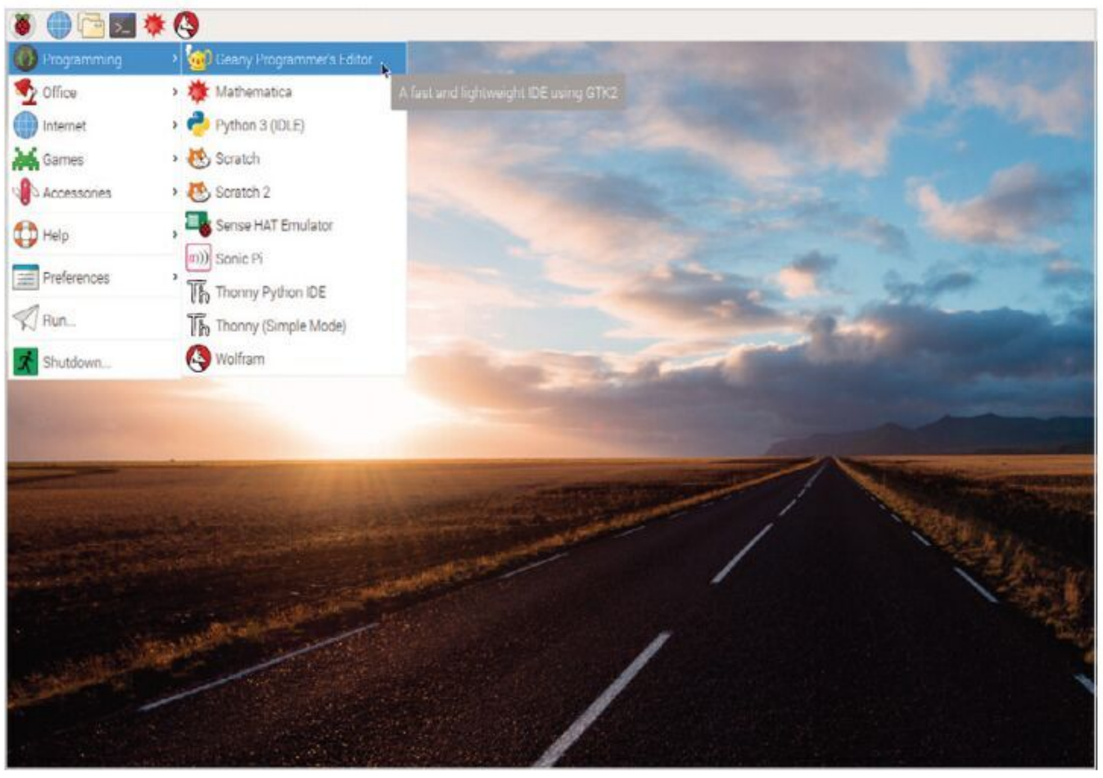
The Raspberry Pi makes for a great coding base on which to learn C++. You won't need any complicated, third-party IDEs or to install any extra components. Everything you need to get started is already built into the Pi.

HELLO, WORLD!

It's traditional in programming for the first code to be entered to output the words 'Hello, World!' to the screen. Interestingly, this dates back to 1968 using a language called BCPL.

STEP 1

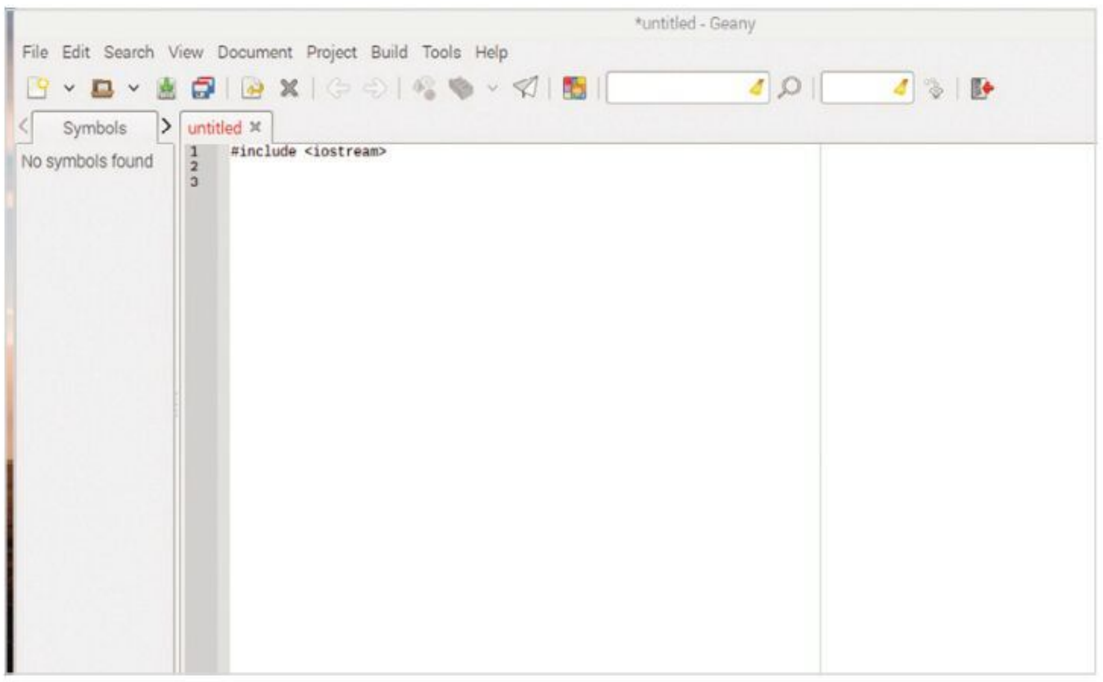
You can use the included Geany Programmer's Editor to write, compile and execute your C++ code on the Raspberry Pi. You can find Geany as the first entry in the Programming section from the main Raspberry Pi menu. Click the Geany icon to open and start coding.



STEP 2

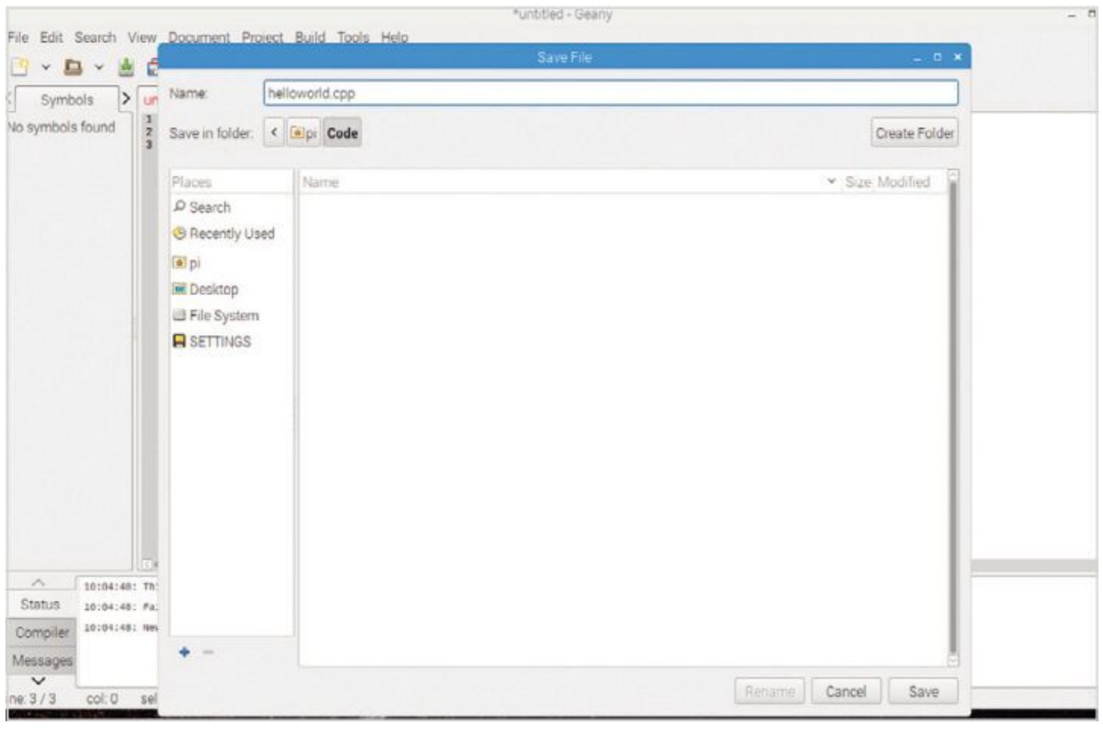
The Geany editor is quite powerful while still being easy to use. All the code you enter shows in the main window and is numbered to help you keep track of your code. For now, in line 1, enter:

```
#include <iostream>
```



STEP 3

It doesn't look like much at the moment but you can get into what the C++ header commands mean in due course. Geany uses syntax highlighting, meaning it colour-codes the code depending on the language. To activate this, save the code by clicking on File > Save As. Now name the code helloworld.cpp and click the Save button.

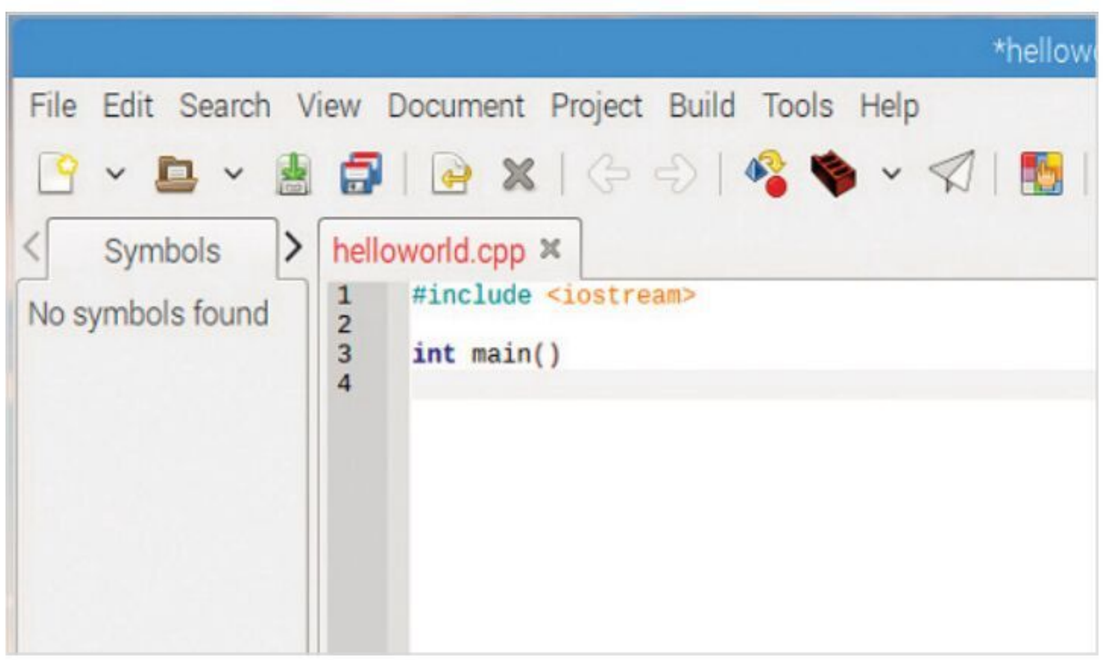


STEP 4

The colour coding helps a lot when your coding. Now press Enter and start a new command on line 3. Enter the following:

```
int main()
```

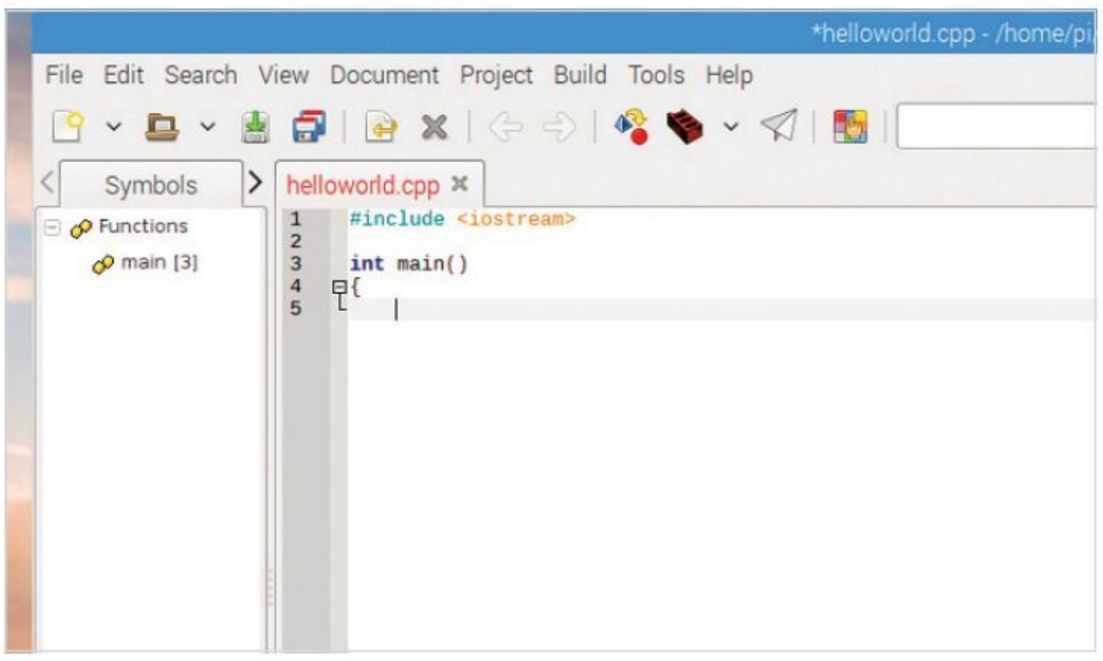
Note: there's no space between the brackets.



STEP 5 On the next line, below int main(), enter a curly bracket:

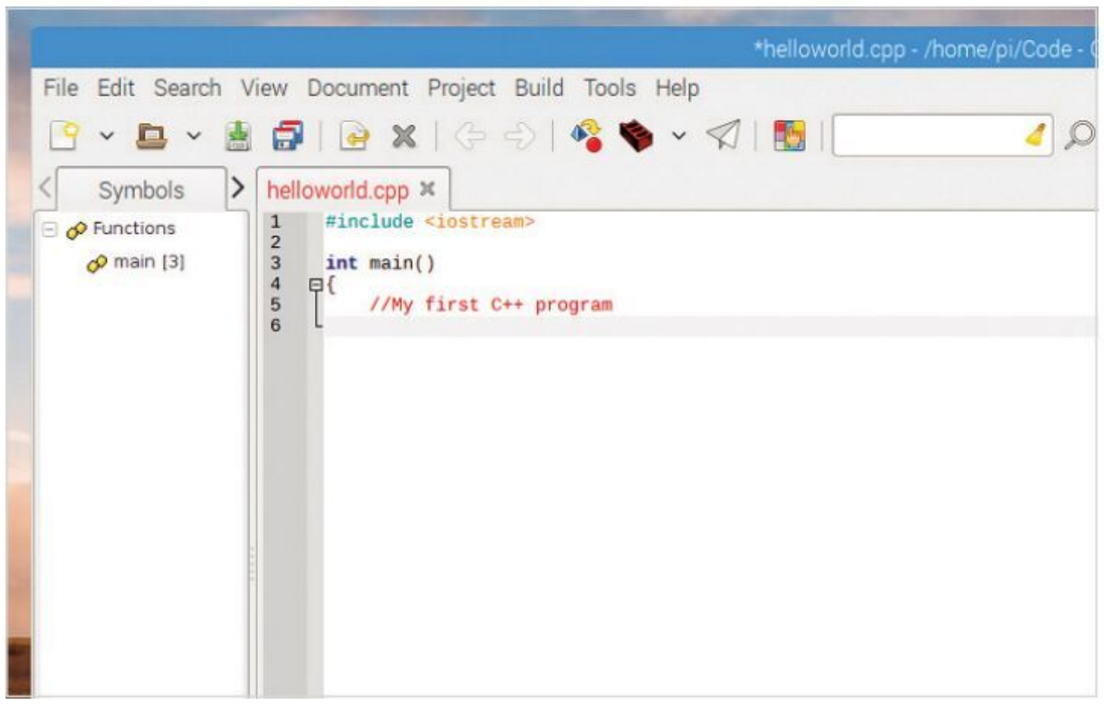
```
{
```

This can be done by pressing Shift and the key to the right of P on an English UK keyboard layout.



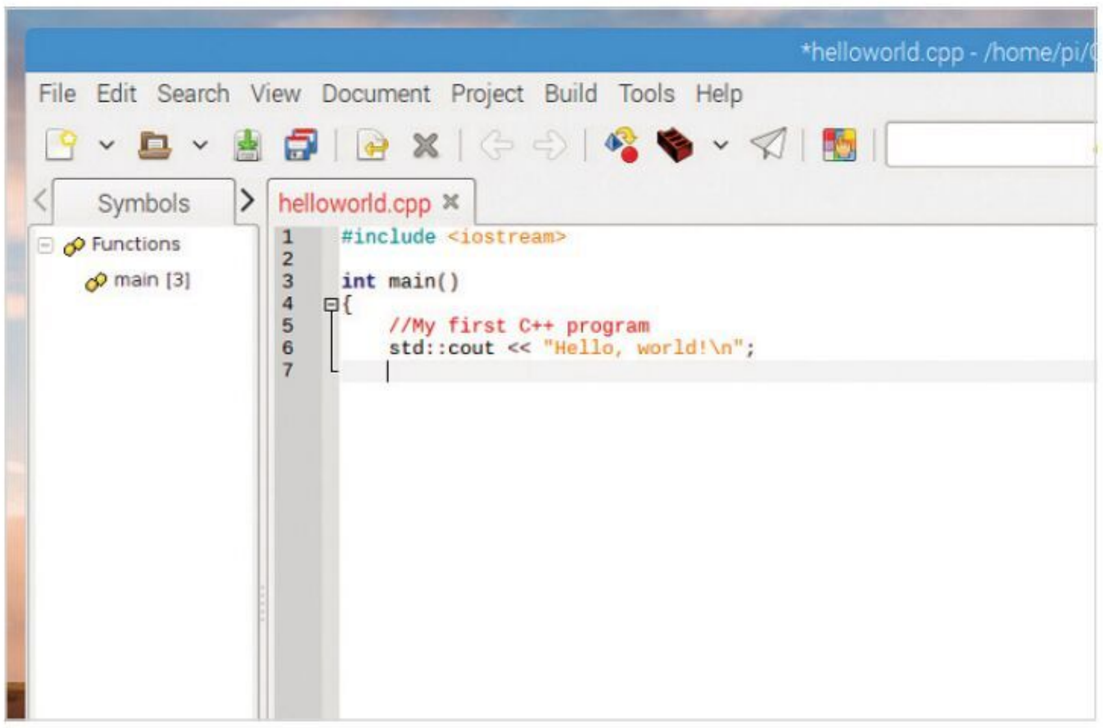
STEP 6 Notice that Geany has automatically created a slight indent. This is due to the structure of C++ and it's where the meat of the code is entered. Now enter:

```
//My first C++ program
```



STEP 7 Note again the colour coding change. Press Return at the end of the previous step's line and then enter:

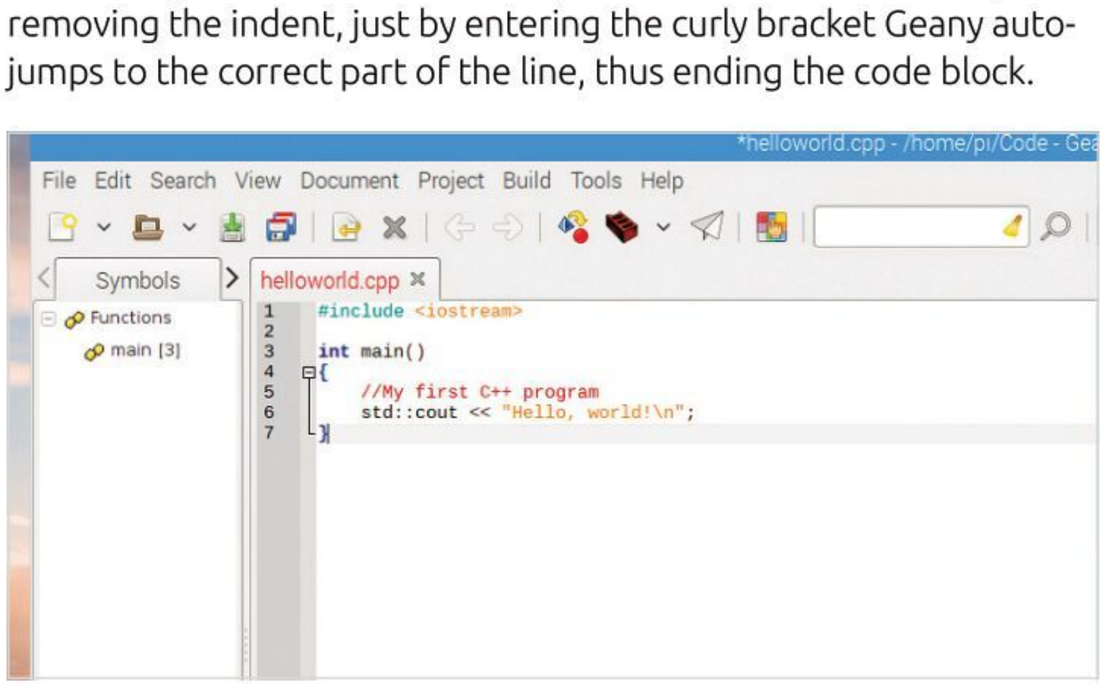
```
std::cout << "Hello, world!\n";
```



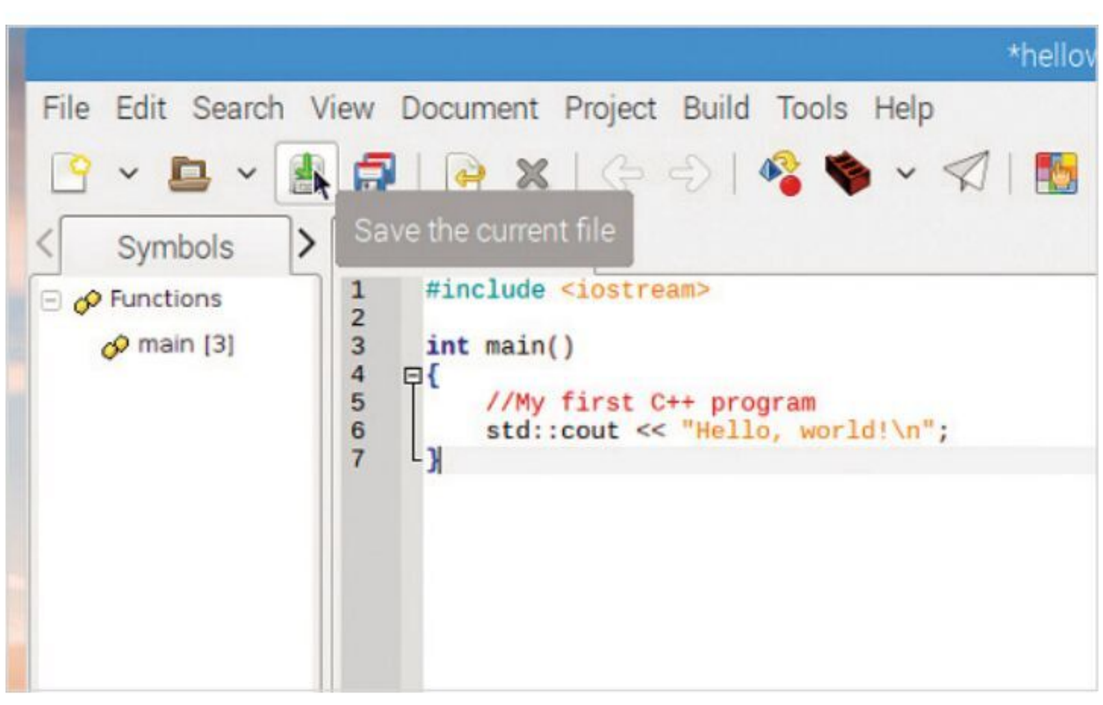
STEP 8 Now you need to close off the code, by inserting the opposite, closing, curly bracket. On line 7 enter:

```
}
```

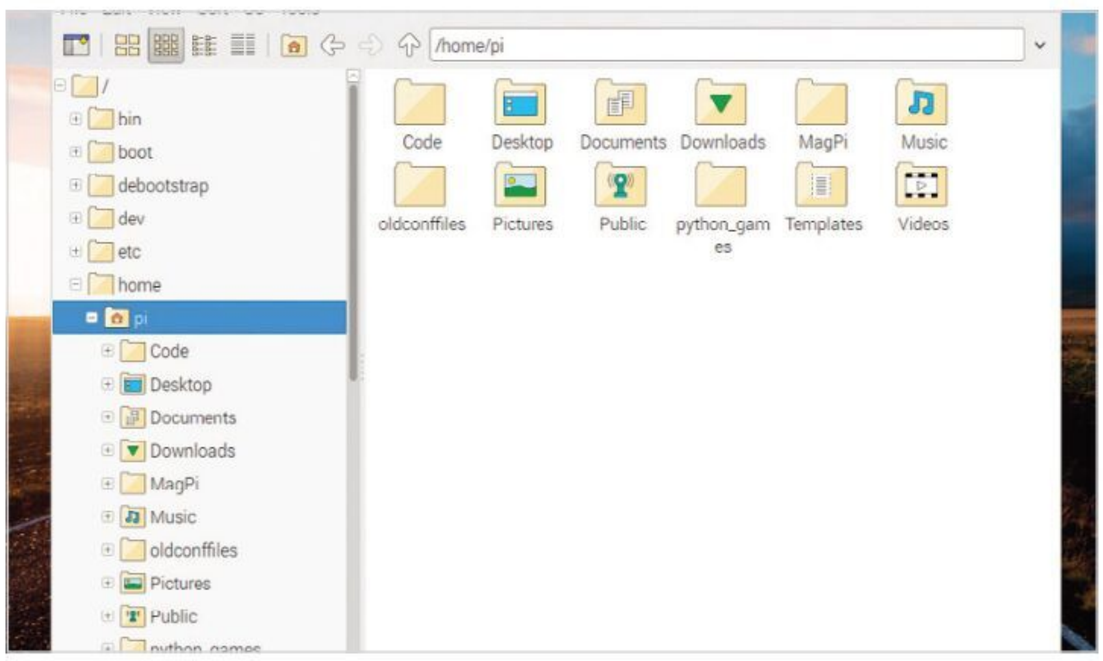
Note that you won't need to backspace to the start of the line, removing the indent, just by entering the curly bracket Geany auto-jumps to the correct part of the line, thus ending the code block.



STEP 9 Essentially, these are your first steps into the world of coding in C++. We will look at the structure of the code in more detail on the next couple of pages; for now though, click on the Save button as represented by a filing cabinet with a green downward pointing arrow.



STEP 10 If you haven't done so already, it's always advisable to create a blank folder where you can save all your code. You can have one each for Python and C++ or simply a single folder called Code, whichever you prefer. It's not a necessity, just good housekeeping.



Structure of a C++ Program

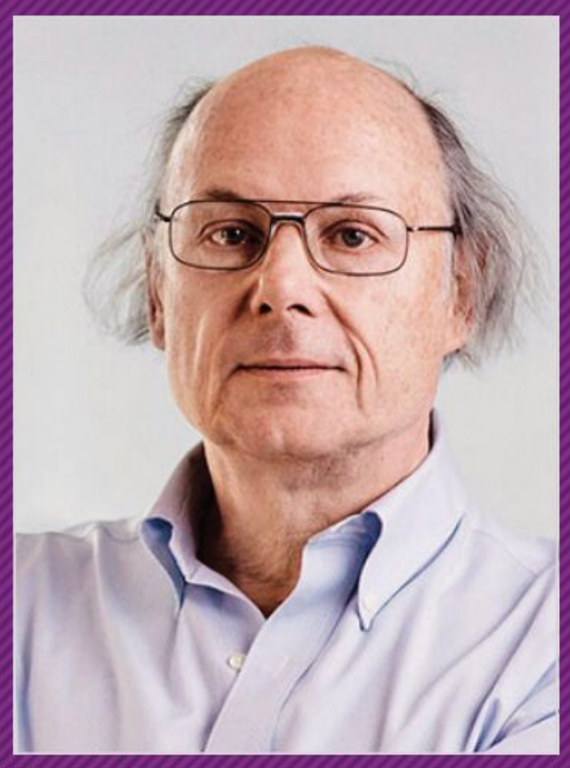
C++ has a very defined structure and way of doing things. Miss something out, even as small as a semicolon, and your entire program will fail to be compiled and executed. Many a professional programmer has fallen foul of sloppy structure.

#INCLUDE <C++ STRUCTURE>

Learning the basics of programming, you begin to understand the structure of a program. The commands may be different from one language to the next but you can start to see how the code works.

C++

C++ was invented by Danish student Bjarne Stroustrup in 1979, as a part of his Ph.D. thesis. Initially C++ was called C with Classes, which added features to the already popular C programming language, while making it a more user-friendly environment through a new structure.



Bjarne Stroustrup, inventor of C++.

INT MAIN()

int main() initiates the declaration of a function, which is a group of code statements under the name 'main'. All C++ code begins at the main function, regardless of where it actually lies within the code.

helloworld.cpp

```
1   #include <iostream>
2
3   int main()
4
5
```

#INCLUDE

The structure of a C++ program is quite precise. Every C++ code begins with a directive: #include <>. The directive instructs the pre-processor to include a section of the standard C++ code. For example: #include <iostream> includes the iostream header to support input/output operations.

helloworld.cpp

```
1   #include <iostream>
2
3
4
```

BRACES

The open brace (curly brackets) is something that you may not have come across before, especially if you're used to Python. The open brace indicates the beginning of the main function and contains all the code that belongs to that function.

helloworld.cpp

```
1   #include <iostream>
2
3   int main()
4   {
5
6
7   }
8
```

COMMENTS

Lines that begin with a double slash are comments. This means they won't be executed in the code and are ignored by the compiler. Comments are designed to help you, or another programmer looking at your code, and explain what's going on. There are two types of comment: `/*` covers multiple line comments and `//` a single line.

helloworld.cpp

```
#include <iostream>

int main()
{
    //My first C++ program

}
```

STD

While std stands for something quite different, in C++ it means Standard. It's part of the Standard Namespace in C++, which covers a number of different statements and commands. You can leave the `std::` part out of the code but it must be declared at the start with: using namespace std; not both. For example:

```
#include <iostream>
using namespace std;
```

helloworld.cpp

```
#include <iostream>

int main()
{
    //My first C++ program
    std::cout << "Hello, world!\n";
}
```

COUT

In this example we're using cout, which is a part of the Standard Namespace, hence why it's there, as you're asking C++ to use it from that particular namespace. Cout means Character OUTput, which displays, or prints, something to the screen. If you leave std:: out you have to declare it at the start of the code, as mentioned previously.

helloworld.cpp

```
#include <iostream>
using namespace std;

int main()
{
    //My first C++ program
    cout
}
```

<<

The two chevrons used here are insertion operators. This means that whatever follows the chevrons is to be inserted into the std::cout statement. In this case they're the words 'Hello, world' which are to be displayed on the screen when you compile and execute the code.

helloworld.cpp

```
#include <iostream>
using namespace std;

int main()
{
    //My first C++ program
    cout <<
}
```

OUTPUTS

Leading on, the "Hello, world!" part is what you want to appear on the screen when the code is executed. You can enter whatever you like, as long as it's inside the quotation marks. Sometimes brackets are needed, depending on the compiler. The \n part indicates a new line is to be inserted.

```
//My first C++ program
cout << "Hello, world!\n"
```

; AND }

Finally you can see that lines within a function code block (except comments) end with a semicolon. This marks the end of the statement; all statements in C++ must have one at the end or the compiler fails to build the code. The very last line has the closing brace to indicate the end of the main function.

helloworld.cpp

```
#include <iostream>
using namespace std;

int main()
{
    //My first C++ program
    cout << "Hello, world!\n";
}
```

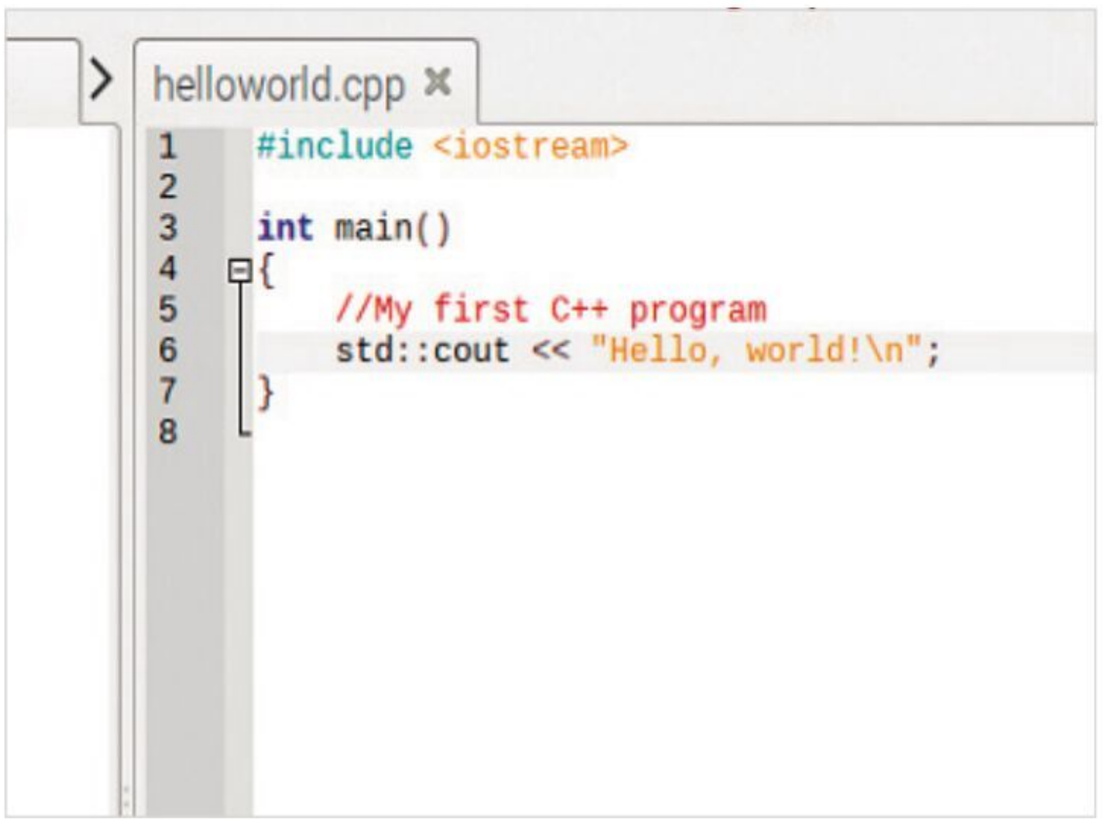
Compile and Execute

You've created your first C++ program and you now understand the basics behind the structure of one. Let's get things moving and compile and execute, or run if you prefer, the program and see how it looks.

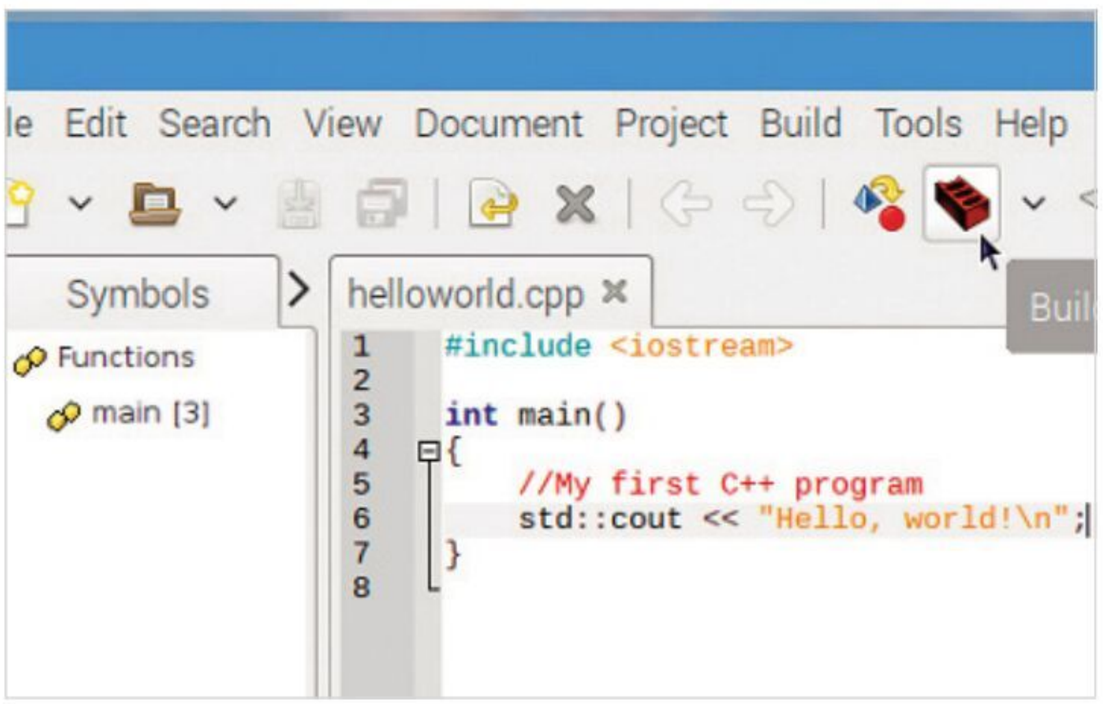
GREETINGS FROM C++

Compiling and executing code from within Geany is remarkably simple and just a matter of clicking a couple of icons and seeing the result. Here's how it's done.

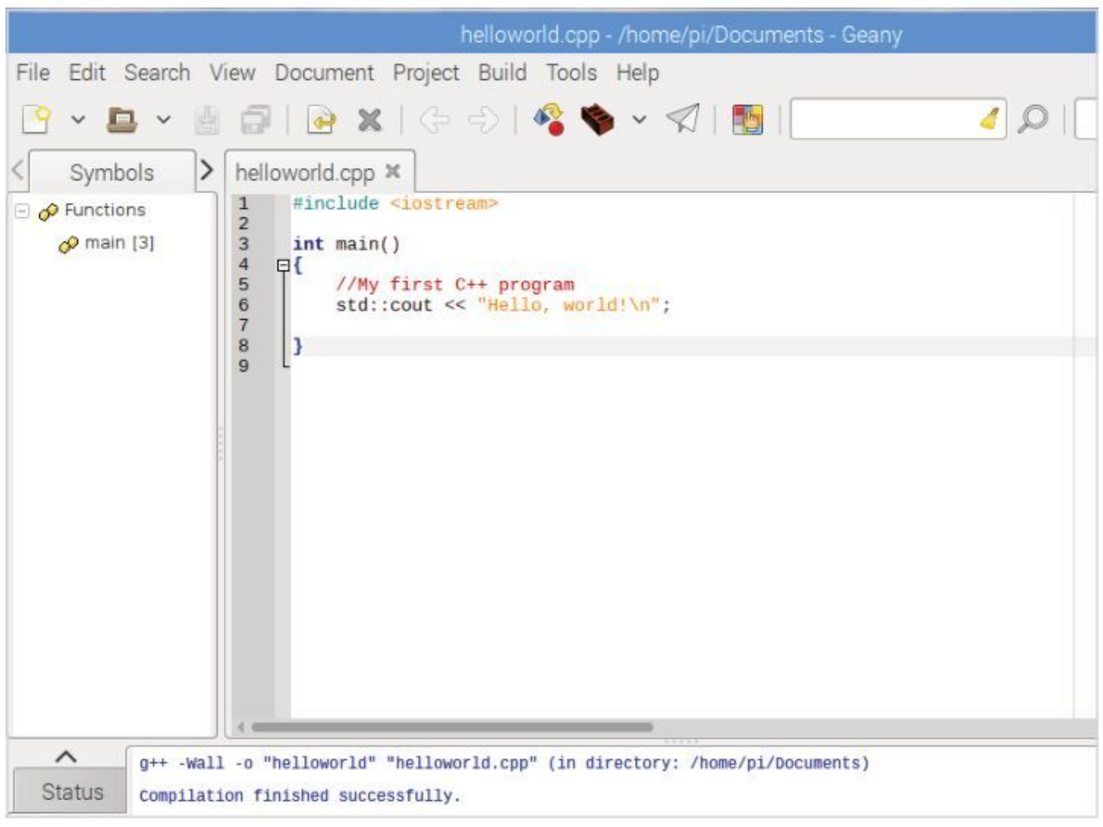
STEP 1 Open Geany, if you haven't already, and load up the previously saved Hello World code you created. Ensure that there are no visible errors, such as a missing semicolon at the end of the std::cout line.



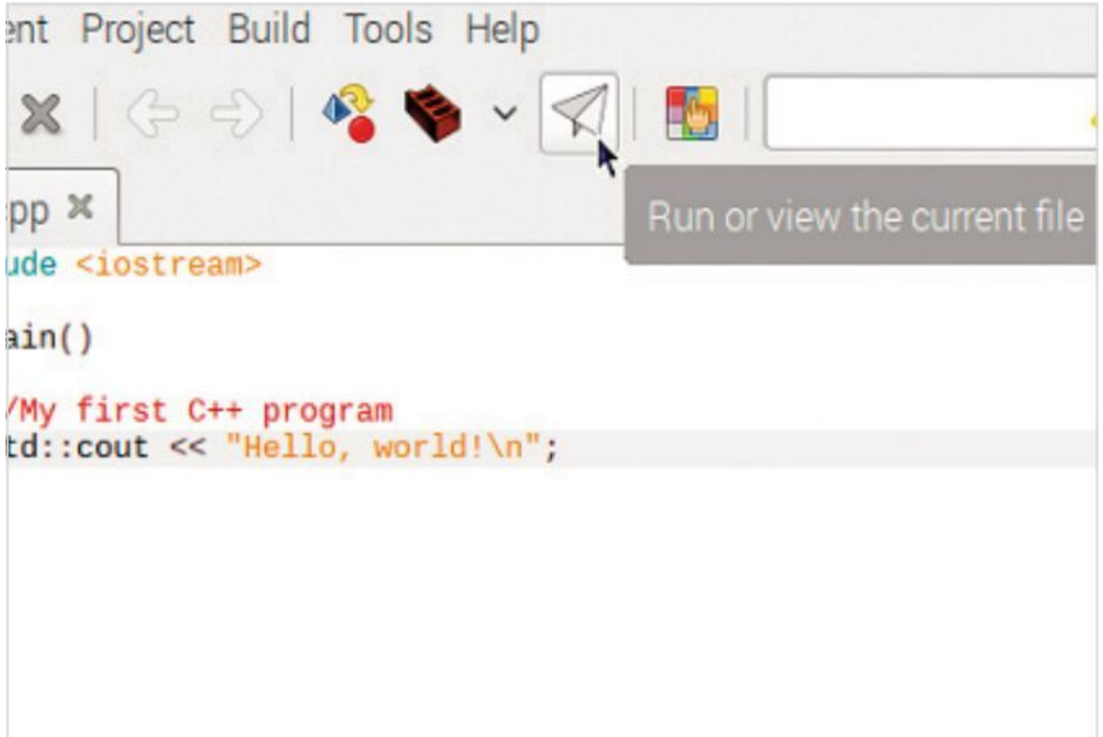
STEP 2 If your code is ready, look to the menu bar along the top of the Geany window. Notice that midway along the line of icons there's a red brick icon. This is Build, which when clicked runs through the code, checking it against the C++ standards to see if there are any errors that stop it from running.



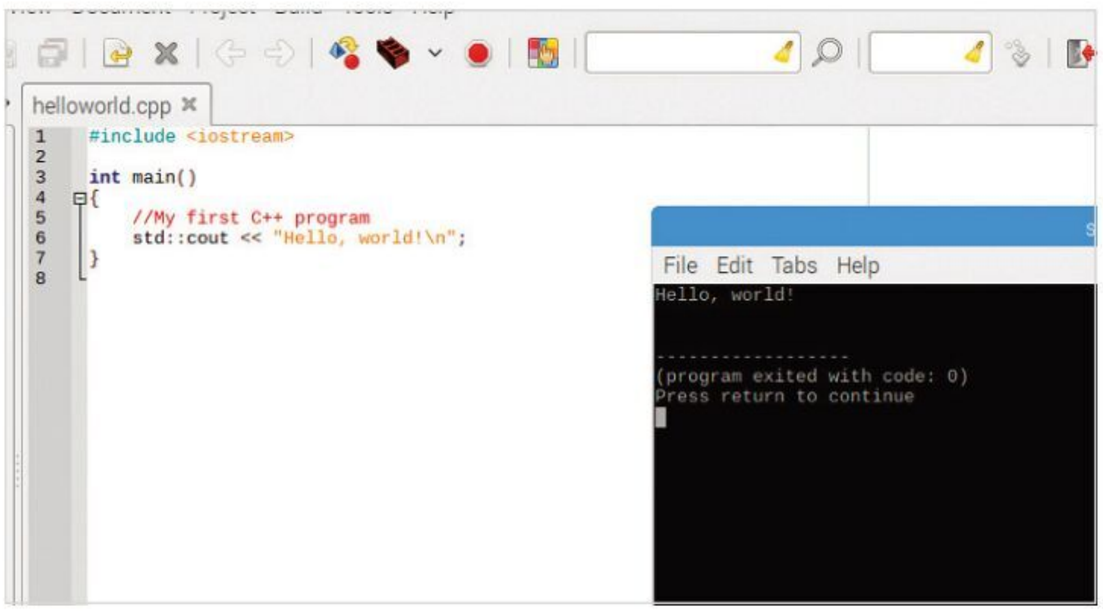
STEP 3 When you click the Build icon, the pane at the bottom of the Geany interface displays some results. If all is well, you will see a successful compilation, if not then Geany issues an error along with the line the error was found on.



STEP 4 With the code successfully built, it's now time to execute it. Next to the Build icon there's a paper aeroplane icon, this is the Run icon. You can't run code that hasn't been previously built or has an error in place. Click the Run icon.



STEP 5 When you click Run, the C++ code executes and a simple Terminal window appears displaying the results of the code. In the case of this simple code, you can see the words: Hello, world! in the window; as you've already learned, the code is cout(ing) (outputting) the contents between the quotes in the std::cout line.

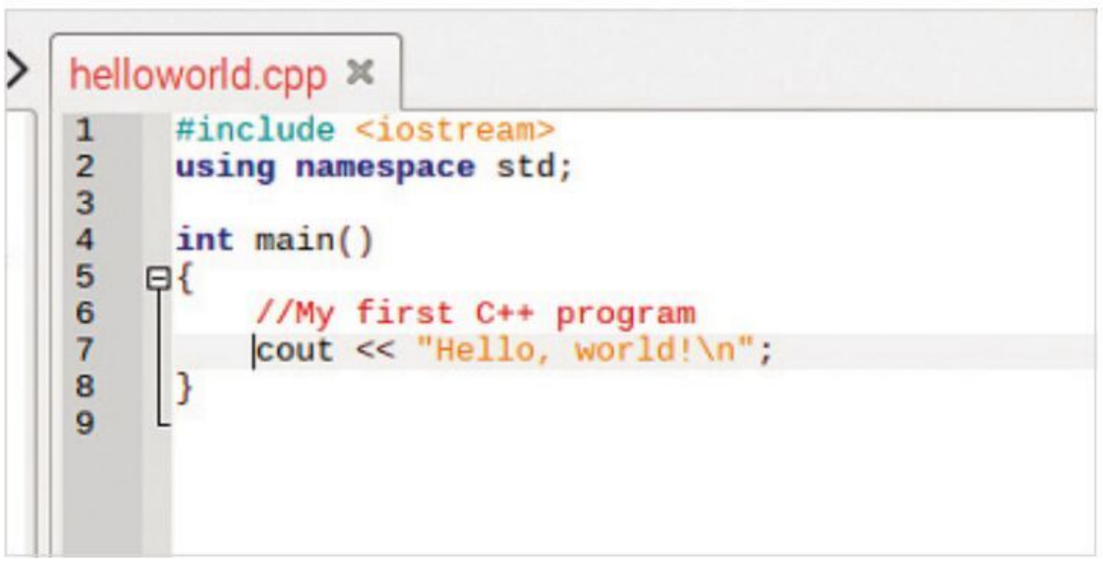


STEP 6 Pressing Return/Enter in the command line box closes it, returning you to Geany. Let's alter the code slightly. Under the #include line, enter:

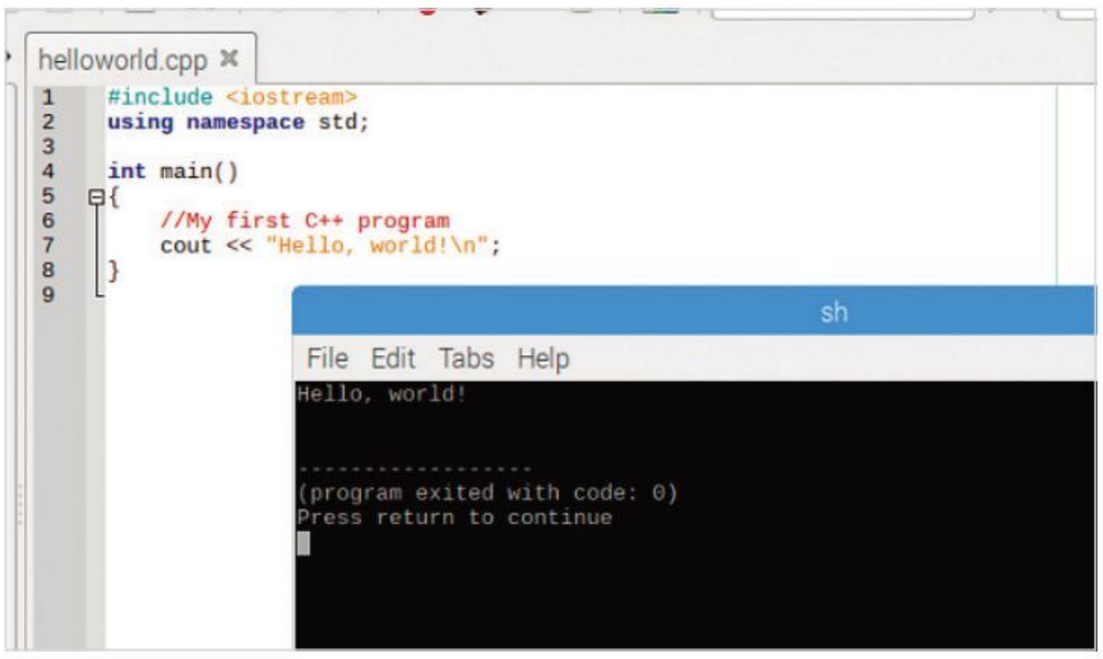
```
using namespace std;
```

Then, delete the std:: part of the Cout line; like so:

```
cout << "Hello, world\n";
```

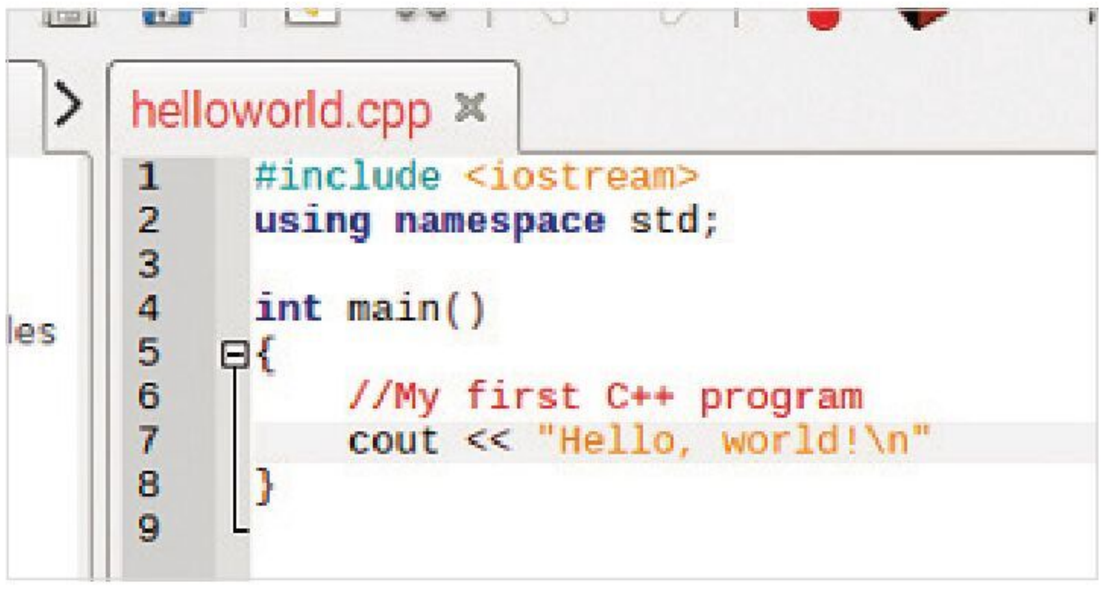


STEP 7 In order to apply the new changes you need to save the file, build it and execute it again. Go through the process as before, you don't need to change the name of the file unless you want to, of course, followed by clicking the Brick icon, then the Aeroplane icon.

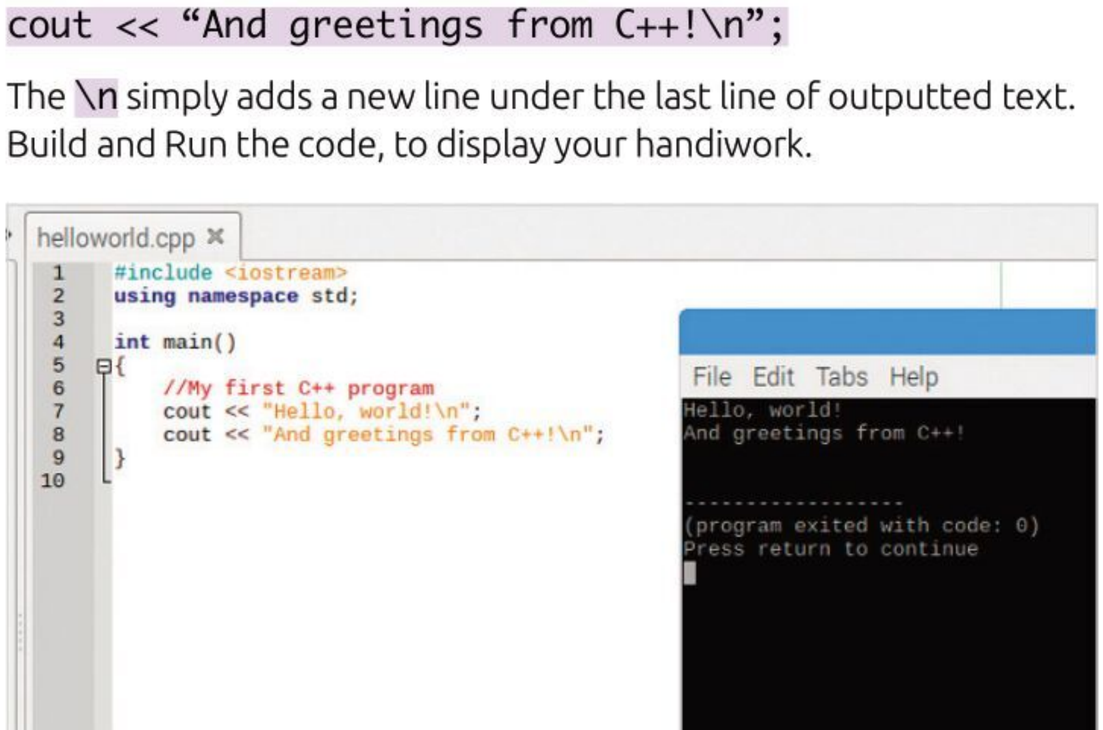


STEP 8 Just as mentioned previously, you don't need to have std::cout if you've already declared the standard namespace at the start of the code. Now, let's create a deliberate mistake, to see what happens. Remove the semicolon from the cout line, so it reads:

```
cout << "Hello, world!\n"
```



STEP 9 Now try and build the code once more. This time, notice that the bottom panel is displaying an error, regarding the missing semicolon. Notice that error reads as being on line 8, the first character, the closed curly bracket. This is because there's an expected semicolon missing before the closed bracket.



STEP 10 Replace the semicolon and under the cout line, enter a new line to your code:

```
cout << "And greetings from C++!\n";
```

The \n simply adds a new line under the last line of outputted text. Build and Run the code, to display your handiwork.

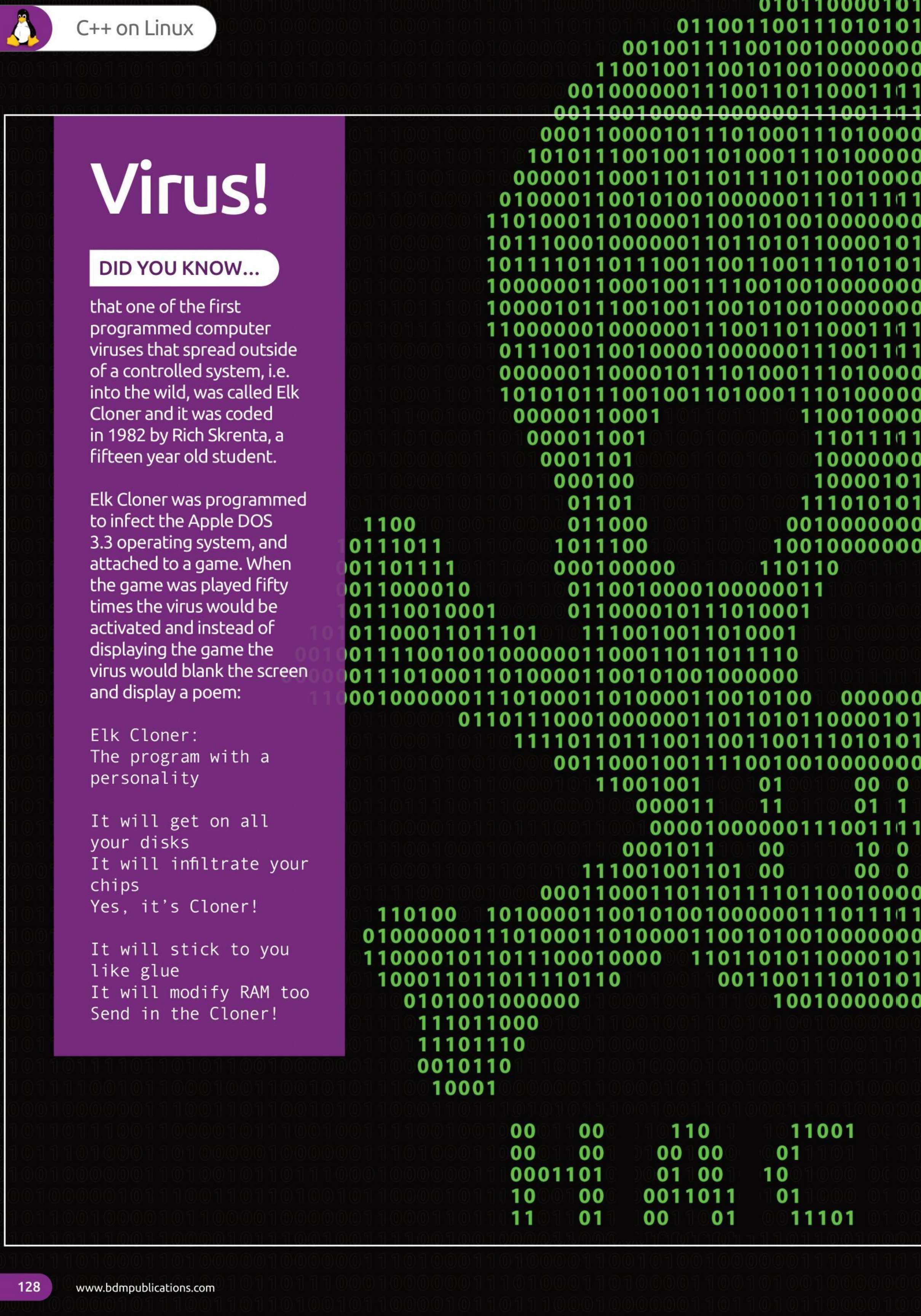
Virus!

DID YOU KNOW...

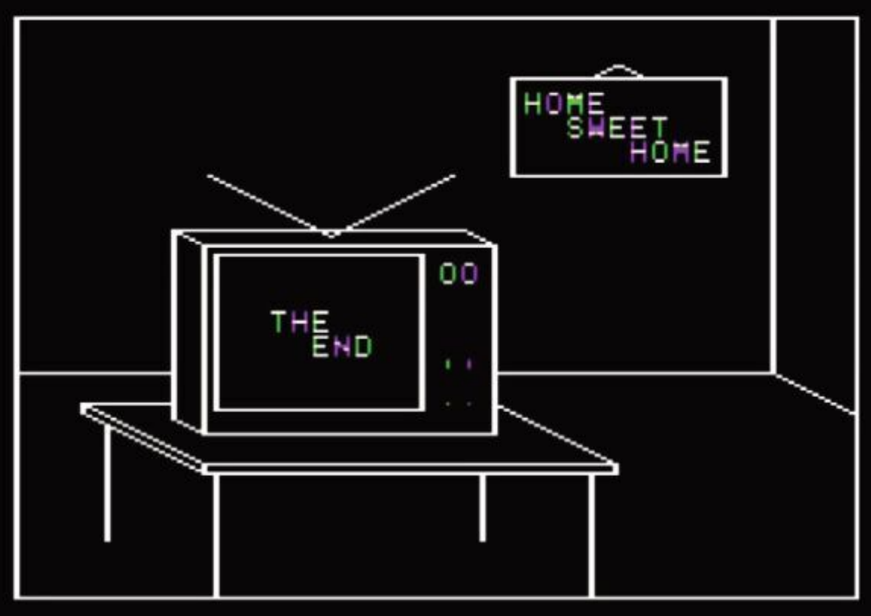
that one of the first programmed computer viruses that spread outside of a controlled system, i.e. into the wild, was called Elk Cloner and it was coded in 1982 by Rich Skrenta, a fifteen year old student.

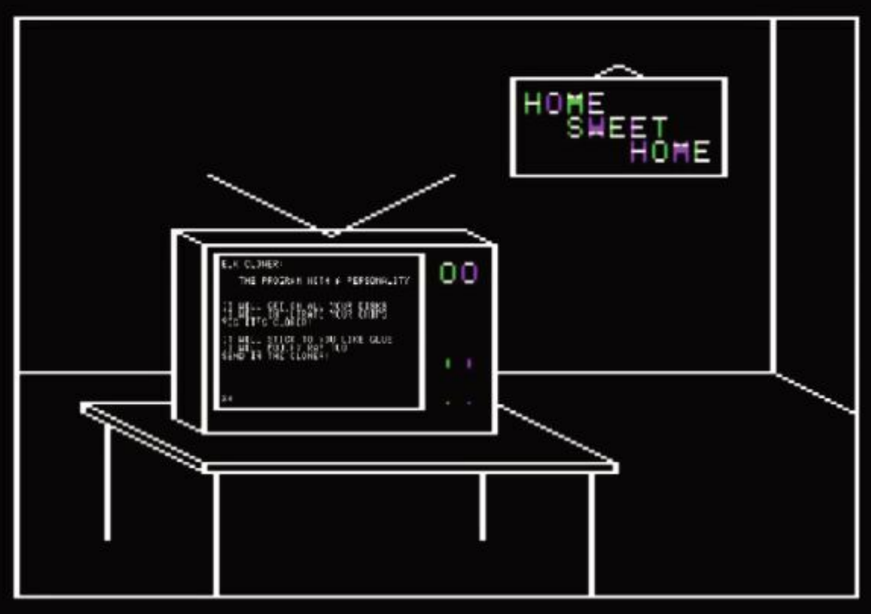
Elk Cloner was programmed to infect the Apple DOS 3.3 operating system, and attached to a game. When the game was played fifty times the virus would be activated and instead of displaying the game the virus would blank the screen and display a poem:

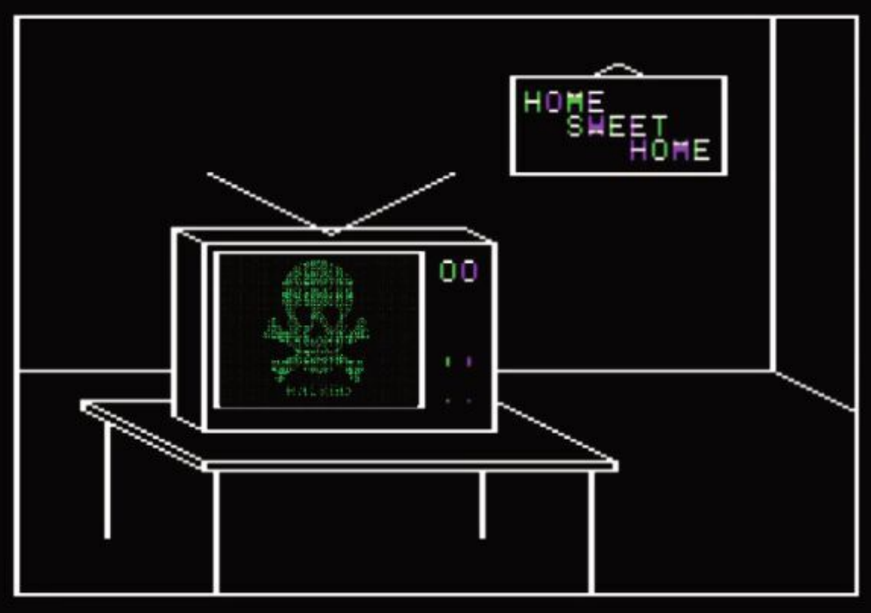
```
Elk Cloner:
The program with a
personality

It will get on all
your disks
It will infiltrate your
chips
Yes, it's Cloner!

It will stick to you
like glue
It will modify RAM too
Send in the Cloner!
```



Did You Know...







Developed as a prank back in '82, Elk Cloner was one of the first recorded viruses in the wild.

Using Comments

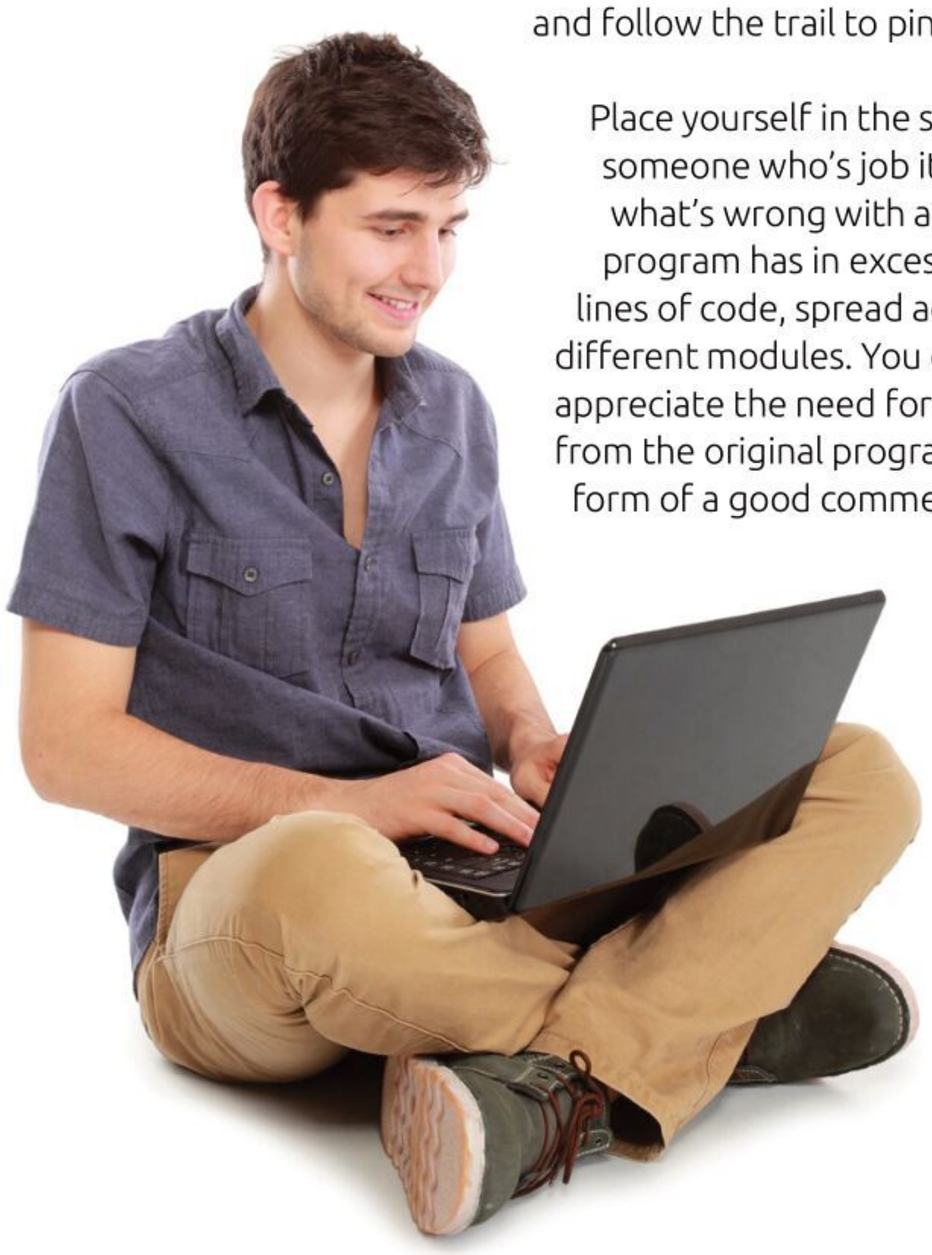
While comments may seem like a minor element to the many lines of code that combine to make a game, application or even an entire operating system, in actual fact they're probably one of the most important factors.

THE IMPORTANCE OF COMMENTING

Comments inside code are basically human readable descriptions that detail what the code is doing at that particular point. They don't sound especially important but code without comments is one of the many frustrating areas of programming, regardless of whether you're a professional or just starting out.

In short, all code should be commented in such a manner as to effectively describe the purpose of a line, section or individual elements. You should get in to the habit of commenting as much as possible, by imagining that someone who doesn't know anything about programming can pick up your code and understand what it's going to do simply by reading your comments.

In a professional environment, comments are vital to the success of the code and ultimately, the company or project. In an organisation, many programmers work in teams alongside engineers, other developers, hardware analysts and so on. If you're a part of the team that's writing a bespoke piece of software for the company, then your comments help save a lot of time should something go wrong and another team member has to pick up and follow the trail to pinpoint the issue.



Place yourself in the shoes of someone who's job it is to find out what's wrong with a program. The program has in excess of 800,000 lines of code, spread across several different modules. You can soon appreciate the need for a little help from the original programmers in the form of a good comment.

The best comments are always concise and link the code logically, detailing what happens when the program hits a line or section. You don't need to comment on every line. Something along the lines of: if x==0 doesn't require you to comment that if x equals zero then do something; that's going to be obvious to the reader. However, if x equalling zero is something that drastically changes the program for the user, such as, they've run out of lives, then it certainly needs to be commented on.

Even if the code is your own, you should write comments as if you were going to share it with others publicly. This way you can return to that code and always understand what it was you did or where it was you went wrong, or what worked brilliantly.

Comments are good practise and once you understand how to add a comment where needed, you soon do it as if it's second nature.

```
        DEFB    26h,30h,32h,26h,30h,32h,0,0,32h,72h,73h,32h,72h,73h,32h
        DEFB    60h,61h,32h,4Ch,4Dh,32h,4Ch,99h,32h,4Ch,4Dh,32h,4Ch,4Dh
        DEFB    32h,4Ch,99h,32h,5Bh,5Ch,32h,56h,57h,32h,33h,0CDh,32h,33h
        DEFB    34h,32h,33h,34h,32h,33h,0CDh,32h,40h,41h,32h,66h,67h,64h
        DEFB    66h,67h,32h,72h,73h,64h,4Ch,4Dh,32h,56h,57h,32h,80h,0CBh
        DEFB    19h,80h,0,19h,80h,81h,32h,80h,0CBh,0FFh
T858C:
        DEFB    80h,72h,66h,60h,56h,66h,56h,56h,51h,60h,51h,51h,56h,66h
        DEFB    56h,56h,80h,72h,66h,60h,56h,66h,56h,56h,51h,60h,51h,51h
        DEFB    56h,56h,56h,56h,80h,72h,66h,60h,56h,66h,56h,56h,51h,60h
        DEFB    51h,51h,56h,66h,56h,56h,80h,72h,66h,60h,56h,66h,56h,40h
        DEFB    56h,66h,80h,66h,56h,56h,56h,56h
;
; Game restart point
;
START:  XOR     A
        LD      (SHEET),A
        LD      (KEMP),A
        LD      (DEMO),A
        LD      (B845B),A
        LD      (B8458),A
        LD      A,2             ;Initial lives count
        LD      (NOMEN),A
        LD      HL,T845C
        SET     0,(HL)
        LD      HL,SCREEN
        LD      DE,SCREEN+1
        LD      BC,17FFh        ;Clear screen image
        LD      (HL),0
        LDIR
        LD      HL,0A000h       ;Title screen bitmap
        LD      DE,SCREEN
        LD      BC,4096
        LDIR
        LD      HL,SCREEN + 800h + 1*32 + 29
        LD      DE,MANDAT+64
        LD      C,0
        CALL    DRWFIX
        LD      HL,0FC00h       ;Attributes for the last room
        LD      DE,ATTR         ;(top third)
        LD      BC,256
        LDIR
        LD      HL,09E00h       ;Attributes for title screen
        LD      BC,512          ;(bottom two-thirds)
        LDIR
        LD      BC,31
        DI
        XOR     A
R8621:
        IN      E,(C)
        OR      E
        DJNZ    R8621   ;$-03
        AND     20h
        JR      NZ,R862F ;$+07
        LD      A,1
```

C++ COMMENTS

Commenting in C++ involves using a double forward slash '//' or a forward slash and an asterisk '/*'. You've already seen some brief examples but this is how they work.

STEP 1 Using the Hello World code as an example, you can easily comment on different sections of the code using the double forward slash:

```
//My first C++ program cout << "Hello, world!\n";
```

helloworld.cpp

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      //My first C++ program
7      cout << "Hello, world!\n";
8      cout << "And greetings from C++!\n";
9  }
10
```

STEP 2 However, you can also add comments to the end of a line of code, to describe in a better way what's going on:

```
cout << "Hello, world!\n"; //This line outputs the words 'Hello, world!'. The \n denotes a new line.
```

Note, you don't have to put a semicolon at the end of a comment. This is because it's a line in the code that's ignored by the compiler.

helloworld.cpp

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      //My first C++ program
7      cout << "Hello, world!\n"; //This line ouputs the words 'Hello, world. The \n denotes a new line.
8      cout << "And greetings from C++!\n";
9  }
10
```

STEP 3 You can comment out several lines by using the forward slash and asterisk:

```
/* This comment can
   cover several lines
   without the need to add more slashes */
```

Just remember to finish the block comment with the opposite asterisk and forward slash.

helloworld.cpp

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      //My first C++ program
7      cout << "Hello, world!\n"; //This line ouputs the words 'Hello, world. The \
8
9      /* This comment can
10      * cover several lines
11      * without the need to add more slashes */
12
13
14     cout << "And greetings from C++!\n";
15 }
16
```

STEP 4 Be careful when commenting, especially with block comments. It's very easy to forget to add the closing asterisk and forward slash and thus negate any code that falls inside the comment block.

helloworld.cpp

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      //My first C++ program
7      cout << "Hello, world!\n"; //This line ouputs the words 'Hello, w
8
9      /* This comment can
10      * cover several lines
11      * without the need to add more slashes
12
13
14     cout << "And greetings from C++!\n";
15 }
16
```

STEP 5 Obviously if you try and build and execute the code it errors out, complaining of a missing curly bracket '}' to finish off the block of code. If you've made the error a few times, then it can be time consuming to go back and rectify. Thankfully, the colour coding in Geany helps identify comments from code.

Symbols

- Functions
  - main [4]
- Extern Variables
  - std [2]

helloworld.cpp

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      //My first C++ program
7      cout << "Hello, world!\n";
8
9      /* This comment can
10      * cover several lines
11      * without the need to add more slashes
12
13
14     cout << "And greetings from C++!\n";
15 }
16
```

STEP 6 With block comments, it's good practise in C++ to add an asterisk to each new line of the comment block. This also helps you to remember to close the comment block off before continuing with the code:

```
/* This comment can
 *  cover several lines
 *  without the need to add more slashes */
```

Thankfully, Geany does this automatically but be aware of it when using other IDEs.

```
/* This comment can
 * cover several lines
 * without the need to add more slashes */
```

Variables

Variables differ slightly when using C++ as opposed to Python. In Python, you can simply state that 'a' equals 10 and a variable is assigned. However, in C++ a variable has to be declared with its type before it can be used.

THE DECLARATION OF VARIABLES

You can declare a C++ variable by using statements within the code. There are several distinct types of variable you can declare. Here's how it works.

STEP 1 Open up a new, blank C++ file and enter the usual code headers:

```
#include <iostream>
using namespace std;

int main()
{

}
```

Variables.cpp

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6
7  }
8
```

STEP 2 Start simple by creating two variables, a and b, with one having a value of 10 and the other 5. You can use the data type int to declare these variables. Within the curly brackets enter:

```
int a;
int b;

a = 10;
b = 5;
```

Variables.cpp

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int a;
7      int b;
8
9      a = 10;
10     b = 5;
11
12 }
13
```

STEP 3 You can build and run the code but it won't do much, other than store the values 10 and 5 to the integers a and b. Ignore the warnings about variables set but not used, the code still works. To output the contents of the variables, add:

```
cout << a;
cout << "\n";
cout << b;
```

The cout << "\n"; part simply places a new line between the output of 10 and 5.

Variables.cpp

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int a;
7      int b;
8
9      a = 10;
10     b = 5;
11
12     cout << a;
13     cout << "\n";
14     cout << b;
15
16
17 }
18
```

STEP 4 Naturally you can declare a new variable, call it result and output some simple arithmetic:

```
int result;

result = a + b;
cout << result;
```

Insert the above into the code as per the screenshot.

Variables.cpp

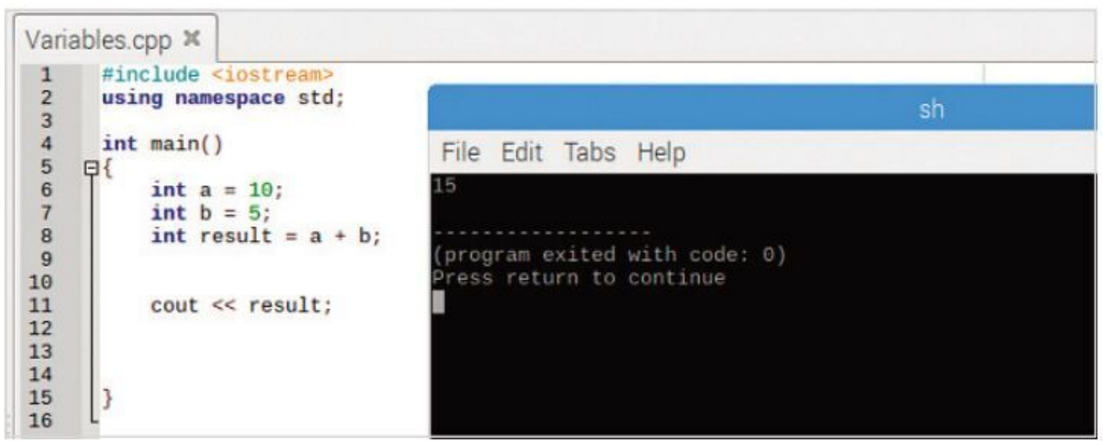
```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int a;
7      int b;
8      int result;
9
10     a = 10;
11     b = 5;
12     result = a + b;
13
14     cout << a;
15     cout << "\n";
16     cout << b;
17     cout << "\n";
18     cout << result;
19
20
21
22 }
23
```

sh

```
File  Edit  Tabs  Help
10
5
15

-----------------
(program exited with code: 0)
Press return to continue
```

STEP 5 You can assign a value to a variable as soon as you declare it. The code you've typed in could look like this, instead:

```
int a = 10;
int b = 5;
int result = a + b;
cout << result;
```

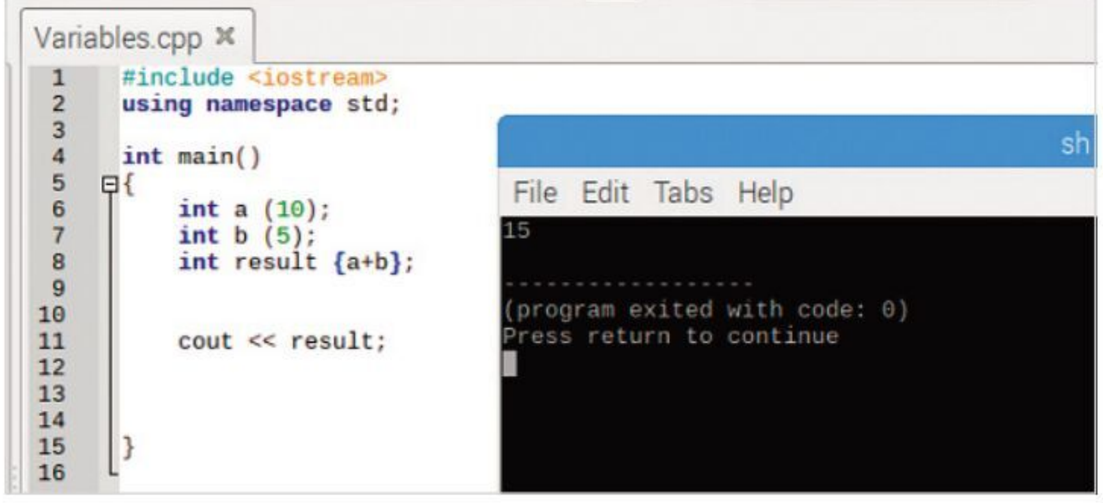


STEP 6 Specific to C++, you can also use the following to assign values to a variable as soon as you declare them:

```
int a (10);
int b (5);
```

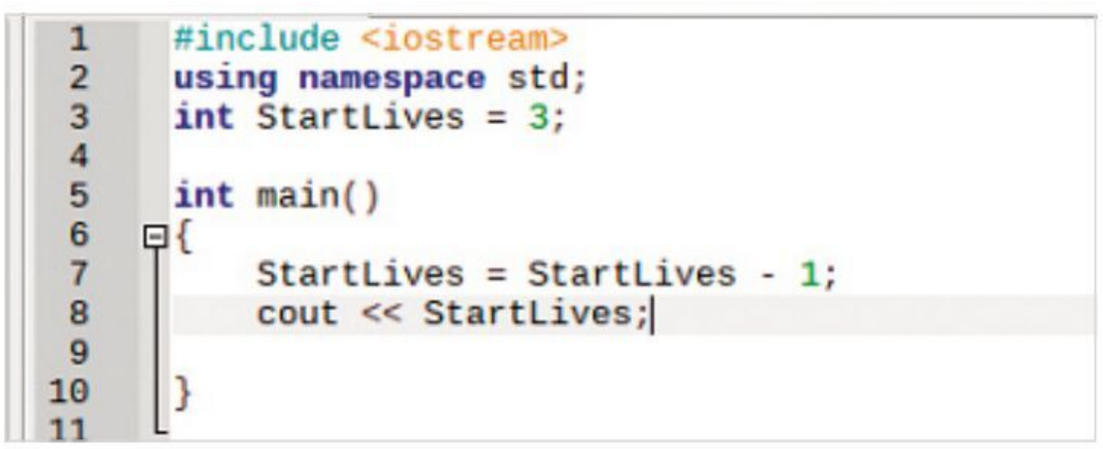
Then, from the C++ 2011 standard, using curly brackets:

```
int result {a+b};
```

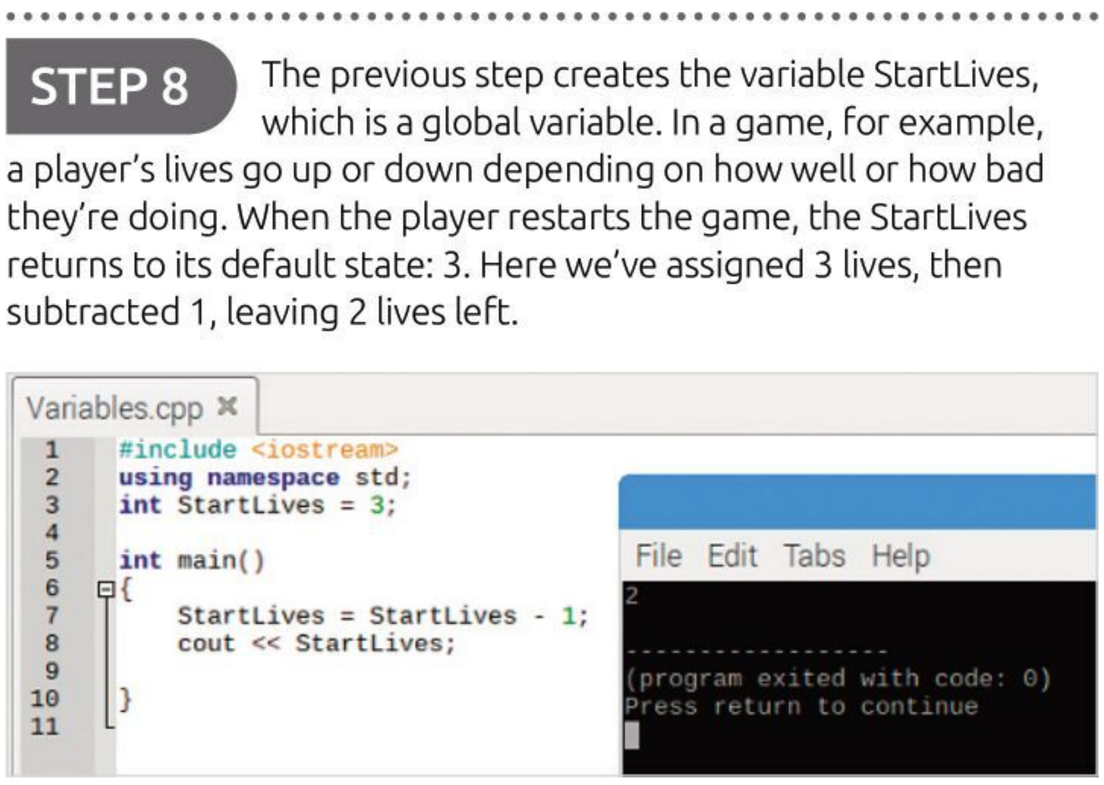


STEP 7 You can create global variables, which are variables that are declared outside any function and used in any function within the entire code. What you've used so far are local variables: variables used inside the function. For example:

```
#include <iostream>
using namespace std;
int StartLives = 3;

int main ()
{
    startLives = StartLives – 1;
    cout << StartLives;
```

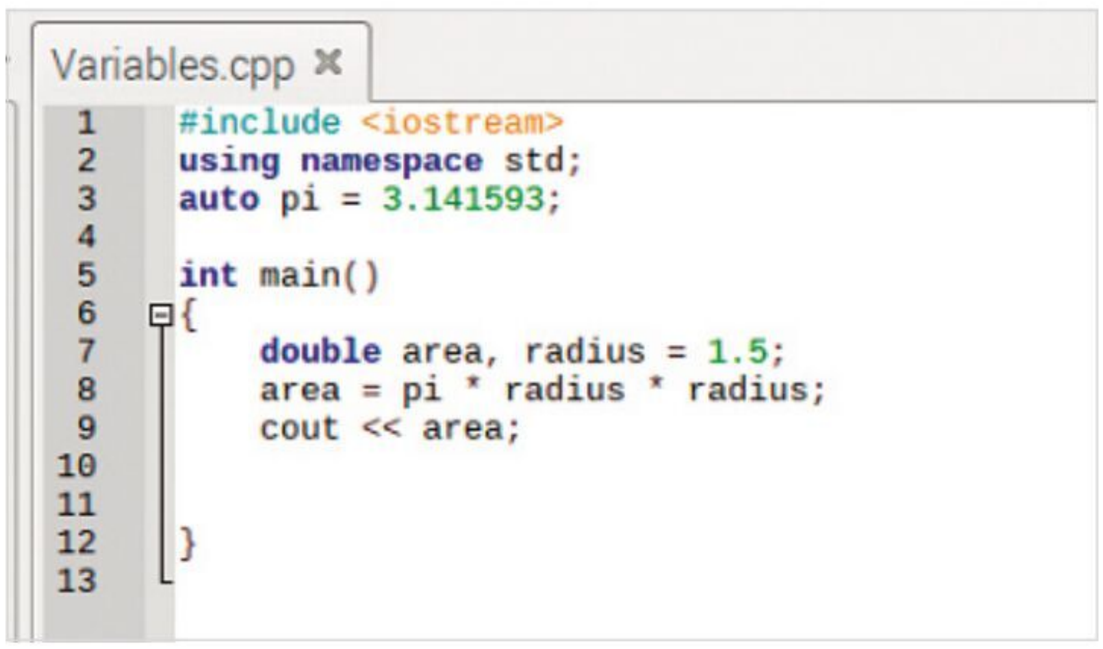


STEP 8 The previous step creates the variable StartLives, which is a global variable. In a game, for example, a player's lives go up or down depending on how well or how bad they're doing. When the player restarts the game, the StartLives returns to its default state: 3. Here we've assigned 3 lives, then subtracted 1, leaving 2 lives left.

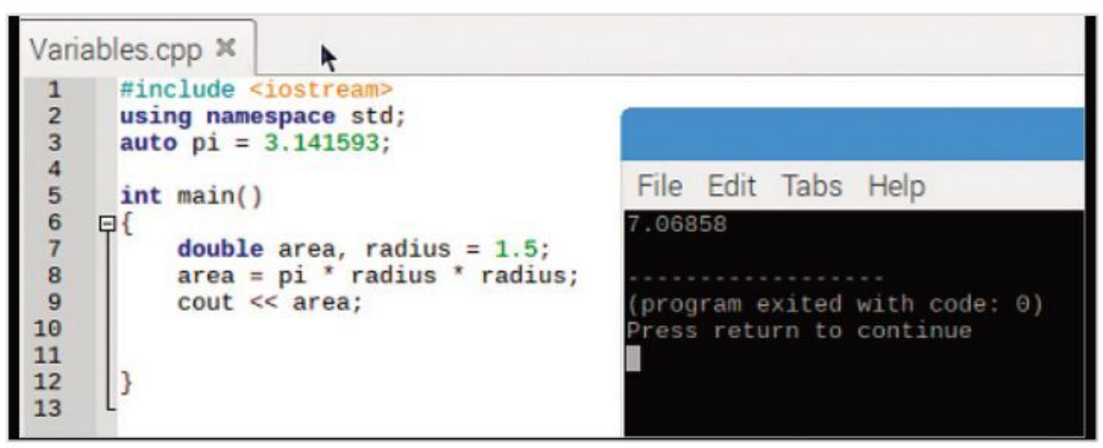


STEP 9 The modern C++ compiler is far more intelligent than most programmers give it credit. While there are numerous data types you can declare for variables, you can in fact use the auto feature:

```
#include <iostream>
using namespace std;
auto pi = 3.141593;

int main()
{
    double area, radius = 1.5;
    area = pi * radius * radius;
    cout << area;
```



STEP 10 Although we said to ignore the warnings in Step 3, there's a difference between a warning and an error. In this case, it's just a simple warning. The new data type, double, means double-precision floating point value, which makes the code more accurate. The result should be 7.06858. Essentially, you can often work with a warning but not an error.



Data Types

Variables, as you've seen, store information that the programmer can then later call up, and manipulate if required. Variables are simply reserved memory locations that store the values the programmer assigns, depending on the data type used.

THE VALUE OF DATA

There are many different data types available for the programmer in C++, such as an integer, floating point, Boolean, character and so on. It's widely accepted that there are seven basic data types, often called Primitive Built-in Types; however, you can create your own data types should the need ever arise within your code.

Type	Command
Integer	`int`
Floating Point	`float`
Character	`char`
Boolean	`bool`
Double Floating Point	`double`
Wide Character	`wchar_t`
No Value	`void`

These basic types can also be extended using the following modifiers: Long, Short, Signed and Unsigned. Basically this means the modifiers can expand the minimum and maximum range values for each data type. For example, the int data type has a default value range of -2147483648 to 2147483647, a fair value, you would agree.

Now, if you were to use one of the modifiers, the range alters:

Unsigned int = 0 to 4294967295
Signed int = -2147483648 to 2147483647
Short int = -32768 to 32767
Unsigned Short int = 0 to 65,535
Signed Short int = -32768 to 32767
Long int = -2147483647 to 2147483647
Signed Long int = -2147483647 to 2147483647
Unsigned Long int = 0 to 4294967295

Naturally you can get away with using the basic type without the modifier, as there's plenty of range provided with each data type. However, it's considered good C++ programming practise to use the modifiers when possible.

There are issues when using the modifiers though. Double represents a double-floating point value, which you can use for incredibly accurate numbers but those numbers are only accurate up to the fifteenth decimal place. There's also the problem when displaying such numbers in C++ using the cout function, in that cout by default only outputs the first five decimal places. You can combat that by adding a cout.precision () function and adding a value inside the brackets but even then you're still limited by the accuracy of the double data type. For example, try this code:

```
#include <iostream>
using namespace std;
double PI = 3.141592653589793238463;

int main()
{
    cout << PI;

}
```

DataTypes.cpp

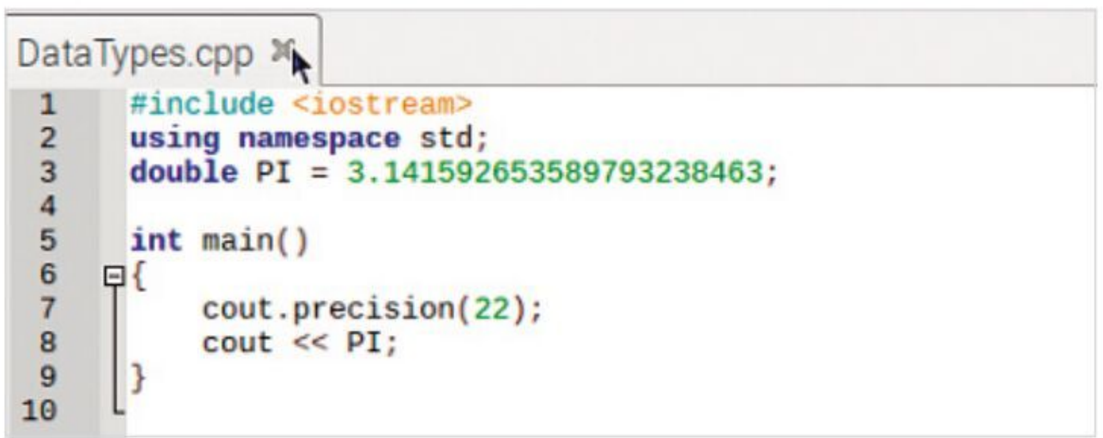
```
1  #include <iostream>
2  using namespace std;
3  double PI = 3.141592653589793238463;
4
5  int main()
6  {
7      cout << PI;
8  }
9
```

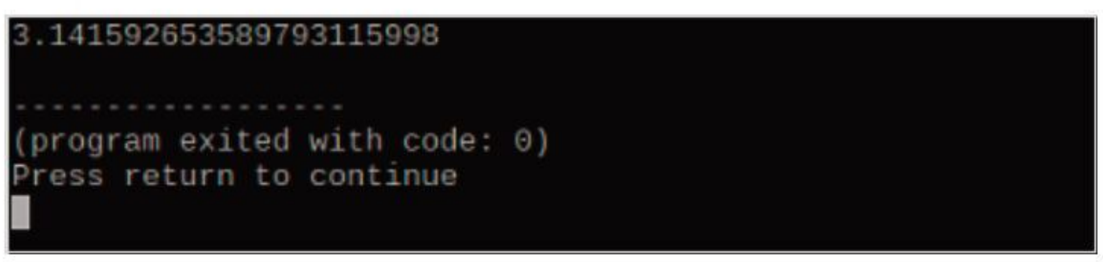
```
3.14159

-----------------
(program exited with code: 0)
Press return to continue
```

Build and run the code and as you can see the output is only 3.14159, representing cout's limitations in this example.

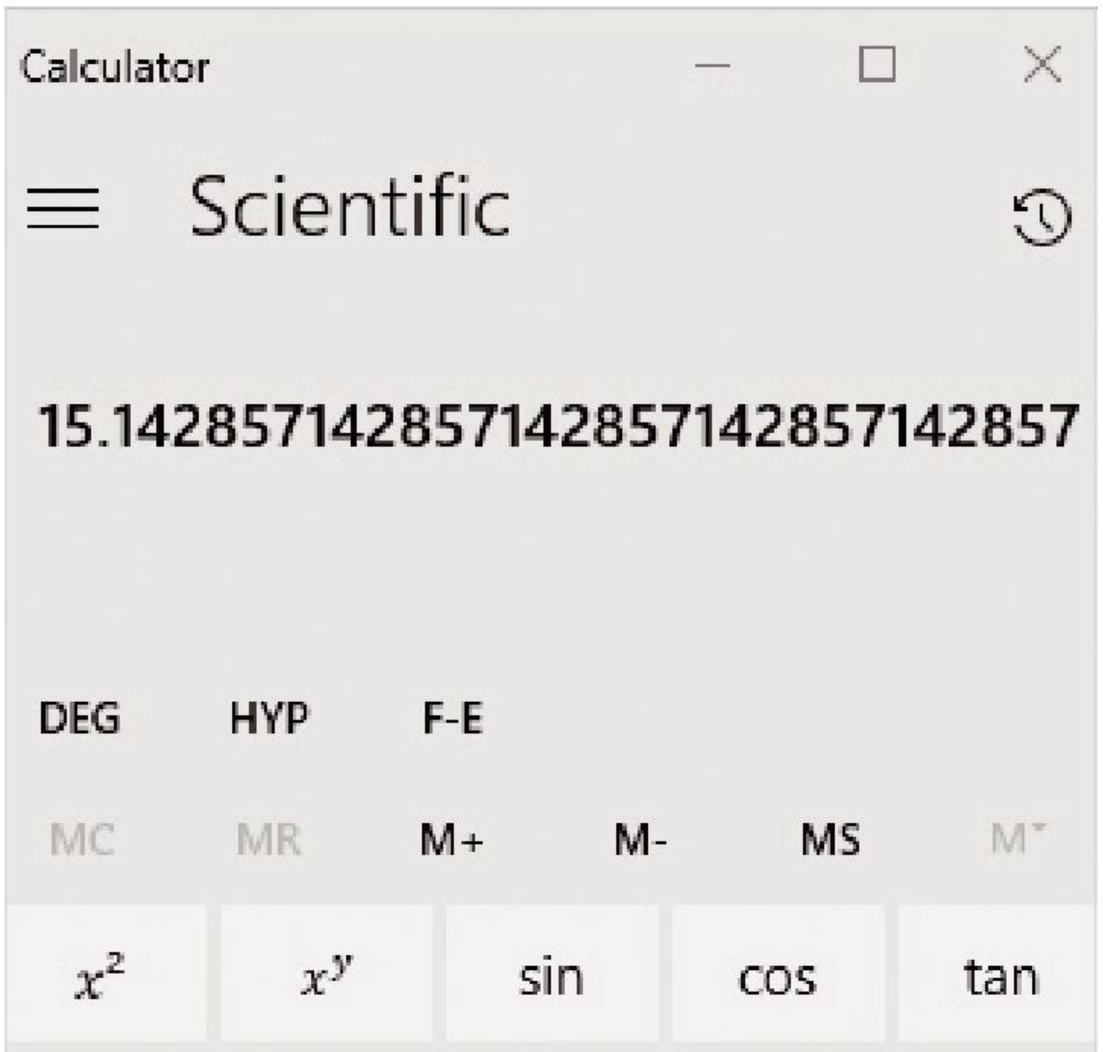
You can alter the code including the aforementioned cout.precision function, for greater accuracy. Take precision all the way up to 22 decimal places, with the following code:

```
#include <iostream>
using namespace std;
double PI = 3.141592653589793238463;

int main()
{
    cout.precision(22);
    cout << PI;

}
```





Again, build and run the code; as you can see from the command line window, the number represented by the variable PI is different to the number you've told C++ to use in the variable. The output reads the value of PI as 3.141592653589793115998, with the numbers going awry from the fifteenth decimal place.

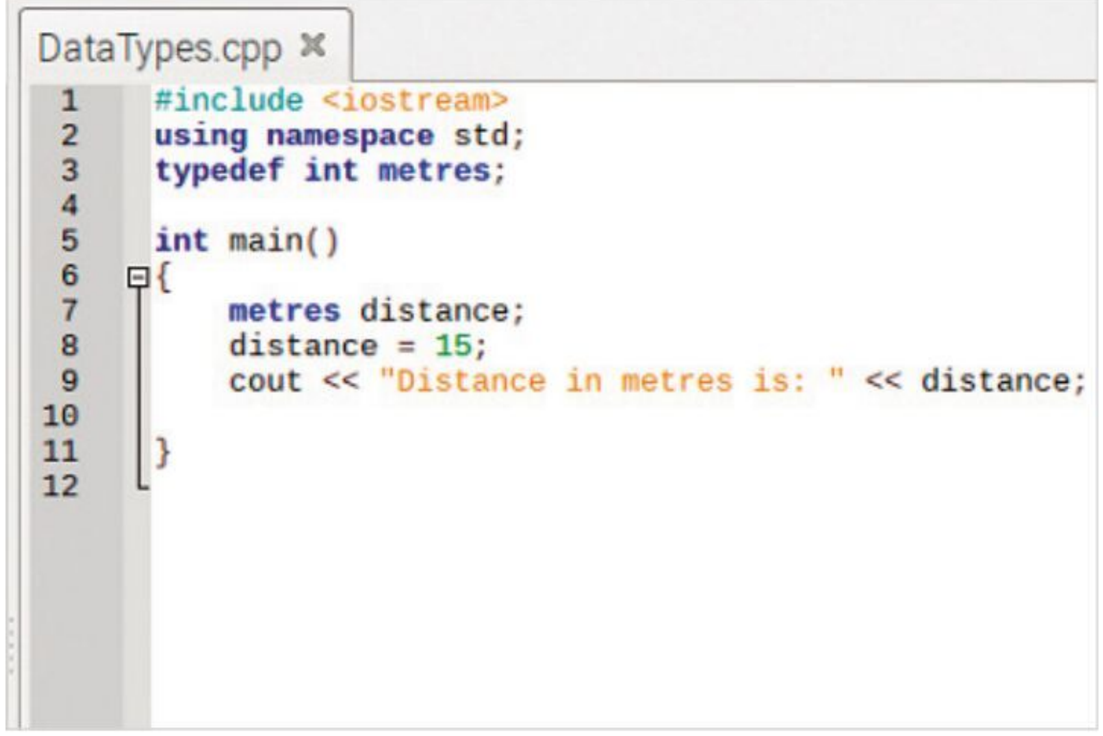


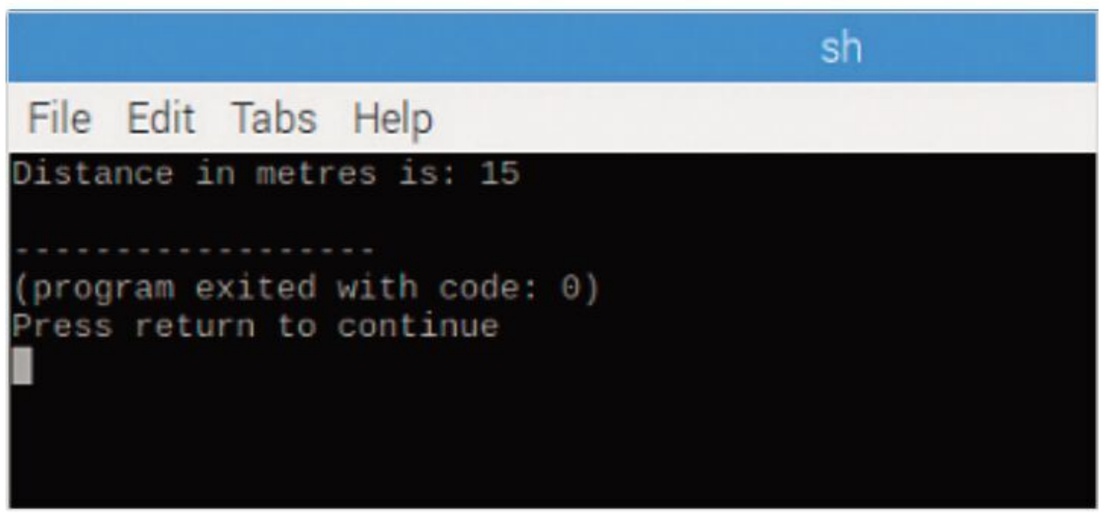
This is mainly due to the conversion from binary in the compiler and that the IEEE 754 double precision standard occupies 64-bits of data, of which 52-bits are dedicated to the significant (the significant digits in a floating-point number) and roughly 3.5-bits are taken holding the values 0 to 9. If you divide 53 by 3.5, then you arrive at 15.142857 recurring, which is 15-digits of precision.

To be honest, if you're creating code that needs to be accurate to more than fifteen decimal places, then you wouldn't be using C++, you would use some scientific specific language with C++ as the connective tissue between the two languages.

You can create your own data types, using an alias-like system called typedef. For example:

```
#include <iostream>
using namespace std;
typedef int metres;

int main()
{
    metres distance;
    distance = 15;
    cout << "distance in metres is: " << distance;

}
```





This code when executed creates a new int data type called metres. Then, in the main code block, there's a new variable called distance, which is an integer; so you're telling the compiler that there's another name for int. We assigned the value 15 to distance and displayed the output: distance in metres is 15.

It might sound a little confusing to begin with but the more you use C++ and create your own code, the easier it becomes.

Strings

Strings are objects that represent and hold sequences of characters. For example, you could have a universal greeting in your code 'Welcome' and assign that as a string to be called up wherever you like in the program.

STRING THEORY

There are different ways in which you can create a string of characters, which historically are all carried over from the original C language and still supported by C++.

STEP 1 To create a string, you use the char function. Open a new C++ file and begin with the usual header:

```
#include <iostream>
using namespace std;

int main ()
{

}
```

strings.cpp

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6
7  }
8
```

STEP 2 It's easy to confuse a string with an array. Here's an array, which can be terminated with a null character:

```
#include <iostream>
using namespace std;

int main ()

{
    char greet[8] = {'W', 'e', 'l', 'c', 'o', 'm',
'e', '\0'};
    cout << greet << "\n";

}
```

strings.cpp

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      char greet[8] = {'W', 'e', 'l', 'c', 'o', 'm', 'e', '\0'};
7      cout << greet << "\n";
8
9  }
10
```

STEP 3 Build and run the code and 'Welcome' appears on the screen. While this is perfectly fine, it's not a string. A string is a class, which defines objects that can be represented as a stream of characters and doesn't need to be terminated like an array. The code can therefore be represented as:

```
#include <iostream>
using namespace std;

int main ()
{
    char greet[] = "Welcome";
    cout << greet << "\n";

}
```

strings.cpp

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      char greet[] = "Welcome";
7      cout << greet << "\n";
8
9  }
10
```

STEP 4 In C++ there's also a string function, which works in much the same way. Using the greeting code again, you can enter:

```
#include <iostream>
using namespace std;
int main ()
{
    string greet = "Welcome";
    cout << greet << "\n";

}
```

strings.cpp

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      string greet = "Welcome";
7      cout << greet << "\n";
8
9  }
10
```

STEP 5

There are also many different operations that you can apply with the string function. For instance, to get the length of a string you can use:

```
#include <iostream>
using namespace std;

int main ()
{
    string greet = “Welcome”;
    cout << “The length of the string is: “;
    cout << greet.size() << “\n”;
```

strings.cpp

```
1   #include <iostream>
2   using namespace std;
3
4   int main()
5   {
6       string greet = "Welcome";
7       cout << "The length of the string is: ";
8       cout << greet.size() << "\n";
9
10  }
11
```

STEP 6

You can see that we used greet.size() to output the length, the number of characters there are, of the contents of the string. Naturally, if you call your string something other than greet, then you need to change the command to reflect this. It’s always stringname.operation. Build and run the code to see the results.

```
The length of the string is: 7


----------------
(program exited with code: 0)
Press return to continue
```

STEP 7

You can of course add strings together, or rather combine them to form longer strings:

```
#include <iostream>
using namespace std;

int main ()
{
    string greet1 = “Hello”;
    string greet2 = “, world!”;
    string greet3 = greet1 + greet2;
    cout << greet3 << “\n”;
}
```

strings.cpp

```
1   #include <iostream>
2   using namespace std;
3
4   int main()
5   {
6       string greet1 = "Hello";
7       string greet2 = ", world!";
8       string greet3 = greet1 + greet2;
9
10      cout << greet3 << "\n";
11
12  }
13
```

STEP 8

Just as you might expect, you can mix in an integer and store something to do with the string. In this example, we created int length, which stores the result of string. size() and outputs it to the user:

```
#include <iostream>
using namespace std;

int main ()
{
    int length;
    string greet1 = “Hello”;
    string greet2 = “, world!”;
    string greet3 = greet1 + greet2;

    length = greet3.size();

    cout << “The length of the combined strings
is: “ << length << “\n”;

}
```

STEP 9

Using the available operations that come with the string function, you can manipulate the contents of a string. For example, to remove characters from a string you could use:

```
#include <iostream>
using namespace std;

int main ()
{
  string strg (“Here is a long sentence in a
string.”);
  cout << strg << ‘\n’;

  strg.erase (10,5);
  cout << strg << ‘\n’;

  strg.erase (strg.begin()+8);
  cout << strg << ‘\n’;

  strg.erase (strg.begin()+9, strg.end()-9);
  cout << strg << ‘\n’;

}
```

STEP 10

It’s worth spending some time playing around with the numbers, which are the character positions in the string. Occasionally, it can be hit and miss whether you get it right, so practice makes perfect. Take a look at the screenshot to see the result of the code.

```
sh
File  Edit  Tabs  Help
Here is a long sentence in a string.
Here is a sentence in a string.
Here is  sentence in a string.
Here is  a string.


------------------
(program exited with code: 0)
Press return to continue
```

C++ Maths

Programming is mathematical in nature and as you might expect, there's plenty of built-in scope for some quite intense maths. C++ has a lot to offer someone who's implementing mathematical models into their code. It can be extremely complex or relatively simple.

C++ = MC2

The basic mathematical symbols apply in C++ as they do in most other programming languages. However, by using the C++ Math Library, you can also calculate square roots, powers, trig and more.

STEP 1

C++'s mathematical operations follow the same patterns as those taught in school, in that multiplication and division take precedence over addition and subtraction. You can alter that though. For now, create a new file and enter:

```
#include <iostream>
using namespace std;

int main ()
{
    float numbers = 100;

    numbers = numbers + 10; // This adds 10 to the initial 100

    cout << numbers << "\n";

    numbers = numbers - 10; // This subtracts 10 from the new 110

    cout << numbers << "\n";

}
```

maths.cpp

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      float numbers = 100;
7      numbers = numbers + 10; //This adds 10 to the initial 100
8
9      cout << numbers << "\n";
10
11     numbers = numbers - 10; //This subtracts 10 from the new 110
```

STEP 2

While simple, it does get the old maths muscle warmed up. Note that we used a float for the numbers variable. While you can happily use an integer, if you suddenly started to use decimals, you would need to change to a float or a double, depending on the accuracy needed. Run the code and see the results.

```
110
100


------------------
(program exited with code: 0)
Press return to continue
```

STEP 3

Multiplication and division can be applied as such:

```
#include <iostream>
using namespace std;

int main ()
{
    float numbers = 100;

    numbers = numbers * 10; // This multiplies 100 by 10

    cout << numbers << "\n";

    numbers = numbers / 10; // And this divides 1000 by 10

    cout << numbers << "\n";

}
```

maths.cpp

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      float numbers = 100;
7      numbers = numbers * 10; //This multiplies 100 by 10
8
9      cout << numbers << "\n";
10
11     numbers = numbers / 10; //And this divides 1000 by 10
12
13     cout << numbers << "\n";
14
15
16 }
17
```

STEP 4

Again, execute the simple code and see the results. While not particularly interesting, it's an introduction to C++ maths. We used a float here, so you can play around with the code and multiply by decimal places, as well as divide, add and subtract.

```
File  Edit  Tabs  Help
1000
100


------------------
(program exited with code: 0)
Press return to continue
```

STEP 5 The interesting maths content comes when you call upon the C++ Math Library. Within this header are dozens of mathematical functions along with further operations. Everything from computing cosine to arc tangent with two parameters, to the value of PI. You can call the header with:

```
#include <iostream>
#include <cmath>
using namespace std;

int main ()
{

}
```

maths.cpp

```
1  #include <iostream>
2  #include <cmath>
3  using namespace std;
4
5  int main()
6  {
7
8
9
10 }
11
```

STEP 6 Start by getting the square root of a number:

```
#include <iostream>
#include <cmath>
using namespace std;

int main ()
{
    float number = 134;

    cout << “The square root of “ << number << “ is: “ << sqrt(number) << “\n”;

}
```

maths.cpp

```
1  #include <iostream>
2  #include <cmath>
3  using namespace std;
4
5  int main()
6  {
7      float number = 134;
8
9      cout << "The square root of " << number << " is: "
10
11
12 }
13
```

STEP 7 Here we created a new float called number and used the sqrt(number) function to display the square root of 134, the value of the variable, number. Build and run the code and your answer reads 11.5758.

```
The square root of 134 is: 11.5758


-----------------
(program exited with code: 0)
Press return to continue
```

STEP 8 Calculating powers of numbers can be done with:

```
#include <iostream>
#include <cmath>
using namespace std;

int main ()
{
    float number = 12;
    cout << number << “ to the power of 2 is “ << pow(number, 2) << “\n”;
    cout << number << “ to the power of 3 is “ << pow(number, 3) << “\n”;
    cout << number << “ to the power of 0.8 is “ << pow(number, 0.8) << “\n”;

}
```

maths.cpp

```
1  #include <iostream>
2  #include <cmath>
3  using namespace std;
4
5  int main()
6  {
7      float number = 12;
8
9      cout << number << " to the power of 2 is " << pow(number, 2) << "\n";
10
```

STEP 9 Here we created a float called number with the value of 12 and the pow(variable, power) is where the calculation happens. Of course, you can calculate powers and square roots without using variables. For example, pow (12, 2) outputs the same value as the first cout line in the code.

```
12 to the power of 2 is 144
12 to the power of 3 is 1728
12 to the power of 0.8 is 7.30037


-----------------
(program exited with code: 0)
Press return to continue
```

STEP 10 The value of Pi is also stored in the cmath header library. It can be called up with the M_PI function. Enter cout << M_PI; into the code and you get 3.14159; or you can use it to calculate:

```
#include <iostream>
#include <cmath>
using namespace std;

int main ()
{
    double area, radius = 1.5;

    area = M_PI * radius * radius;

    cout << area << “\n”;

}
```

maths.cpp

```
1  #include <iostream>
2  #include <cmath>
3  using namespace std;
4
5  int main()
6  {
7      double area, radius = 1.5;
8
9      area = M_PI * radius * radius;
10
11     cout << area << "\n";
12
13
14 }
15
```

sh

File Edit Tabs Help

```
7.06858


-----------------
(program exited with code: 0)
Press return to continue
```

User Interaction

There’s nothing quite as satisfying as creating a program that responds to you. This basic user interaction is one of the most taught aspects of any language and with it you’re able to do much more than simply greet the user by name.

HELLO, DAVE

You have already used cout, the standard output stream, throughout our code. Now you’re going to be using cin, the standard input stream, to prompt a user response.

STEP 1 Anything that you want the user to input into the program needs to be stored somewhere in the system memory, so it can be retrieved and used. Therefore, any input must first be declared as a variable, so it’s ready to be used by the user. Start by creating a blank C++ file with headers.

userinteraction.cpp

```
#include <iostream>
using namespace std;

int main()
{

}
```

STEP 2 The data type of the variable must also match the type of input you want from the user. For example, to ask a user their age, you would use an integer like this:

```
#include <iostream>
using namespace std;

int main ()
{
   int age;
   cout << “what is your age: “;
   cin >> age;

   cout <<”\nYou are “ << age << “ years old.\n”;
}
```

userinteraction.cpp

```
#include <iostream>
using namespace std;

int main()
{
    int age;
    cout << "What is your age: ";
    cin >> age;

    cout << "\nYou are " << age << " years old.\n";


}
```

STEP 3 The cin command works in the opposite way from the cout command. With the first cout line you’re outputting ‘What is your age’ to the screen, as indicated with the chevrons. Cin uses opposite facing chevrons, indicating an input. The input is put into the integer age and called up in the second cout command. Build and run the code.

sh

File Edit Tabs Help

```
What is your age: 45

You are 45 years old.


----------------
(program exited with code: 0)
Press return to continue
```

STEP 4 If you’re asking a question, you need to store the input as a string; to ask the user their name, you would use:

```
#include <iostream>
using namespace std;

int main ()
{
   string name;
   cout << “what is your name: “;
   cin >> name;

   cout << “\nHello, “ << name << “. I hope you’re well today?\n”;

}
```

userinteraction.cpp

```
#include <iostream>
using namespace std;

int main()
{
    string name;
    cout << "What is your name: ";
    cin >> name;

    cout << "\nHello, " << name << ". I hope you're well today>\n";


}
```

STEP 5 The principal works the same as the previous code. The user's input, their name, is stored in a string, because it contains multiple characters, and retrieved in the second cout line. As long as the variable 'name' doesn't change, then you can recall it wherever you like in your code.

```
What is your name: David

Hello, David. I hope you're well today>


----------------
(program exited with code: 0)
Press return to continue
```

STEP 6 You can chain input requests to the user but just make sure you have a valid variable to store the input to begin with. Let's assume you want the user to enter two whole numbers:

```
#include <iostream>
using namespace std;

int main ()
{
   int num1, num2;

   cout << "Enter two whole numbers: ";
   cin >> num1 >> num2;

   cout << "you entered " << num1 << " and " << num2 << "\n";

}
```

userinteraction.cpp

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int num1, num2;
7
8      cout << "Enter two whole numbers: ";
9      cin >> num1 >> num2;
10
11     cout << "You entered " << num1 << " and " << num2 << "\n";
12
```

STEP 7 Likewise, inputted data can be manipulated once you have it stored in a variable. For instance, ask the user for two numbers and do some maths on them:

```
#include <iostream>
using namespace std;

int main ()
{
   float num1, num2;

   cout << "Enter two numbers: \n";
   cin >> num1 >> num2;

   cout << num1 << " + " << num2 << " is: " <<
num1 + num2 << "\n";

}
```

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int num1, num2;
7
8      cout << "Enter two whole numbers: ";
9      cin >> num1 >> num2;
10
11     cout << num1 << " + " << num2 << " is: " << num1 + num2 << "\n";
12
```

STEP 8 While cin works well for most input tasks, it does have a limitation. Cin always considers spaces as a terminator, so it's designed for just single words not multiple words. However, getline takes cin as the first argument and the variable as the second:

```
#include <iostream>
using namespace std;

int main ()
{
   string mystr;
   cout << "Enter a sentence: \n";
   getline(cin, mystr);

   cout << "Your sentence is: " << mystr.size() <<
" characters long.\n";

}
```

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      string mystr;
7      cout << "Enter a sentence: \n";
8      getline(cin, mystr);
9
10     cout << "Your sentence is: " << mystr.size() << " charcters long.\n";
11
12
```

STEP 9 Build and execute the code, then enter a sentence with spaces. When you're done the code reads the number of characters. If you remove the getline line and replace it with cin >> mystr and try again, the result displays the number of characters up to the first space.

```
Enter a sentence:
BDM Publications' Raspberry Pi: Tricks, Hacks & Fixes
Your sentence is: 53 charcters long.


----------------
(program exited with code: 0)
Press return to continue
```

STEP 10 Getline is usually a command that new C++ programmers forget to include. The terminating white space is annoying when you can't figure out why your code isn't working. In short, it's best to use getline(cin, variable) in future:

```
#include <iostream>
using namespace std;

int main ()
{
   string name;
   cout << "Enter your full name: \n";
   getline(cin, name);

   cout << "\nHello, " << name << ".\n";

}
```

userinteraction.cpp

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      string name;
7      cout << "Enter your full name: \n";
8      getline(cin, name);
9
10     cout << "\nHello, " << name << ".\n";
11
```

The Hobbit

DID YOU KNOW...

released in 1982 for various platforms, Melbourne House's The Hobbit was a coding work of genius. Developed by Philip Mitchell and Dr. Veronika Megler, The Hobbit was uniquely coded using a parser called 'Inglish' which allowed the player to enter full sentences, such as "Take sword from Gandalf, and kill goblin with it."

Furthermore, each object in the game had a specific size, weight and solidity; so if you sat on a stool, and someone picked up the stool or threw it, then your character would be taken along with it. Also, in early versions of the game, the non-playing characters would often move around the map and capture and kill each other before the player had even reached that stage in the game. There was even a randomiser event in place with each start of the game, where any character could pick up a random object from any location. This meant that, potentially, it was possible for Thorin to pick you up and carry you for the entire adventure. Remarkable for a game that took up less than 38KB of memory.

No time to sit down and start singing about gold, Thorin.

The Hobbit, a marvel in advanced coding from 1982.

Common Coding Mistakes

When you start something new you're inevitably going to make mistakes, this is purely down to inexperience and those mistakes are great teachers in themselves. However, even experts make the occasional mishap. Thing is, to learn from them as best you can.

X=MISTAKE, PRINT Y

There are many pitfalls for the programmer to be aware of, far too many to be listed here. Being able to recognise a mistake and fix it is when you start to move into more advanced territory, and become a better coder. Everyone makes mistakes, even coders with over thirty years' experience. Learning from these basic, common mistakes help build a better coding foundation.

SMALL CHUNKS

It would be wonderful to be able to work like Neo from The Matrix movies. Simply ask, your operator loads it into your memory and you instantly know everything about the subject. Sadly though, we can't do that. The first major pitfall is someone trying to learn too much, too quickly. So take coding in small pieces and take your time.

EASY VARIABLES

Meaningful naming for variables is a must to eliminate common coding mistakes. Having letters of the alphabet is fine but what happens when the code states there's a problem with x variable. It's not too difficult to name variables lives, money, player1 and so on.

```
var points = 1023;
var lives = 3;
var totalTime = 45;
write("Points: "+points);
write("Lives: "+lives);
write("Total Time: "+totalTime+" secs");
write("--------------------------");
var totalScore = 0;
write("Your total Score is: "+totalScore);
```

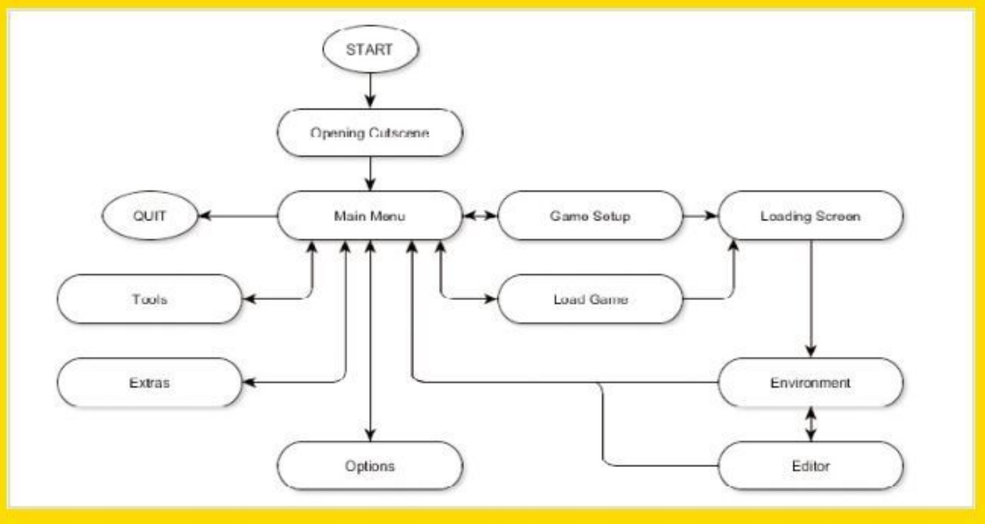
//COMMENTS

Use comments. It's a simple concept but commenting on your code saves so many problems when you next come to look over it. Inserting comment lines helps you quickly sift through the sections of code that are causing problems; also useful if you need to review an older piece of code.

```
        orig += 2;
        target += 2;
        --n;
    }
#endif
  if (n == 0)
    return;

  //
  // Loop unrolling. Here be dragons.
  //

  // (n & (~3)) is the greatest multiple of 4 n
  // In the while loop ahead, orig will move ov
  // increments (4 elements of 2 bytes).
  // end marks our barrier for not falling outs
  char const * const end = orig + 2 * (n & (~3)

  // See if we're aligned for writting in 64 or
#if ACE_SIZEOF_LONG == 8 && \
    !(defined(__amd64__) || defined (__x86_64
```

PLAN AHEAD

While it's great to wake up one morning and decide to code a classic text adventure, it's not always practical without a good plan. Small snippets of code can be written without too much thought and planning but longer and more in-depth code requires a good working plan to stick to and help iron out the bugs.



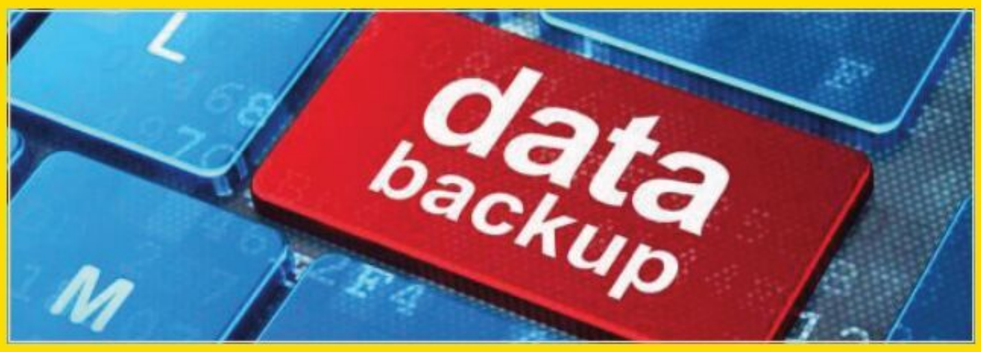
USER ERROR

User input is often a paralysing mistake in code. For example, when the user is supposed to enter a number for their age and instead they enter it in letters. Often a user can enter so much into an input that it overflows some internal buffer, thus sending the code crashing. Watch those user inputs and clearly state what's needed from them.

```
Enter an integer number
aswdfdsf
You have entered wrong input
s
You have entered wrong input
!"£"!£!"
You have entered wrong input
sdfsdf213213123
You have entered wrong input
123234234234234234
You have entered wrong input
12
the number is: 12

Process returned 0 (0x0)   execution time : 21.495 s
Press any key to continue.
```

BACKUPS

Always make a backup of your work, with a secondary backup for any changes you've made. Mistakes can be rectified if there's a good backup in place to revert to for those times when something goes wrong. It's much easier to start where you left off, rather than starting from the beginning again.



RE-INVENTING WHEELS

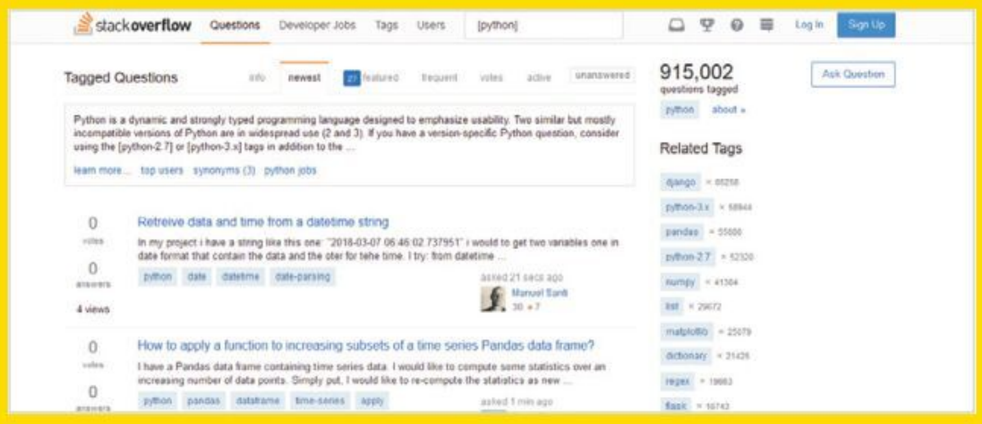
You can easily spend days trying to fathom out a section of code to achieve a given result and it's frustrating and often time-wasting. While it's equally rewarding to solve the problem yourself, often the same code is out there on the Internet somewhere. Don't try and re-invent the wheel, look to see if some else has done it first.

SECURE DATA

If you're writing code to deal with usernames and passwords, or other such sensitive data, then ensure that the data isn't in cleartext. Learn how to create a function to encrypt sensitive data, prior to feeding into a routine that can transmit or store it where someone may be able to get to view it.

HELP!

Asking for help is something most of us has struggled with in the past. Will the people we're asking laugh at us? Am I wasting everyone's time? It's a common mistake for someone to suffer in silence. However, as long as you ask the in the correct manner, obey any forum rules and be polite, then your question isn't silly.



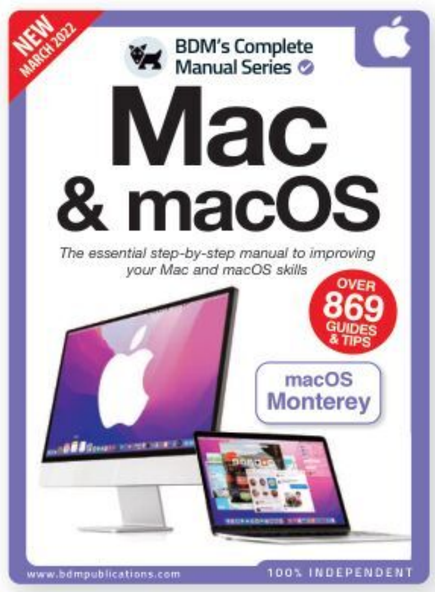
MATHS

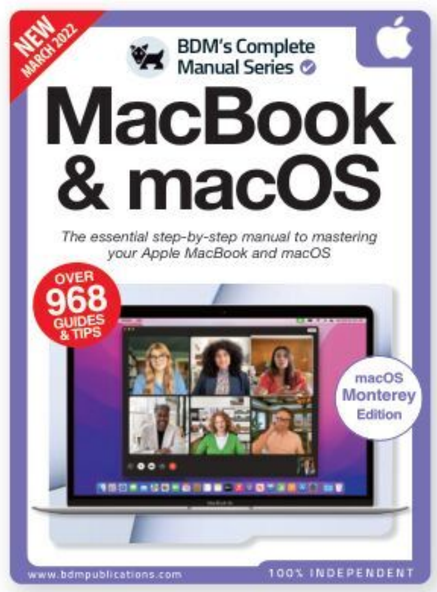
If your code makes multiple calculations then you need to ensure that the maths behind it is sound. There are thousands of instances where programs have offered incorrect data based on poor Mathematical coding, which can have disastrous effects depending on what the code is set to do. In short, double check your code equations.

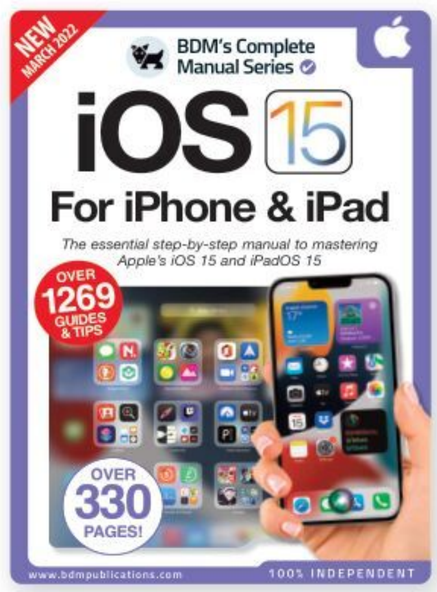
```
set terminal x11
set output

rmax = 5
nmax = 100

complex (x, y) = x * {1, 0} + y * {0, 1}
mandel (x, y, z, n) = (abs (z)> rmax || n>= 100)? n: mandel (x, y, z * z + complex (x, y), n + 1)

set xrange [-0.5:0.5]
set yrange [-0.5:0.5]
set logscale z
set samples 200
set isosample 200
set pm3d map
set size square
a= #A#
b= #B#
splot mandel(-a/100,-b/100,complex(x,y),0) notitle
```

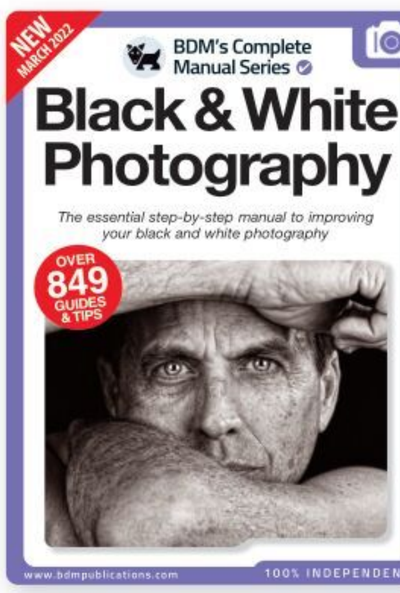
Read More

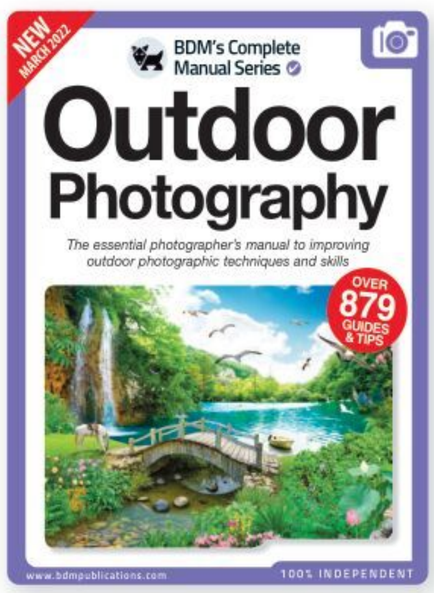
BDM's Complete Manual Series

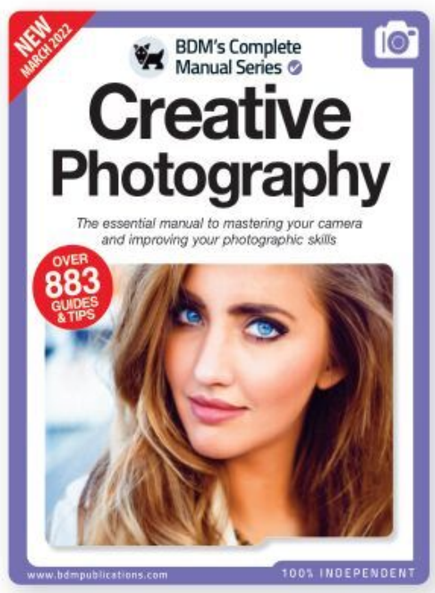


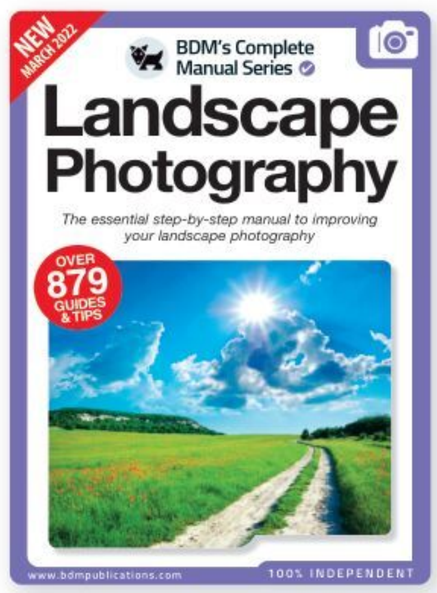


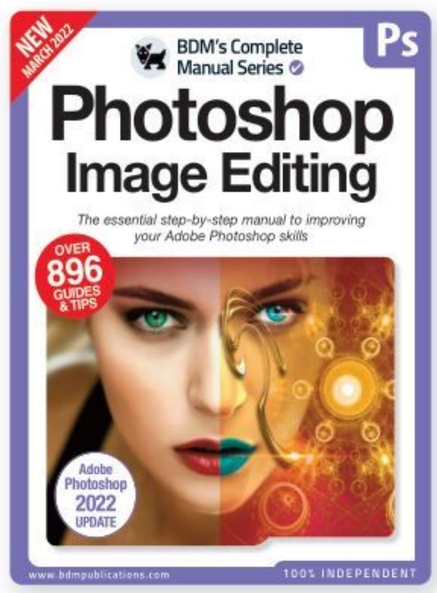


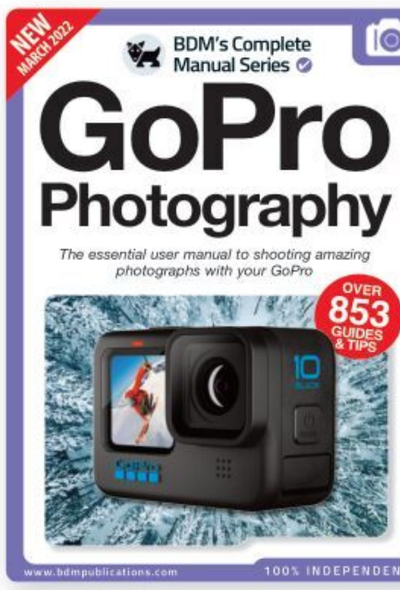


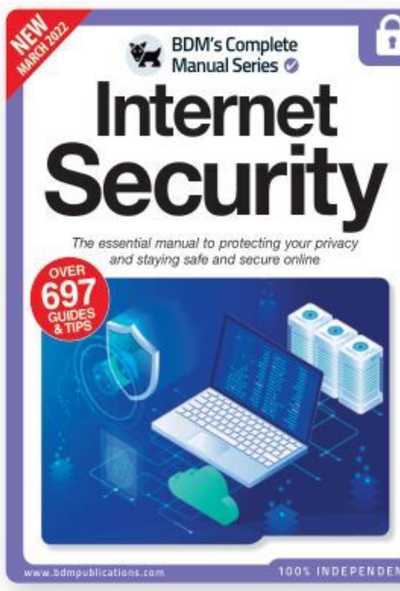


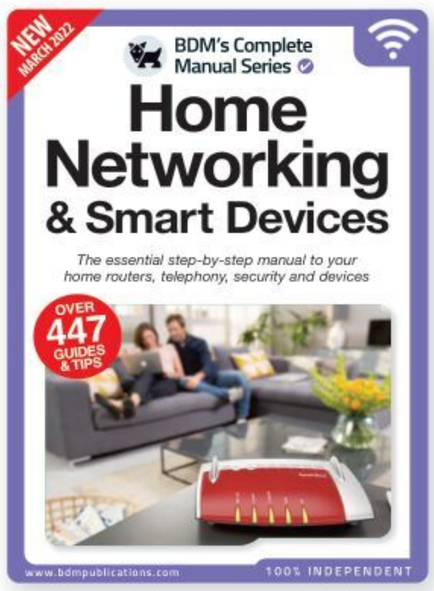


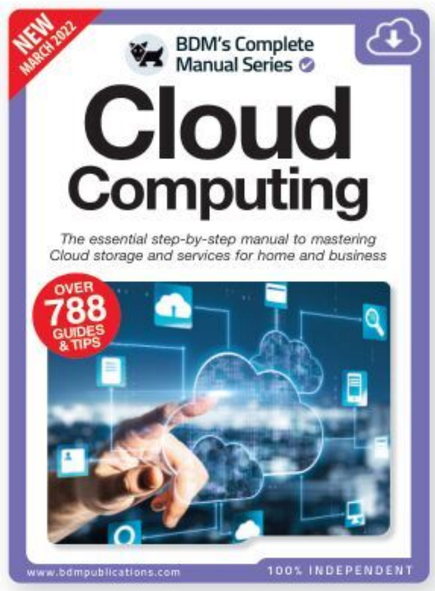


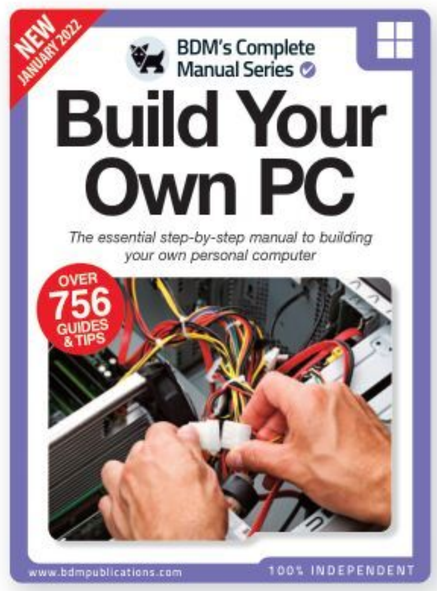


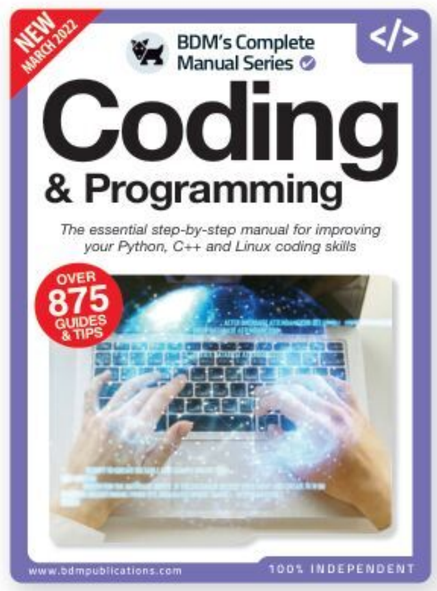




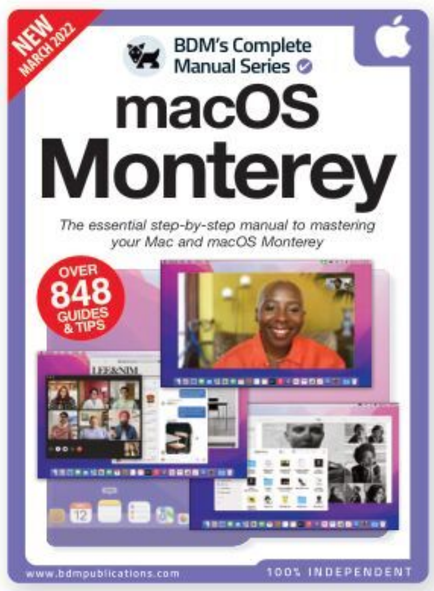


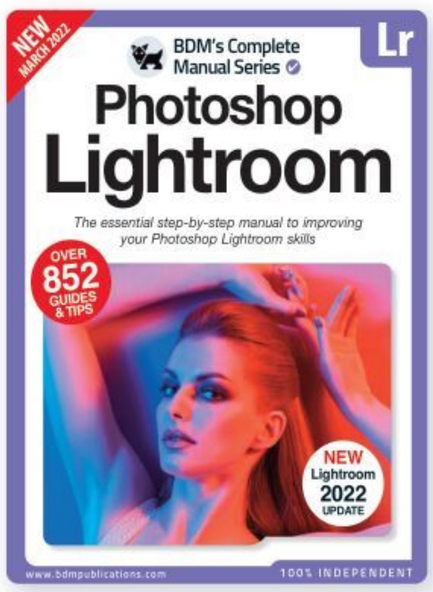


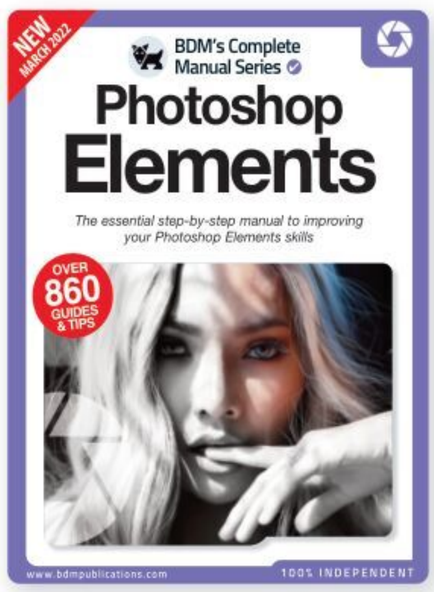


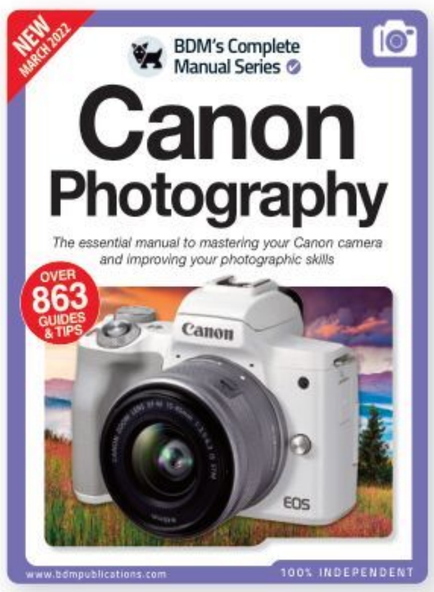


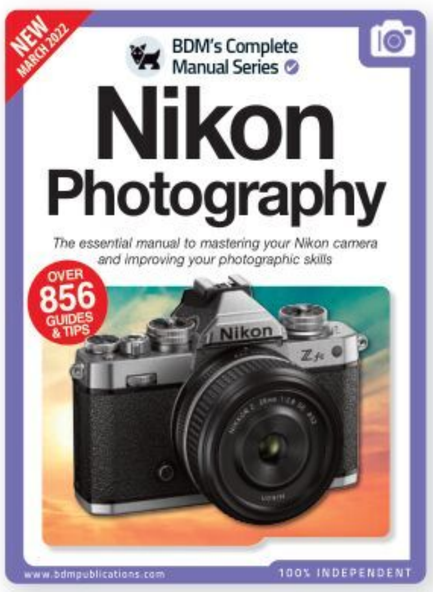
Available on Readly

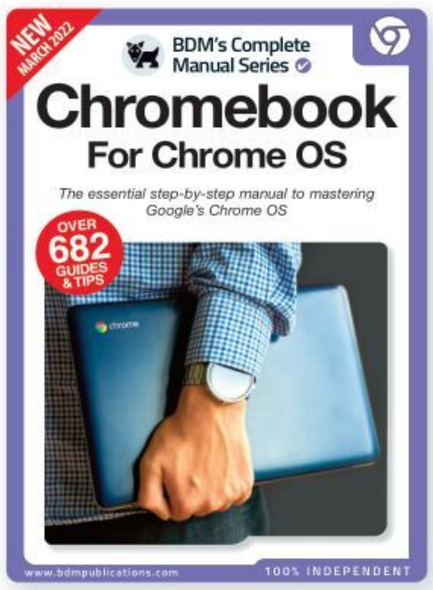


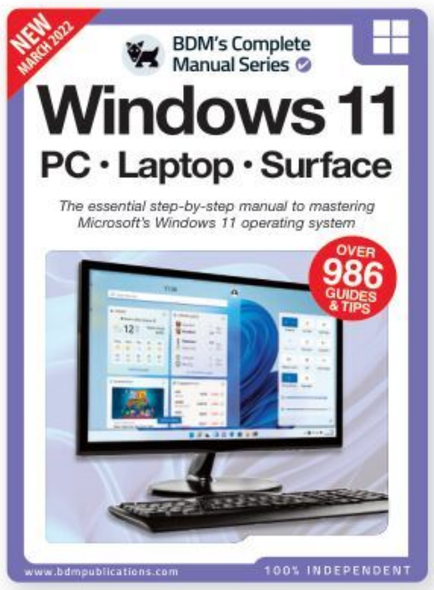


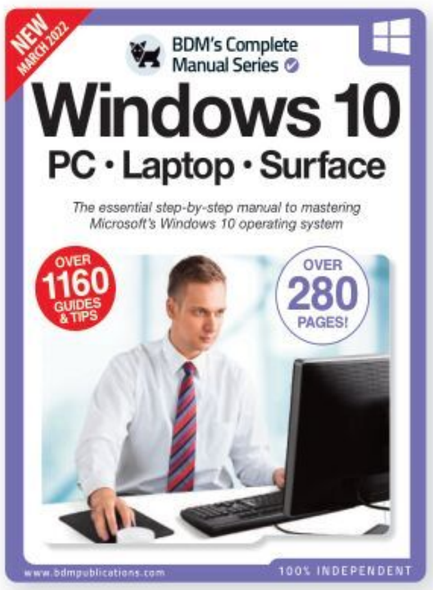


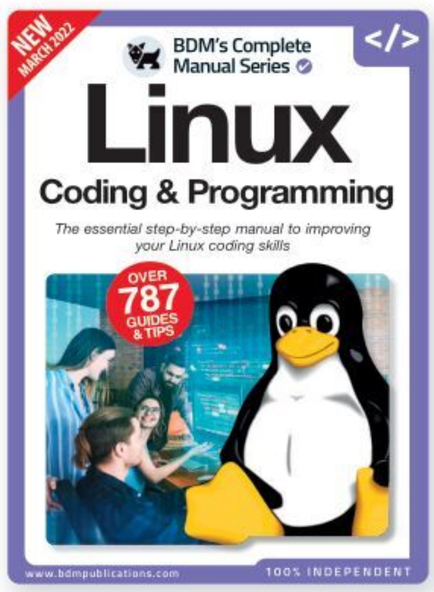


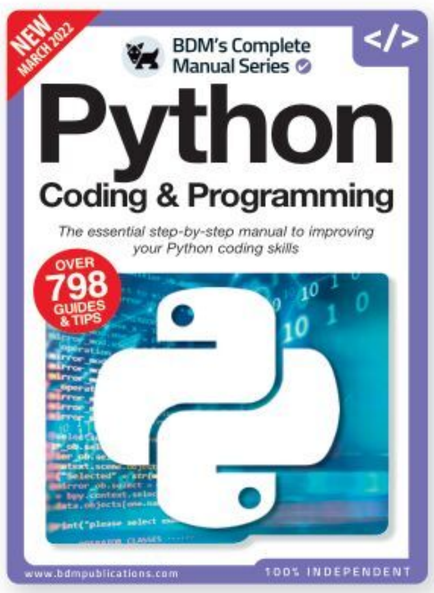


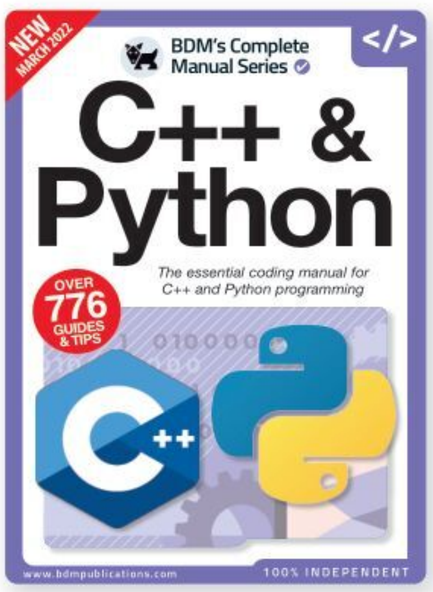












www.bdmpublications.com

Black Dog Media

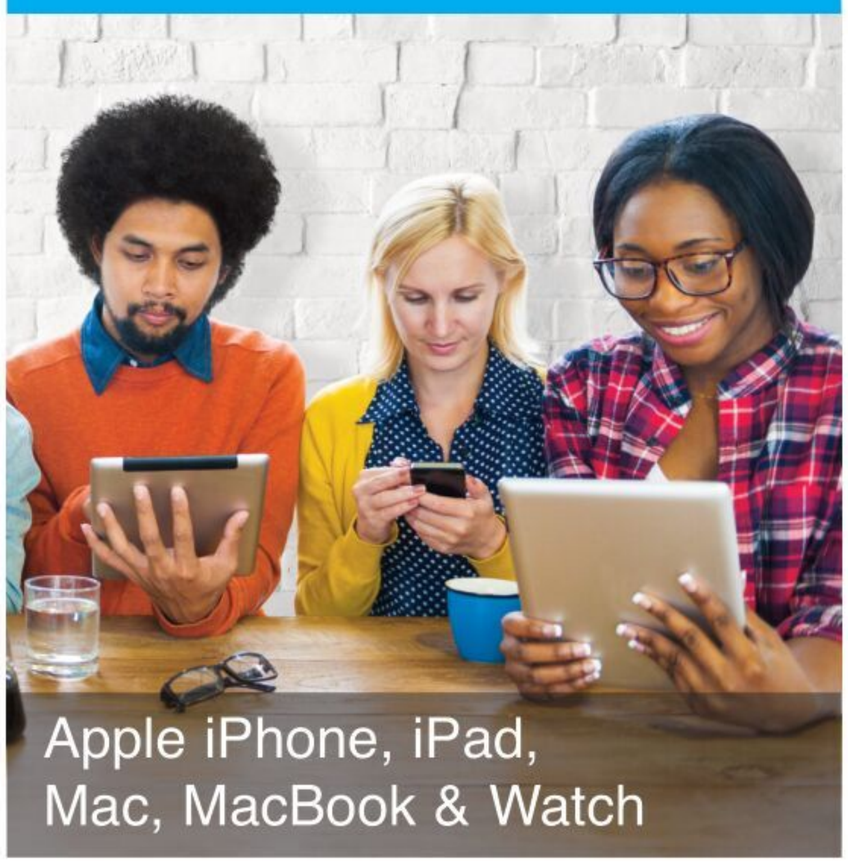
Master Your Tech

FROM BEGINNER TO EXPERT

To continue learning more about your tech visit us at:

www.bdmpublications.com

FREE Tech Guides



EXCLUSIVE OFFERS on our Tech Guidebooks

- Print & Digital Editions
- Featuring the Latest Updates
- Step-by-step Tutorials & Guides
- Created by BDM Experts

Check out our latest titles today!

PLUS

SPECIAL DEALS and Bonus Content

Sign up to our monthly newsletter and get the latest updates, offers and news from BDM. We are here to help you Master Your Tech!

bdmpublications.com/ultimate-photoshop

Buy our Photoshop guides and download tutorial images for free! *Simply sign up and get creative.*