Tricks & Tips

Coding and Programming

● Advanced guides and tutorials for coding Linux and Windows Batch Files

OVER 480 SECRETS & HACKS

● We share our awesome coding tips and projects to push your skills further

● Next level fixes and secrets help you get more from Linux distributions

Everything you need to master essential coding and programming skills

TRICKS & TIPS

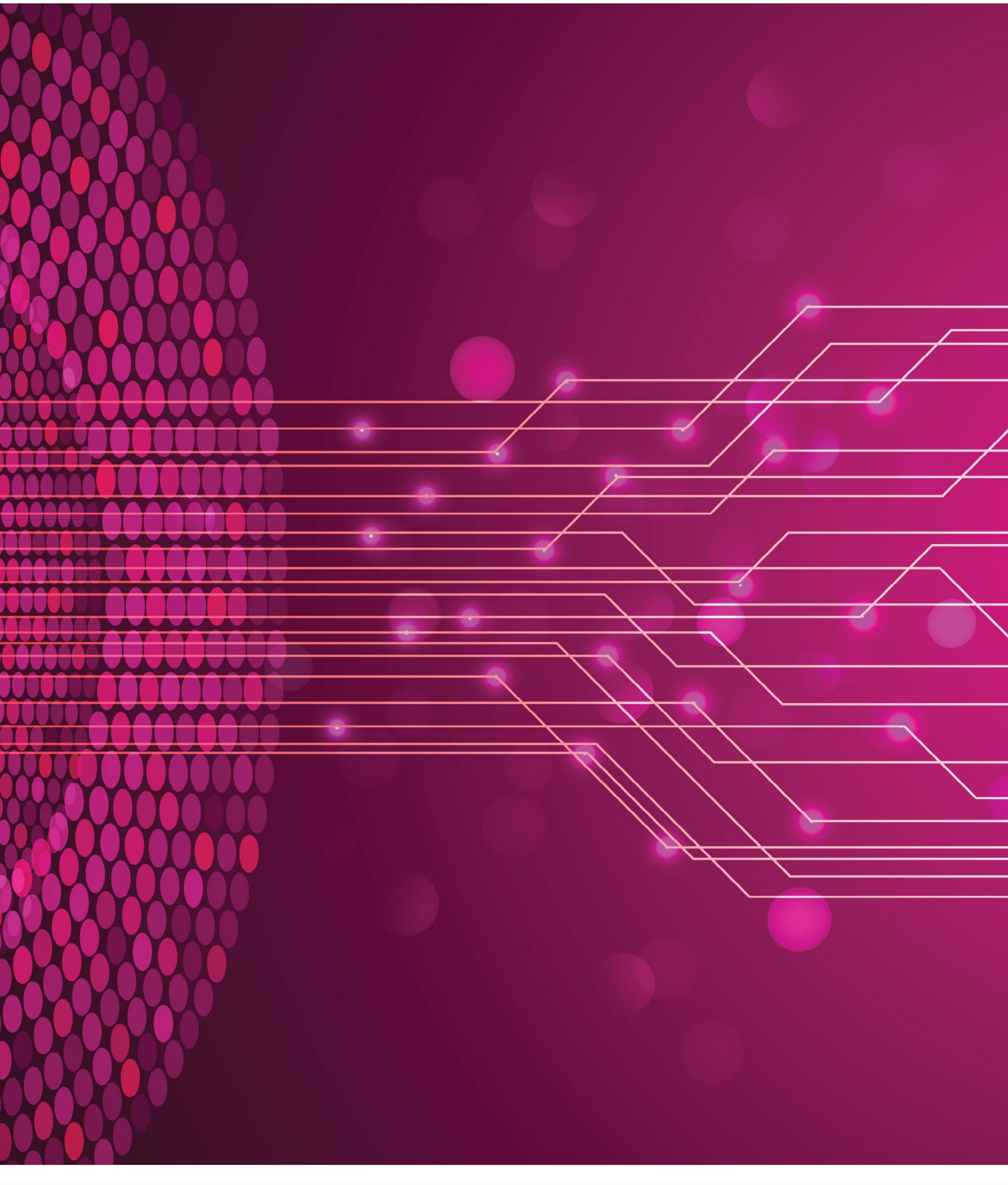
Coding and Programming

Coding and Programming Tricks & Tips is the perfect digital publication for the user that wants to take their skill set to the next level. Do you want to enhance your user experience? Or wish to gain insider knowledge? Do you want to learn directly from experts in their field? Learn the numerous short cuts that the professionals use? Over the pages of this new advanced user guide you will learn everything you will need to know to become a more confident, better skilled and experienced owner. A user that will make the absolute most of their software and ultimately their coding skills themselves. An achievement you can earn by simply enabling us to exclusively help and teach you the abilities we have gained over our decades of experience.

Over the page our journey continues, and we will be with you at every stage to advise, inform and ultimately inspire you to go further.

Contents

"Within these pages, you'll find the foundations and fundamentals for Python and C++, two of the most powerful languages in the world that are used by the likes of NASA, Microsoft, Apple and throughout the Internet. These languages are used in Big Data, Encryption, AI, gaming, science, engineering and advanced mathematics. From visualising black holes millions of light-years away to using technology to help solve the environmental crisis, Python and C++ are at the forefront of code."



Coding with Windows 10 Batch Files

Did you know that Windows has its own built-in scripting language? Batch files have been around since the early days of Windows, and while they are overshadowed by the might of Windows' modern graphical user interface, they are still there and still just as capable as they were thirty years ago.

Batch file programming is a skill that system administrators still use, so it's worth spending a bit of time learning how they work, and what you can do with them. This section introduces batch files, and covers user interactions, variables, loops and even a batch file quiz game to inject an element of fun.

What is a Batch File?

The Windows batch file has been around since the early days of DOS, and was once a critical element of actually being able to boot into a working system. There's a lot you can do with a batch file but let's just take a moment to see what one is.

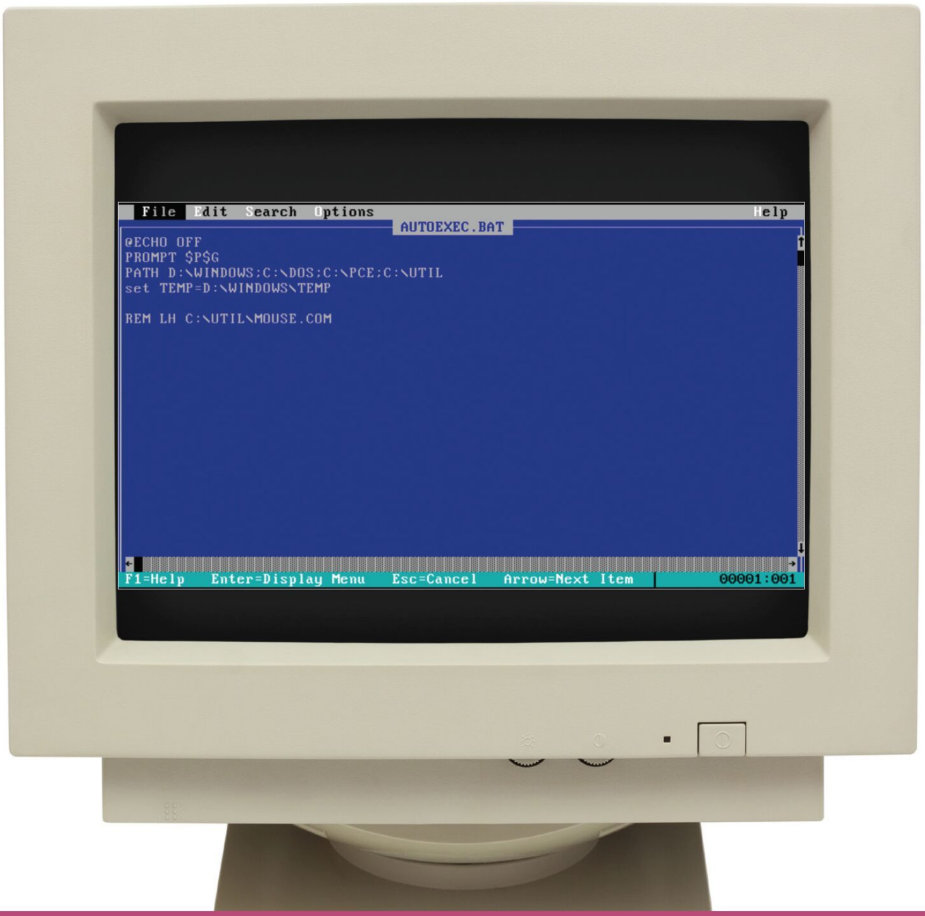
.BAT MAN

A Windows batch file is simply a script file that runs a series of commands, one line at a time, much in the same fashion as a Linux script. The series of commands are executed by the command line interpreter and stored in a plain text file with the .BAT extension; this signifies to Windows that it's an executable file, in this case, a script.

Batch files have been around since the earliest versions of Microsoft DOS. Although not exclusively a Microsoft scripting file, batch files are mainly associated with Microsoft's operating systems. In the early days, when a PC booted into a version of DOS (which produced a simple command prompt when powered up), the batch file was used in the form of a system file called Autoexec.bat. Autoexec.bat was a script that automatically executed (hence Autoexec) commands once the operating system had finished dealing with the Config.sys file.

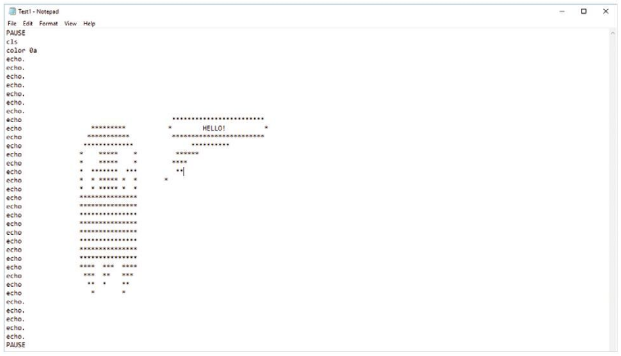
When a user powered up their DOS-based computer, and once the BIOS had finished checking the system memory and so on, DOS would look to the Config.sys file to load any specific display requirements and hardware drivers, allocate them a slot in the available memory, assign any memory managers and tell the system where the Command.com file, which is the command line interpreter for DOS, was. Once it had done that, then the Autoexec.bat file took over and ran through each line in turn, loading programs that would activate the mouse or optical drive into the memory areas assigned by the Config.sys file.

The DOS user of the day could opt to create different Autoexec.bat files depending on what they wanted to do. For example, if they wanted to play a game and have as much memory available as possible, they'd create a Config.sys and Autoexec.bat set of files that loaded the bare minimum of drivers and so on. If they needed access to the network, an Autoexec.bat file could be created to



```
@ECHO OFF
PROMPT $P$G
PATH D:\WINDOWS;C:\DOS;C:\PCE;C:\UTIL
set TEMP=D:\WINDOWS\TEMP

REM LH C:\UTIL\MOUSE.COM
```

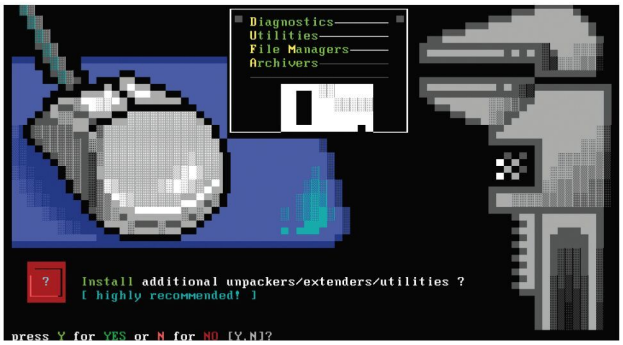
The Autoexec.bat file was a PC user's first experience with a batch file.



Test1 - Notepad

```
PAUSE
cls
color 0a
echo.
echo.
echo.
echo.
echo.
echo.
echo.
echo                                ************************
echo            *********          *       HELLO!         *
echo           ***********          ************************
echo          *************              **********
echo         *     *****     *       ******
echo         *     *****     *      ****
echo         *  *******  ***         **|
echo         *  * ***** *  *      *
echo         ***************
echo         ***************
echo         ***************
echo         ***************
echo         ***************
echo         ***************
echo         ***************
echo         ***************
echo         ****  ***  ****
echo          ***  **   ***
echo           **  *    **
echo            *       *
echo.
echo.
echo.
echo.
echo.
PAUSE
```

Batch files are plain text and often created using Notepad.



```
Install additional unpackers/extenders/utilities ?
[ highly recommended! ]

press Y for YES or N for NO [Y,N]?
```

Batch files were often used as utility programs, to help users with complex tasks.

load the network card driver and automatically gain access to the network. Each of these unique setups would be loaded on to a floppy disk and booted as and when required by the user.

The Autoexec.bat was the first such file many users came across in their PC-based computing lives; since many had come from a 16-bit or even 8-bit background; remember, this was the late eighties and early nineties. The batch file was the user's primary tool for automating tasks, creating shortcuts and adventure games and translating complex processes into something far simpler.

Nowadays however, a batch file isn't just for loading in drivers and such when the PC boots. You can use a batch file in the same way as any other scripting language file, in that you can program it to ask for user input and display the results on the screen; or save to a file and even send it to a locally or network attached printer. You can create scripts to back up your files to various locations, compare date stamps and only back up the most recently changed content as well as program the script to do all this automatically. Batch files are remarkably powerful and despite them not being as commonly used as they were during the older days of DOS, they are still there and can be utilised even in the latest version of Windows 10; and can be as complex or simple as you want them to be.

So what do you need to start batch file programming in Windows? Well, as long as you have Windows 10, or any older version of Windows for that matter, you can start batch file programming immediately. All you need is to be able to open Notepad and get to the command prompt of Windows. We show you how it all works, so read on.

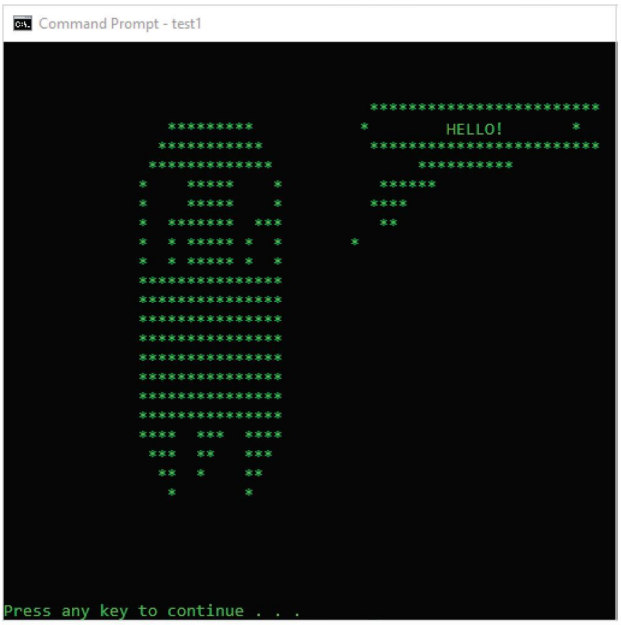
BATCH FILE POWER

Just like any other programming interface that can directly interrogate and manipulate the system, batch files require a certain amount of care when programming. It's hard to damage your system with a batch file, as the more important elements of the modern Windows system are protected by the User Account Control (UAC) security; UAC works by only allowing elevated privileges access to important system files. Therefore if you create a batch file that somehow deletes a system file, the UAC activates and stop the process.

However, if you're working in the command prompt with elevated privileges to begin with, as the Administrator, then the UAC won't question the batch file and continue regardless of what files are being deleted.

That said, you're not likely to create a batch file that intentionally wipes out your operating system. There are system controls in place to help prevent that; but it's worth mentioning as there are batch files available on the Internet that contain malicious code designed to create problems. Much like a virus, a rogue batch file (when executed with Administrator privileges) can cause much mayhem and system damage. In short, don't randomly execute any batch file downloaded from the Internet as an Administrator, without first reviewing what it does.

You can learn more about batch files in the coming pages, so don't worry too much about destroying your system with one. All this just demonstrates how powerful the humble batch file can be.



Command Prompt - test1

```
                              ************************
          *********          *       HELLO!         *
         ***********          ************************
        *************              **********
       *     *****     *       ******
       *     *****     *      ****
       *  *******  ***         **
       *  * ***** *  *      *
       ***************
       ***************
       ***************
       ***************
       ***************
       ***************
       ***************
       ***************
       ****  ***  ****
        ***  **   ***
         **  *    **
          *       *

Press any key to continue . . .
```

You can create complex batch files or simple ones that display ASCII images on screen.

Getting Started with Batch Files

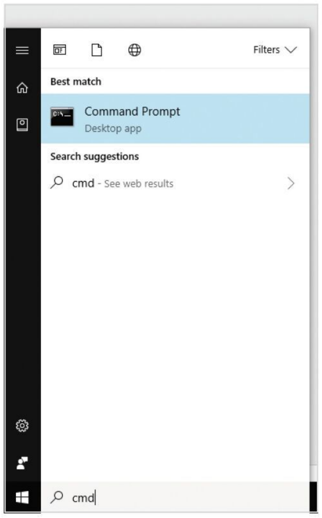
Before you begin to program with batch files, there are a few things you need to know. A batch file can only be executed once it has the .bat extension and editing one with Notepad isn't always straightforward.

A NEW BATCH

Throughout this section on batch files we're going to be working with Notepad, the command prompt and within a folder called 'Batch Files'. To begin with, let's see how you get to the Windows command prompt.

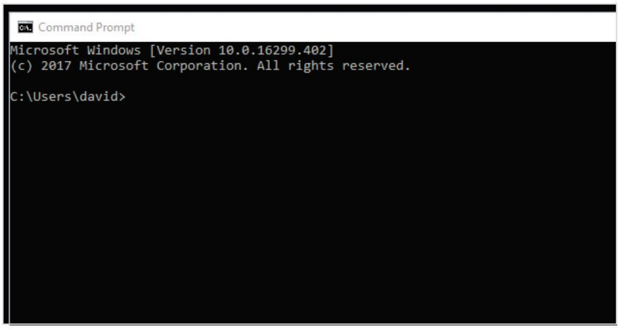
STEP 1

The Windows command prompt may look a little daunting to the newcomer but it's simply another interface (or Shell) used to access the filesystem. You can go anywhere you like in the command prompt, as you would with the graphical interface. To begin, click on the Windows Start button and enter CMD into the search box.



STEP 2

Click on the search result labelled Command Prompt (Desktop App) and a new window pops up. The Command Prompt window isn't much to look at to begin with but you can see the Microsoft Windows version number and copyright information followed by the prompt itself. The prompt details the current directory or folder you're in, together with your username.



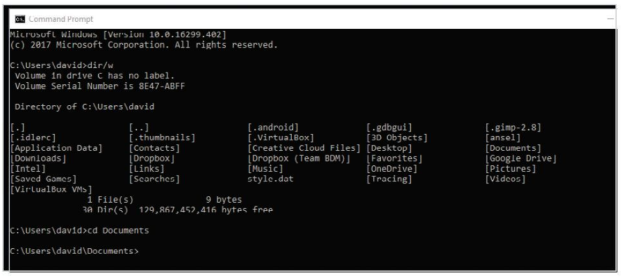
```
Command Prompt
Microsoft Windows [Version 10.0.16299.402]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\david>
```

STEP 3

While at the command prompt window, enter: `dir/w.` This lists all the files and directories from where you are at the moment in the system. In this case, that's your Home directory that Windows assigns every user that logs in. You can navigate by using the CD command (Change Directory). Try:

```
cd Documents
```

Then press Return.



```
Command Prompt
Microsoft Windows [Version 10.0.16299.402]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\david>dir/w
 Volume in drive C has no label.
 Volume Serial Number is 8E47-ABFF

 Directory of C:\Users\david

[.]                     [..]                    [.android]              [.gdbgui]               [.gimp-2.8]
[.idlerc]               [.thumbnails]           [.VirtualBox]           [3D Objects]            [ansel]
[Application Data]      [Contacts]              [Creative Cloud Files]  [Desktop]               [Documents]
[Downloads]             [Dropbox]               [Dropbox (Team BDM)]    [Favorites]             [Google Drive]
[Intel]                 [Links]                 [Music]                 [OneDrive]              [Pictures]
[Saved Games]           [Searches]              style.dat               [Tracing]               [Videos]
[VirtualBox VMs]
               1 File(s)              9 bytes
              30 Dir(s)  129,867,452,416 bytes free

C:\Users\david>cd Documents

C:\Users\david\Documents>
```

STEP 4

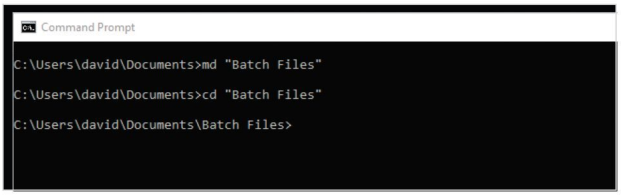
The prompt should change and display `\ Documents>;` this means you're in the Documents directory. Now, create a new directory call Batch Files. Enter:

```
md "Batch Files"
```

You need the quotations because without them, Windows creates two directories: Batch and Files. Now change directory into the newly created Batch Files.

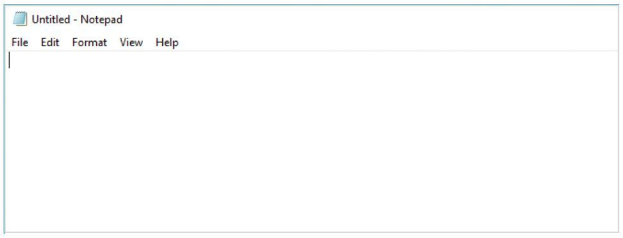
```
cd Batch Files
```

You won't need the quotes to change directories.



```
Command Prompt

C:\Users\david\Documents>md "Batch Files"

C:\Users\david\Documents>cd "Batch Files"

C:\Users\david\Documents\Batch Files>
```

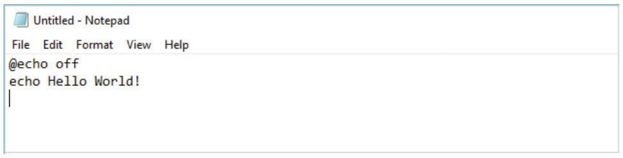
STEP 5 Now that you have the directory set up, where you store your batch files, here is how you can create one. Leave the command prompt window open and click on the Windows Start button again. This time enter Notepad and click on the search result to open the Notepad program. Notepad is a simple text editor but ideal for creating batch scripts with.



STEP 6 To create your first batch file, enter the following into Notepad:

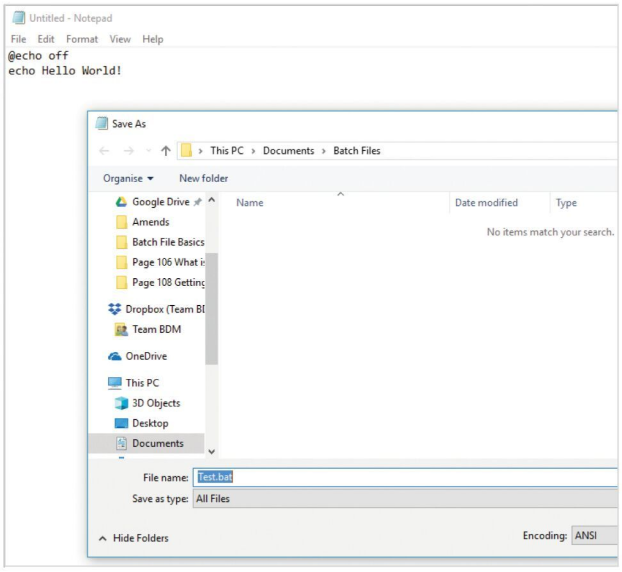
```
@echo off
echo Hello World!
```

By default, a batch file displays all the commands that it runs through, line by line. What the @echo off command does is turn that feature off for the whole script; with the '@' (at) sign to apply that command to itself.



```
@echo off
echo Hello World!
```

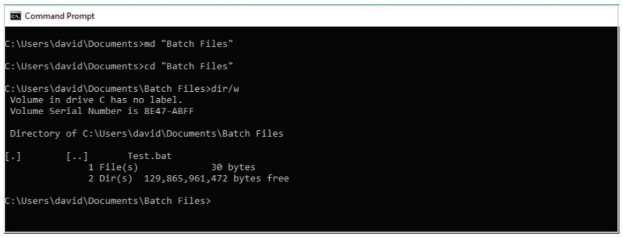
STEP 7 When saving anything in Notepad the default extension is .txt, to denote a text file. However, you want the extension to be .bat. Click on File > Save As and navigate to the newly created Batch Files directory in Documents. Click the drop-down menu Save as Type, and select All Files from the menu. In File Name, call the file **Test.bat**.



```
@echo off
echo Hello World!
```

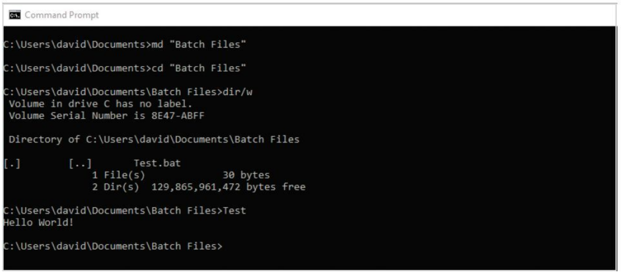
File name: Test.bat
Save as type: All Files
Encoding: ANSI

STEP 8 Back at the command prompt window, enter: `dir/w` again to list the newly created Test.bat file. By the way, the /w part of dir/w means the files are listed across the screen as opposed to straight down. Enter `dir` if you want (although you need more files to view) but it's considered easier to read with the /w flag.



```
C:\Users\david\Documents>md "Batch Files"

C:\Users\david\Documents>cd "Batch Files"

C:\Users\david\Documents\Batch Files>dir/w
 Volume in drive C has no label.
 Volume Serial Number is 8E47-ABFF

 Directory of C:\Users\david\Documents\Batch Files

[.]          [..]         Test.bat
               1 File(s)             30 bytes
               2 Dir(s)  129,865,961,472 bytes free

C:\Users\david\Documents\Batch Files>
```

STEP 9 To execute the batch file you've just created, simply enter its name, Test, in the command prompt window. You don't need to add the .bat part, as Windows recognises it as an executable file, and the only one with that particular name in the current directory. Press return and see how you're greeted with Hello World! in the command prompt.

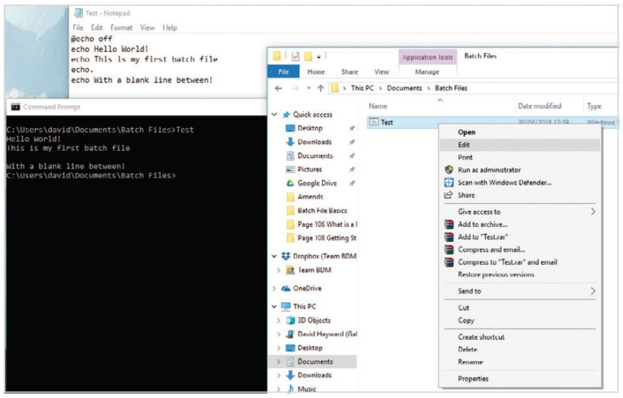


```
C:\Users\david\Documents>md "Batch Files"

C:\Users\david\Documents>cd "Batch Files"

C:\Users\david\Documents\Batch Files>dir/w
 Volume in drive C has no label.
 Volume Serial Number is 8E47-ABFF

 Directory of C:\Users\david\Documents\Batch Files

[.]          [..]         Test.bat
               1 File(s)             30 bytes
               2 Dir(s)  129,865,961,472 bytes free

C:\Users\david\Documents\Batch Files>Test
Hello World!

C:\Users\david\Documents\Batch Files>
```

STEP 10 The echo command displays whatever is after it to the screen. Right-click the Test.bat file from Windows Explorer and select Edit to add more echo commands if you like. Try this:

```
@echo off
echo Hello World!
echo This is my first batch file
echo.
echo With a blank line between!
```

Remember to save each new change to the batch file.



```
@echo off
echo Hello World!
echo This is my first batch file
echo.
echo With a blank line between!
```

```
C:\Users\david\Documents\Batch Files>Test
Hello World!
This is my first batch file

With a blank line between!
C:\Users\david\Documents\Batch Files>
```

Getting an Output

While it's great having the command prompt window display what you're putting after the echo command in the batch file, it's not very useful at the moment, or interactive for that matter. Let's change up a gear and get some output.

INPUT, OUTPUT

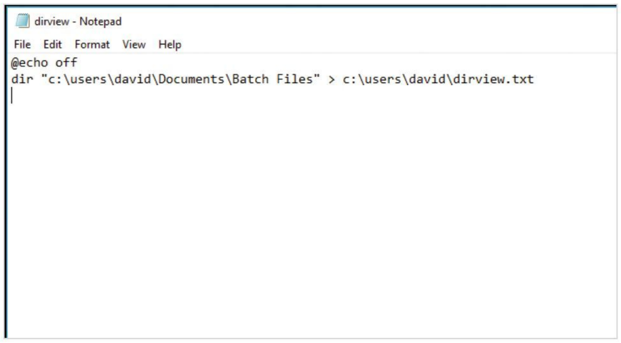
Batch files are capable of taking a normal Windows command and executing it, while also adding extra options and flags in to the equation.

STEP 1

Let's keep things simple to begin with. Create a new batch file called 'dirview.bat', short for Directory View. Start with the @echo off command and under that add:

```
dir "c:\users\YOURNAME\Documents\Batch Files" >
c:\users\YOURNAME\dirview.txt
```

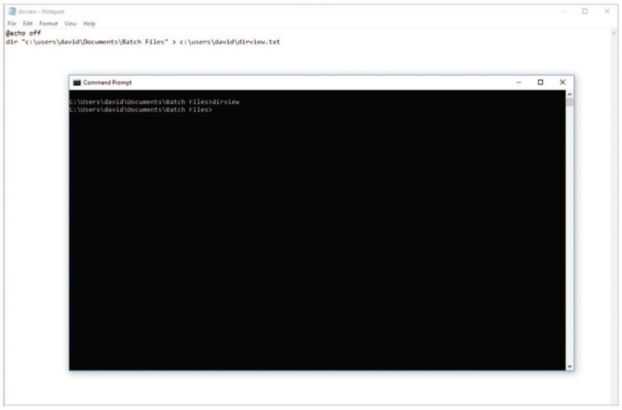
Substitute YOURNAME with your Windows username.



```
@echo off
dir "c:\users\david\Documents\Batch Files" > c:\users\david\dirview.txt
```

STEP 2

The new line uses the dir command to list the contents of the directory Batch Files, in your Home directory, dumping the output to a text file called `dirview.txt` in the root of your Home directory. This is done, so that the Windows UAC doesn't require elevated permissions, as everything is in your own Home area. Save and run the batch file.

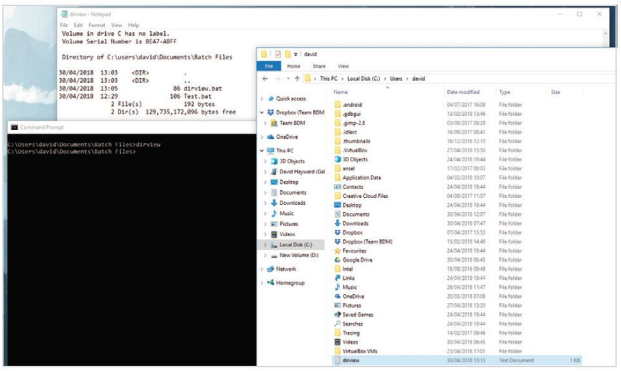


```
@echo off
dir "c:\users\david\Documents\Batch Files" > c:\users\david\dirview.txt
```

```
C:\Users\david\Documents\Batch Files>dirview
C:\Users\david\Documents\Batch Files>
```

STEP 3

You have no doubt noticed that there is no indication that the batch file worked as there's no meaningful output on the screen. However, if you now open Explorer and browse to `c:\Users\YOURNAME`, remembering to substitute `YOURNAME` with your Windows username, and double-click the `dirview.txt` file, you can see the batch file's output.



```
 Volume in drive C has no label.
 Volume Serial Number is 8E47-ABFF

 Directory of C:\users\david\Documents\Batch Files

30/04/2018  13:03    <DIR>          .
30/04/2018  13:03    <DIR>          ..
30/04/2018  13:05                86 dirview.bat
30/04/2018  12:29               106 Test.bat
               2 File(s)            192 bytes
               2 Dir(s)  129,735,172,096 bytes free
```

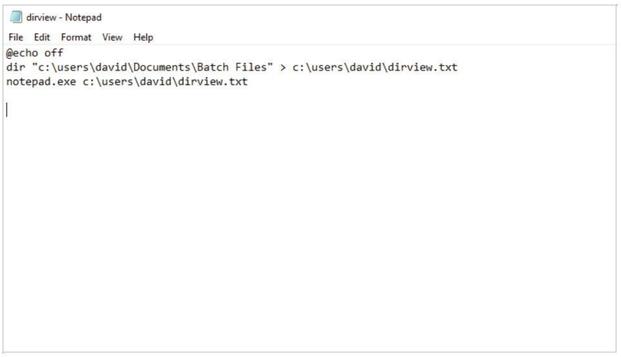
```
C:\Users\david\Documents\Batch Files>dirview
C:\Users\david\Documents\Batch Files>
```

STEP 4

If you want to automate the task of opening the text file that contains the output, add the following line to the batch file:

```
notepad.exe c:\users\YOURNAME\dirview.txt
```

Save the file and once again run from the command prompt. This time, it creates the output and automatically opens Notepad with the output contents.



```
@echo off
dir "c:\users\david\Documents\Batch Files" > c:\users\david\dirview.txt
notepad.exe c:\users\david\dirview.txt
```

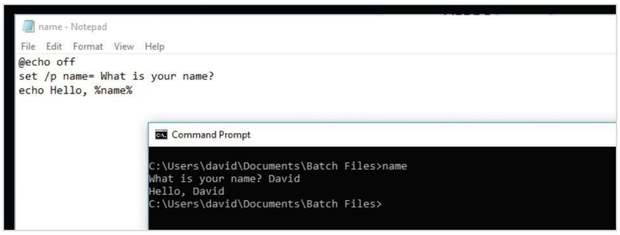
OUTPUT WITH VARIABLES

Variables offer a more interesting way of outputting something to the screen and create a higher level of interaction between the user and the batch file. Try this example below.

STEP 1 Create a new batch file and call it **name.bat**. Start with the `@echo off` command, then add the following lines:

```
set /p name= What is your name?
echo Hello, %name%
```

Note: there's a space after the question mark. This is to make it look neater on the screen. Save it and run the batch file.



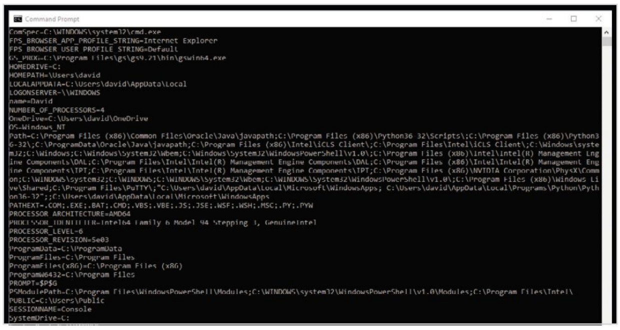
```
@echo off
set /p name= What is your name?
echo Hello, %name%
```

```
C:\Users\david\Documents\Batch Files>name
What is your name? David
Hello, David
C:\Users\david\Documents\Batch Files>
```

STEP 2 The set **/p** name creates a variable called name, with the **/p** part indicating that an '**=prompt string**' is to follow. The Set command displays, sets or removes system and environmental variables. For example, while in the command prompt window enter:

```
set
```

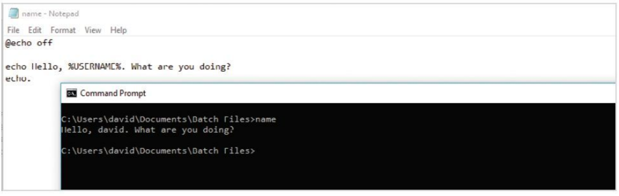
To view the current system variables. Note the name= variable we just created.



```
ComSpec=C:\WINDOWS\system32\cmd.exe
FPS_BROWSER_APP_PROFILE_STRING=Internet Explorer
FPS_BROWSER_USER_PROFILE_STRING=Default
GS_PROG=C:\Program Files\gs\gs9.21\bin\gswin64.exe
HOMEDRIVE=C:
HOMEPATH=\Users\david
LOCALAPPDATA=C:\Users\david\AppData\Local
LOGONSERVER=\\WINDOWS
name=David
NUMBER_OF_PROCESSORS=4
OneDrive=C:\Users\david\OneDrive
OS=Windows_NT
Path=C:\Program Files (x86)\Common Files\Oracle\Java\javapath;C:\Program Files (x86)\Python36-32\Scripts\;C:\Program Files (x86)\Python3
6-32\;C:\ProgramData\Oracle\Java\javapath;C:\Program Files (x86)\Intel\iCLS Client\;C:\Program Files\Intel\iCLS Client\;C:\Windows\syste
m32;C:\Windows;C:\Windows\System32\Wbem;C:\Windows\System32\WindowsPowerShell\v1.0\;C:\Program Files (x86)\Intel\Intel(R) Management Eng
ine Components\DAL;C:\Program Files\Intel\Intel(R) Management Engine Components\DAL;C:\Program Files (x86)\Intel\Intel(R) Management Eng
ine Components\IPT;C:\Program Files\Intel\Intel(R) Management Engine Components\IPT;C:\Program Files (x86)\NVIDIA Corporation\PhysX\Comm
on;C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;C:\WINDOWS\System32\WindowsPowerShell\v1.0\;C:\Program Files (x86)\Windows Li
ve\Shared;C:\Program Files\PuTTY\;"C:\Users\david\AppData\Local\Microsoft\WindowsApps; C:\Users\david\AppData\Local\Programs\Python\Pyth
on36-32";;C:\Users\david\AppData\Local\Microsoft\WindowsApps
PATHEXT=.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC;.PY;.PYW
PROCESSOR_ARCHITECTURE=AMD64
PROCESSOR_IDENTIFIER=Intel64 Family 6 Model 94 Stepping 3, GenuineIntel
PROCESSOR_LEVEL=6
PROCESSOR_REVISION=5e03
ProgramData=C:\ProgramData
ProgramFiles=C:\Program Files
ProgramFiles(x86)=C:\Program Files (x86)
ProgramW6432=C:\Program Files
PROMPT=$P$G
PSModulePath=C:\Program Files\WindowsPowerShell\Modules;C:\WINDOWS\system32\WindowsPowerShell\v1.0\Modules;C:\Program Files\Intel\
PUBLIC=C:\Users\Public
SESSIONNAME=Console
SystemDrive=C:
```

STEP 3 Variables stored with Set can be called with the **%VARIABLENAME% syntax**. In the batch file, we used the newly created %name% syntax to call upon the contents of the variable called name. Your username, for example, is stored as a variable. Try this in a batch file:

```
echo Hello, %USERNAME%. What are you doing?
```

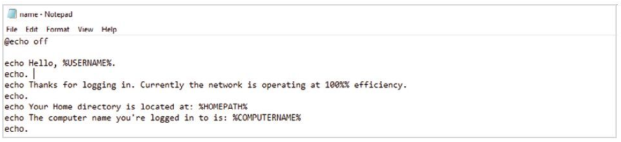


```
@echo off

echo Hello, %USERNAME%. What are you doing?
echo.
```

```
C:\Users\david\Documents\Batch Files>name
Hello, david. What are you doing?

C:\Users\david\Documents\Batch Files>
```

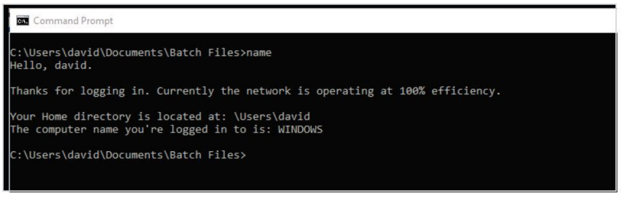
STEP 4 This is extremely useful if you want to create a unique, personal batch file that automatically runs when a user logs into Windows. Using the default systems variables that Windows itself creates, you can make a batch file that greets each user:

```
@echo off

echo Hello, %USERNAME%.
echo.
echo Thanks for logging in. Currently the network
is operating at 100%% efficiency.
echo.
echo Your Home directory is located at: %HOMEPATH%
echo The computer name you're logged in to is:
%COMPUTERNAME%
echo.
```



```
@echo off

echo Hello, %USERNAME%.
echo.
echo Thanks for logging in. Currently the network is operating at 100%% efficiency.
echo.
echo Your Home directory is located at: %HOMEPATH%
echo The computer name you're logged in to is: %COMPUTERNAME%
echo.
```

STEP 5 Save and execute the batch file changes; you can overwrite and still use name.bat if you want. The batch file takes the current system variables and reports them accordingly, depending on the user's login name and the name of the computer. Note: the double percent symbol means the percent sign will be displayed, and is not a variable.



```
C:\Users\david\Documents\Batch Files>name
Hello, david.

Thanks for logging in. Currently the network is operating at 100% efficiency.

Your Home directory is located at: \Users\david
The computer name you're logged in to is: WINDOWS

C:\Users\david\Documents\Batch Files>
```

STEP 6 Alternatively, you can run the batch file and display it on the user's desktop as a text file:

```
@echo off

echo Hello, %USERNAME%. > c:%HOMEPATH%\user.txt
echo. >> c:%HOMEPATH%\user.txt
echo Thanks for logging in. Currently the network
is operating at 100%% efficiency. >> c:%HOMEPATH%\
user.txt
echo. >> c:%HOMEPATH%\user.txt
echo Your Home directory is located at: %HOMEPATH%
>> c:%HOMEPATH%\user.txt
echo The computer name you're logged in to is:
%COMPUTERNAME% >> c:%HOMEPATH%\user.txt
echo. >> c:%HOMEPATH%\user.txt

notepad c:%HOMEPATH%\user.txt
```

The > outputs to a new file called user.txt, while the >> adds the lines within the file.

Playing with Variables

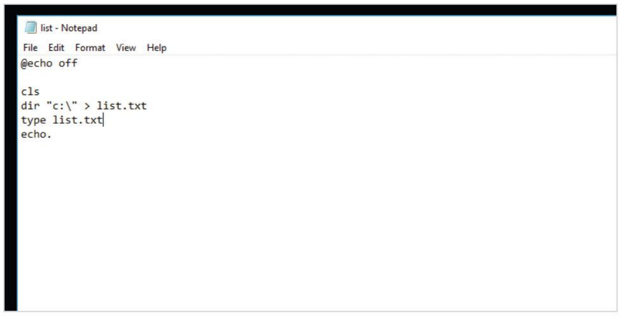
There's a lot you can accomplish with both the system and environmental variables, alongside your own. Mixing the two can make for a powerful and extremely useful batch file and when combined with other commands, the effect is really impressive.

USING MORE VARIABLES

Here's a good example of mixing system and environmental variables with some of your own creation, along with a number of external Windows commands.

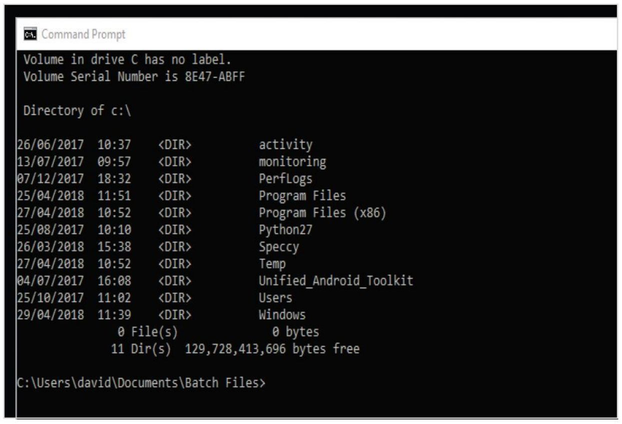
STEP 1 Create a new batch file called list.bat and start it off with the `@echo off` command. Begin by clearing the command prompt screen and displaying a list of the current directories on the computer:

```
cls
dir "c:\" > list.txt
type list.txt
echo.
```



```
list - Notepad
File  Edit  Format  View  Help
@echo off

cls
dir "c:\" > list.txt
type list.txt
echo.
```

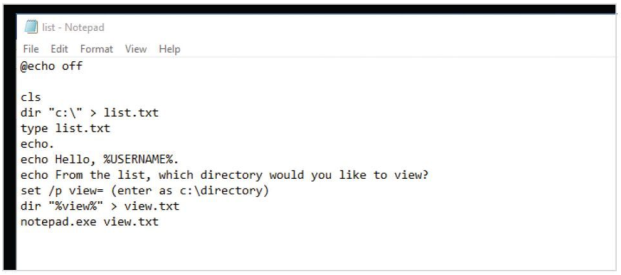
STEP 2 Save and execute the batch file. Within the command prompt you can see the contents of all the files and directories from the root of the C:\ drive; and as any user under Windows has permission to see this, there's no UAC elevated privileges required.



```
Command Prompt
 Volume in drive C has no label.
 Volume Serial Number is 8E47-ABFF

 Directory of c:\

26/06/2017  10:37    <DIR>          activity
13/07/2017  09:57    <DIR>          monitoring
07/12/2017  18:32    <DIR>          PerfLogs
25/04/2018  11:51    <DIR>          Program Files
27/04/2018  10:52    <DIR>          Program Files (x86)
25/08/2017  10:10    <DIR>          Python27
26/03/2018  15:38    <DIR>          Speccy
27/04/2018  10:52    <DIR>          Temp
04/07/2017  16:08    <DIR>          Unified_Android_Toolkit
25/10/2017  11:02    <DIR>          Users
29/04/2018  11:39    <DIR>          Windows
               0 File(s)              0 bytes
              11 Dir(s)  129,728,413,696 bytes free

C:\Users\david\Documents\Batch Files>
```

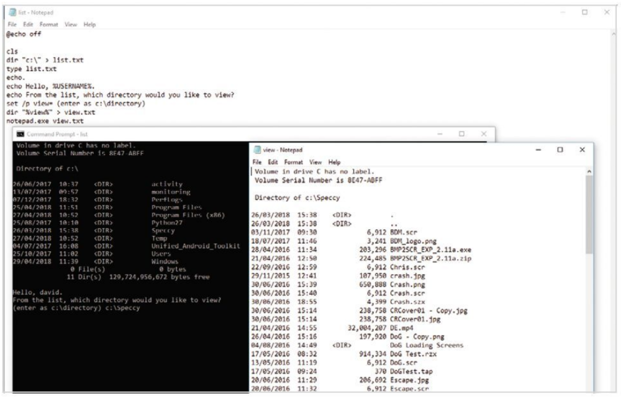
STEP 3 Now, create a batch file that displays the contents of any directory and post it as a text file to the user's screen. Add the following to the list.bat batch file:

```
echo Hello, %USERNAME%.
echo From the list, which folder would you like to
view?
set /p view= (enter as c:\folder)
dir "%view%" > view.txt
notepad.exe view.txt
```



```
list - Notepad
File  Edit  Format  View  Help
@echo off

cls
dir "c:\" > list.txt
type list.txt
echo.
echo Hello, %USERNAME%.
echo From the list, which directory would you like to view?
set /p view= (enter as c:\directory)
dir "%view%" > view.txt
notepad.exe view.txt
```

STEP 4 What's happening here is the batch file asks the user to enter any of the directories displayed in the list it generated, in the form of '**c:\directory**'. Providing the user enters a valid directory, its contents are displayed as a text file. We created the view variable here along with %HOMEPATH%, to store the input and the text file.



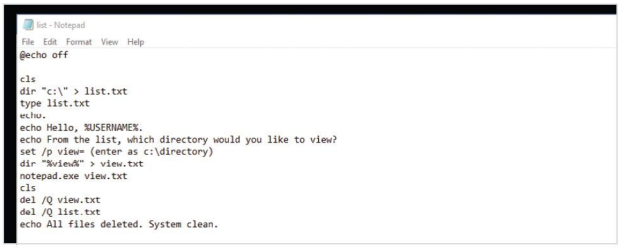
```
list - Notepad
File  Edit  Format  View  Help
@echo off

cls
dir "c:\" > list.txt
type list.txt
echo.
echo Hello, %USERNAME%.
echo From the list, which directory would you like to view?
set /p view= (enter as c:\directory)
dir "%view%" > view.txt
notepad.exe view.txt
```

```
Command Prompt - list
 Volume in drive C has no label.
 Volume Serial Number is 8E47-ABFF

 Directory of c:\

26/06/2017  10:37    <DIR>          activity
13/07/2017  09:57    <DIR>          monitoring
07/12/2017  18:32    <DIR>          PerfLogs
25/04/2018  11:51    <DIR>          Program Files
27/04/2018  10:52    <DIR>          Program Files (x86)
25/08/2017  10:10    <DIR>          Python27
26/03/2018  15:38    <DIR>          Speccy
27/04/2018  10:52    <DIR>          Temp
04/07/2017  16:08    <DIR>          Unified_Android_Toolkit
25/10/2017  11:02    <DIR>          Users
29/04/2018  11:39    <DIR>          Windows
               0 File(s)              0 bytes
              11 Dir(s)  129,724,956,672 bytes free

Hello, david.
From the list, which directory would you like to view?
(enter as c:\directory) c:\Speccy
```

```
view - Notepad
File  Edit  Format  View  Help
 Volume in drive C has no label.
 Volume Serial Number is 8E47-ABFF

 Directory of c:\Speccy

26/03/2018  15:38    <DIR>          .
26/03/2018  15:38    <DIR>          ..
03/11/2017  09:30             6,912 BDM.scr
18/07/2017  11:46             3,241 BDM_logo.png
28/04/2016  11:34           203,296 BMP2SCR_EXP_2.11a.exe
21/04/2016  12:50           224,485 BMP2SCR_EXP_2.11a.zip
22/09/2016  12:59             6,912 Chris.scr
29/11/2015  12:41           107,950 crash.jpg
30/06/2016  15:39           650,888 Crash.png
30/06/2016  15:40             6,912 Crash.scr
30/06/2016  18:55             4,399 Crash.szx
30/06/2016  15:14           238,758 CRCover01 - Copy.jpg
30/06/2016  15:14           238,758 CRCover01.jpg
21/04/2016  14:55        32,004,207 DE.mp4
26/04/2016  15:16           197,920 DoG - Copy.png
04/08/2016  14:49    <DIR>          DoG Loading Screens
17/05/2016  08:32           914,334 DoG Test.rzx
13/05/2016  11:19             6,912 DoG.scr
17/05/2016  09:24               370 DoGTest.tap
20/06/2016  11:29           206,692 Escape.jpg
20/06/2016  11:32             6,912 Escape.scr
```

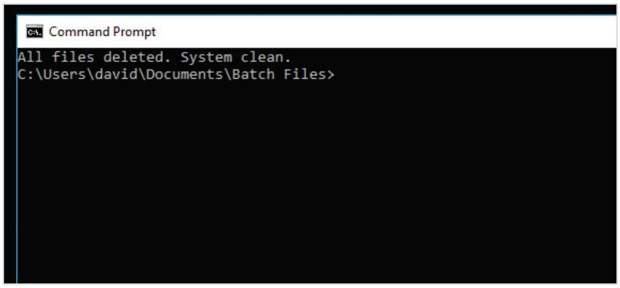
STEP 5 It's always a good idea, when creating text files for the user to temporarily view, to clean up after yourself. There's nothing worse than having countless, random text files cluttering up the file system. That being the case, let's clear up with:

```
cls
del /Q view.txt
del /Q list.txt
echo All files deleted. System clean.
```



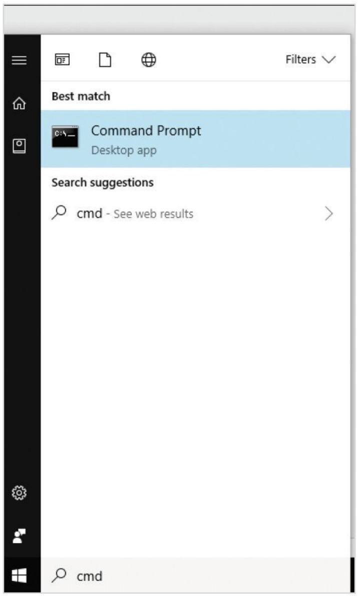
```
list - Notepad
File  Edit  Format  View  Help
@echo off

cls
dir "c:\" > list.txt
type list.txt
echo.
echo Hello, %USERNAME%.
echo From the list, which directory would you like to view?
set /p view= (enter as c:\directory)
dir "%view%" > view.txt
notepad.exe view.txt
cls
del /Q view.txt
del /Q list.txt
echo All files deleted. System clean.
```

STEP 6 The additions to the batch file simply clear the command prompt window (using the cls command) and delete both the view.txt and list.txt files that were created by the batch file. The /Q flag in the del command means it deletes the files without any user input or notification. The final message informs the user that the files are removed.

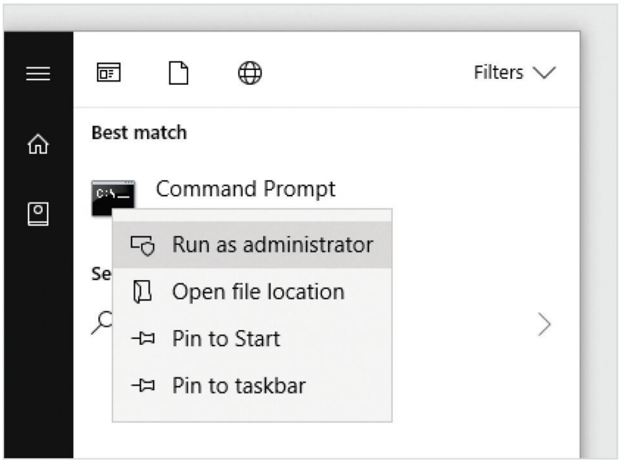


```
Command Prompt
All files deleted. System clean.
C:\Users\david\Documents\Batch Files>
```

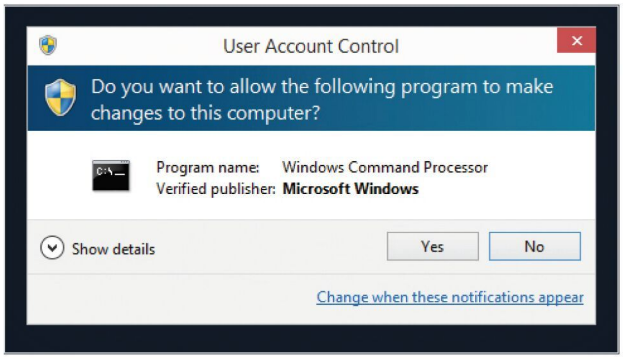
STEP 7 Depending on how your system is configured, you may not get any directory information at all or a message stating Access Denied. This is because the UAC is blocking access to protected areas of the system, like c:\Windows or C:\Program Files. Therefore, you need to run the batch file as an Administrator. Click the Windows Start button and enter **CMD** again.



STEP 8 Instead of left clicking on the Command Prompt result, as you did the first time you opened it, right-click it and from the menu choose Run as Administrator. There is a risk that you could damage system files as the Administrator but as long as you're careful and don't do anything beyond viewing directories, you will be okay.



STEP 9 This action triggers the UAC warning message, asking you if you're sure you want to run the Windows command prompt with the elevated Administrator privileges. Most of the time we wouldn't recommend this course of action: the UAC is there to protect your system. In this case, however, click Yes.

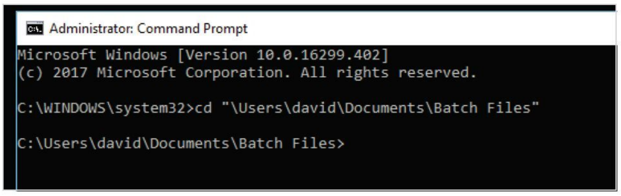


User Account Control

Do you want to allow the following program to make changes to this computer?

Program name: Windows Command Processor
Verified publisher: **Microsoft Windows**

STEP 10 With the UAC active, the command prompt looks a little different. For starters, it's now defaulting to the C:\WINDOWS\system32 folder and the top of the windows is labelled Administrator. To run the batch file, you need to navigate to the Batch Files directory with: **cd \Users\USERNAME\Documents\Batch Files**. To help, press the Tab key to auto-complete the directory names.



```
Administrator: Command Prompt
Microsoft Windows [Version 10.0.16299.402]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>cd "\Users\david\Documents\Batch Files"

C:\Users\david\Documents\Batch Files>
```

Batch File Programming

It's the little additions we can make to a batch file that help it stand out and ultimately become more useful. While the Windows graphical interface is still king, the command line can do just as much, and this is where batch files come into their own.

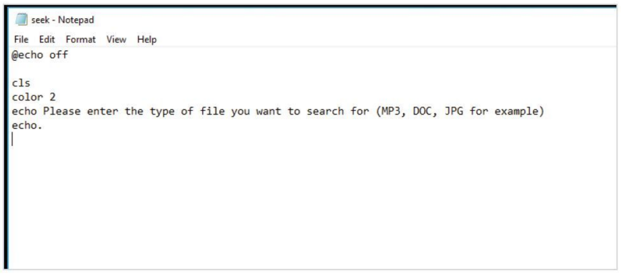
SEARCHING FOR FILES

Here's an interesting little batch file that you can easily extend for your own use. It asks the user for a file type to search for and displays the results.

STEP 1

We are introducing a couple of new commands into the mix here but we think they're really useful. Create a new batch file called seek.bat and in it put:

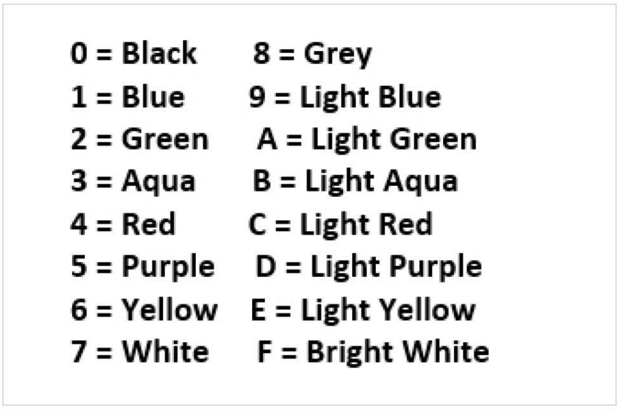
```
@echo off
cls
color 2
echo Please enter the type of file you want to
search for (MP3, DOC, JPG for example)
echo.
```



```
seek - Notepad
File  Edit  Format  View  Help
@echo off

cls
color 2
echo Please enter the type of file you want to search for (MP3, DOC, JPG for example)
echo.
```

STEP 2

The new command in this instance is color (Americanised spelling). Color, as you already assume, changes the colour of the command prompt display. The color attributes are specified by two hex digits, the first corresponds to the background colour of the Command console and the second to the foreground, and can be any of the following values:

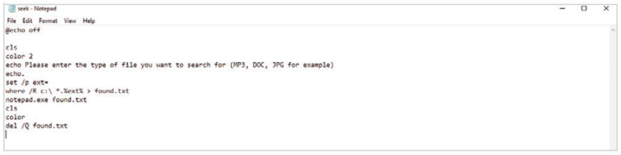


0 = Black	8 = Grey
1 = Blue	9 = Light Blue
2 = Green	A = Light Green
3 = Aqua	B = Light Aqua
4 = Red	C = Light Red
5 = Purple	D = Light Purple
6 = Yellow	E = Light Yellow
7 = White	F = Bright White

STEP 3

Now let's extend the seek.bat batch file:

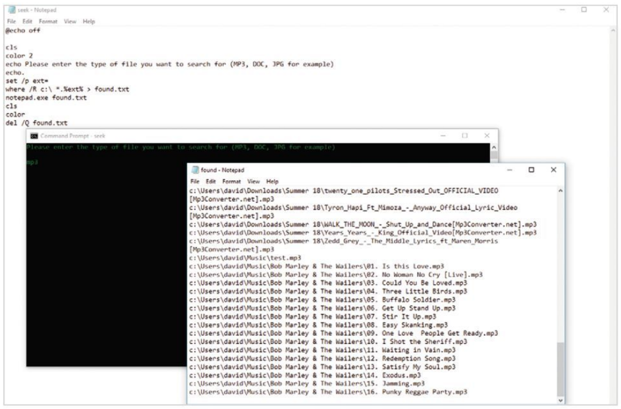
```
@echo off
cls
color 2
echo Please enter the type of file you want to
search for (MP3, DOC, JPG for example)
echo.
set /p ext=
where /R c:\ *.%ext% > found.txt
notepad.exe found.txt
cls
color
del /Q found.txt
```



```
seek - Notepad
File  Edit  Format  View  Help
@echo off

cls
color 2
echo Please enter the type of file you want to search for (MP3, DOC, JPG for example)
echo.
set /p ext=
where /R c:\ *.%ext% > found.txt
notepad.exe found.txt
cls
color
del /Q found.txt
```

STEP 4

Another new command, Where, looks for a specific file or directory based on the user's requirements. In this case, we have created a blank variable called ext that the user can enter the file type in, which then searches using Where and dumps the results in a text file called found.txt. Save and run the batch file.



```
seek - Notepad
File  Edit  Format  View  Help
@echo off

cls
color 2
echo Please enter the type of file you want to search for (MP3, DOC, JPG for example)
echo.
set /p ext=
where /R c:\ *.%ext% > found.txt
notepad.exe found.txt
cls
color
del /Q found.txt
```

```
Command Prompt - seek
Please enter the type of file you want to search for (MP3, DOC, JPG for example)

mp3
```

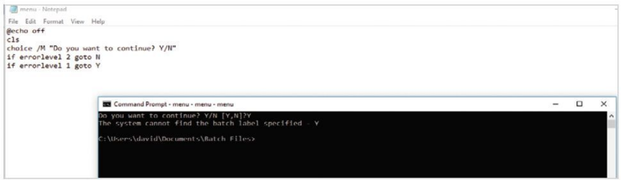
```
found - Notepad
File  Edit  Format  View  Help
c:\Users\david\Downloads\Summer 18\twenty_one_pilots_Stressed_Out_OFFICIAL_VIDEO
[Mp3Converter.net].mp3
c:\Users\david\Downloads\Summer 18\Tyron_Hapi_Ft_Mimoza_-_Anyway_Official_Lyric_Video
[Mp3Converter.net].mp3
c:\Users\david\Downloads\Summer 18\WALK_THE_MOON_-_Shut_Up_and_Dance[Mp3Converter.net].mp3
c:\Users\david\Downloads\Summer 18\Years_Years_-_King_Official_Video[Mp3Converter.net].mp3
c:\Users\david\Downloads\Summer 18\Zedd_Grey_-_The_Middle_Lyrics_ft_Maren_Morris
[Mp3Converter.net].mp3
c:\Users\david\Music\test.mp3
c:\Users\david\Music\Bob Marley & The Wailers\01. Is this Love.mp3
c:\Users\david\Music\Bob Marley & The Wailers\02. No Woman No Cry [Live].mp3
c:\Users\david\Music\Bob Marley & The Wailers\03. Could You Be Loved.mp3
c:\Users\david\Music\Bob Marley & The Wailers\04. Three Little Birds.mp3
c:\Users\david\Music\Bob Marley & The Wailers\05. Buffalo Soldier.mp3
c:\Users\david\Music\Bob Marley & The Wailers\06. Get Up Stand Up.mp3
c:\Users\david\Music\Bob Marley & The Wailers\07. Stir It Up.mp3
c:\Users\david\Music\Bob Marley & The Wailers\08. Easy Skanking.mp3
c:\Users\david\Music\Bob Marley & The Wailers\09. One Love  People Get Ready.mp3
c:\Users\david\Music\Bob Marley & The Wailers\10. I Shot the Sheriff.mp3
c:\Users\david\Music\Bob Marley & The Wailers\11. Waiting in Vain.mp3
c:\Users\david\Music\Bob Marley & The Wailers\12. Redemption Song.mp3
c:\Users\david\Music\Bob Marley & The Wailers\13. Satisfy My Soul.mp3
c:\Users\david\Music\Bob Marley & The Wailers\14. Exodus.mp3
c:\Users\david\Music\Bob Marley & The Wailers\15. Jamming.mp3
c:\Users\david\Music\Bob Marley & The Wailers\16. Punky Reggae Party.mp3
```

CHOICE MENUS

Creating a menu of choices is a classic batch file use and a good example to help expand your batch file programming skills. Here's some code to help you understand how it all works.

STEP 1 Rather than using a variable to process a user's response, batch files can instead use the Choice command in conjunction with an ErrorLevel parameter to make a menu. Create a new file called menu.bat and enter the following:

```
@echo off
cls
choice /M "Do you want to continue? Y/N"
if errorlevel 2 goto N
if errorlevel 1 goto Y
goto End:
```

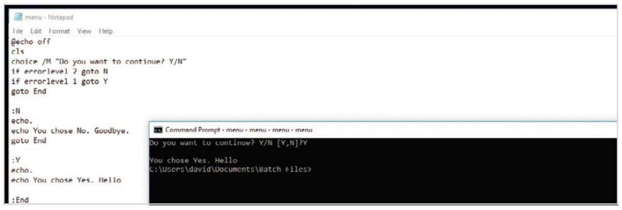


```
@echo off
cls
choice /M "Do you want to continue? Y/N"
if errorlevel 2 goto N
if errorlevel 1 goto Y
```

```
Do you want to continue? Y/N [Y,N]?Y
The system cannot find the batch label specified - Y

C:\Users\david\Documents\Batch Files>
```

STEP 2 Running the code produces an error as we've called a Goto command without any reference to it in the file. Goto does exactly that, goes to a specific line in the batch file. Finish the file with the following and run it again:

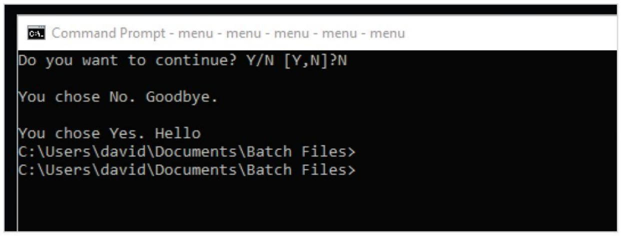
```
:N
echo.
echo You chose No. Goodbye.
goto End

:Y
echo.
echo You chose Yes. Hello

:End
```



```
@echo off
cls
choice /M "Do you want to continue? Y/N"
if errorlevel 2 goto N
if errorlevel 1 goto Y
goto End

:N
echo.
echo You chose No. Goodbye.
goto End

:Y
echo.
echo You chose Yes. Hello

:End
```

```
Do you want to continue? Y/N [Y,N]?Y

You chose Yes. Hello
C:\Users\david\Documents\Batch Files>
```

STEP 3 The output from your choice is different depending on whether you pick Y or N. The :End part simply signifies the end of the file (also known as EOF). Without it the batch file runs through each line and display the Y response even if you enter N; so it's important to remember to follow your Goto commands.



```
Do you want to continue? Y/N [Y,N]?N

You chose No. Goodbye.

You chose Yes. Hello
C:\Users\david\Documents\Batch Files>
C:\Users\david\Documents\Batch Files>
```

STEP 4 ErrorLevels are essentially variables and the /M switch of Choice allows a descriptive message string to be displayed. Extend this menu with something new:

```
@echo off
cls
echo.
echo ---------------------------------------------
echo.
echo Please choose a directory.
echo.
echo Press 1 for c:\Music
echo.
echo Press 2 for c:\Documents
echo.
echo Press 3 for c:\Pictures
echo.
echo Press 4 for c:\Videos
echo.
echo ---------------------------------------------
choice /C 1234
if errorlevel 4 goto Videos
if errorlevel 3 goto Pictures
if errorlevel 2 goto Documents
if errorlevel 1 goto Music
```

STEP 5 Now add the Goto sections:

```
:Videos
cls
CD %HOMEPATH%\Videos
echo You are now in the Videos directory.
goto End

:Pictures
cls
CD %HOMEPATH%\Pictures
echo You are now in the Pictures directory.
goto End

:Documents
cls
CD %HOMEPATH%\Documents
echo You are now in the Documents directory.
goto End

:Music
cls
CD %HOMEPATH%\Music
echo you are now in the Music directory.
goto End

:End
```

STEP 6 When executed, the batch file displays a menu and with each choice the code changes directory to the one the user entered. The %HOMEPATH% system variable will enter the currently logged in user's Music, Pictures and so directories, and not anyone else's.

Loops and Repetition

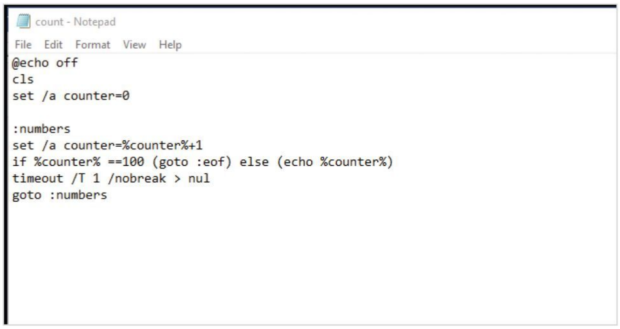
Looping and repeating commands are the staple diet of every programming language, including batch file programming. For example, you can create a simple countdown or even make numbered files or directories in the system.

COUNTERS

Creating code that counts in increasing or decreasing number sets is great for demonstrating loops. With that in mind, let's look at the If statement a little more, alongside more variables, and introduce the Else, Timeout and eof (End of File) commands.

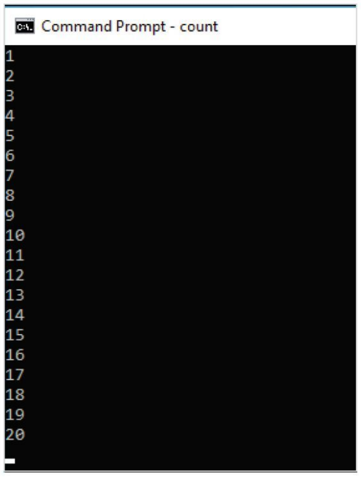
STEP 1 Start by creating a new batch file called count.bat. Enter the following, save it and run:

```
@echo off
cls
set /a counter=0

:numbers
set /a counter=%counter%+1
if %counter% ==100 (goto :eof) else (echo
%counter%)
timeout /T 1 /nobreak > nul
goto :numbers
```



```
@echo off
cls
set /a counter=0

:numbers
set /a counter=%counter%+1
if %counter% ==100 (goto :eof) else (echo %counter%)
timeout /T 1 /nobreak > nul
goto :numbers
```

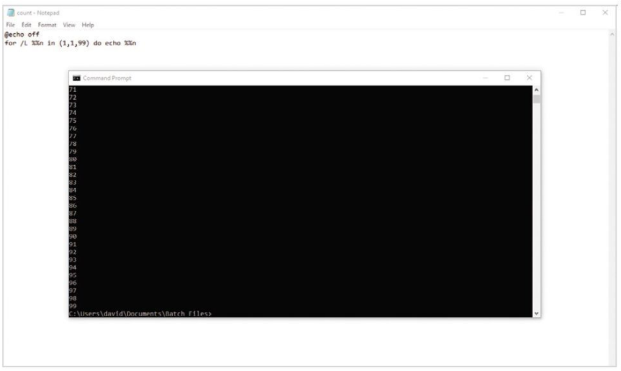
STEP 2 The count.bat code starts at number one and counts, scrolling down the screen, until it reaches 100. The Timeout command leaves a one second gap between numbers and the Else statement continues until the counter variable equals 100 before going to the eof (End Of File), thus closing the loop.



```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
```

STEP 3 The count.bat is a rough way of demonstrating a loop; a better approach would be to use a for loop. Try this example instead:

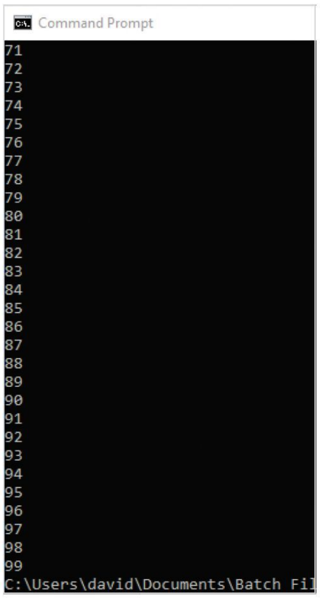
```
@echo off
for /L %%n in (1,1,99) do echo %%n
```



```
@echo off
for /L %%n in (1,1,99) do echo %%n
```

```
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
C:\Users\david\Documents\Batch Files>
```

STEP 4 Breaking it down, there's For, then the /L switch, which handles a range of numbers. Then the parameter labelled as %%n to denote a number. Then the in (1,1,99) part, which tells the statement how to count, as in 1 (start number), 1 (steps to take), 99 (the end number). The next part is do, meaning DO whatever command is after.

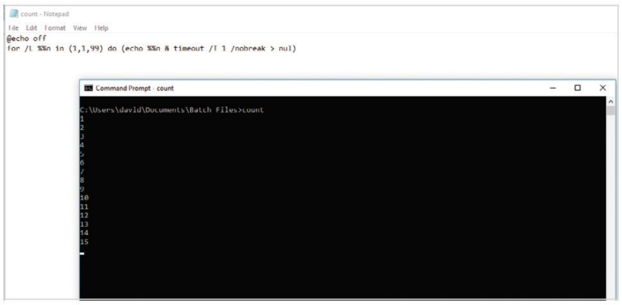


```
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
C:\Users\david\Documents\Batch Fil
```

STEP 5

You can include the pause between the numbers easily enough within the far simpler For loop by adding multiple commands after the Do For loop. The brackets and ampersand (&) separate the different commands. Try this:

```
@echo off
for /L %%n in (1,1,99) do (echo %%n & timeout /T 1
/nobreak > nul)
```



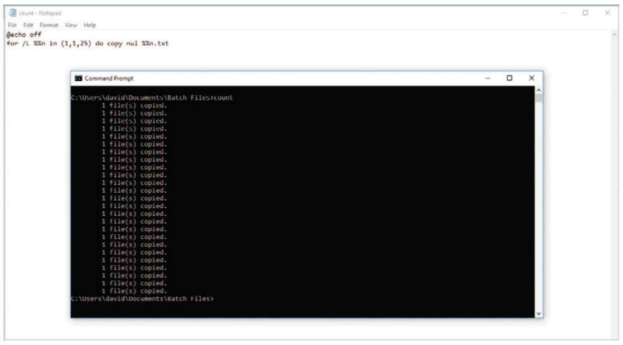
```
count - Notepad
File  Edit  Format  View  Help
@echo off
for /L %%n in (1,1,99) do (echo %%n & timeout /T 1 /nobreak > nul)
```

```
Command Prompt - count
C:\Users\david\Documents\Batch Files>count
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
_
```

STEP 6

One of the great time saving uses of batch files is to create multiple, numbered files. Assume that you want twenty five text files within a directory, all numbered from 1 to 25. A For loop much like the previous example does the trick:

```
@echo off
for /L %%n in (1,1,25) do copy nul %%n.txt
```



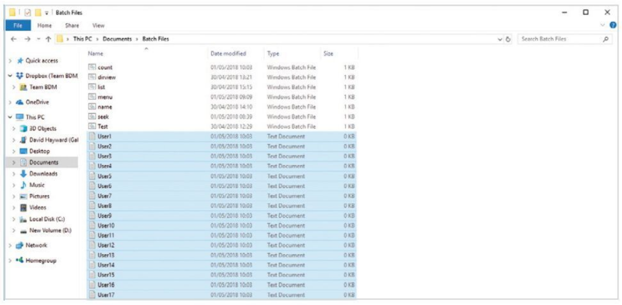
```
count - Notepad
File  Edit  Format  View  Help
@echo off
for /L %%n in (1,1,25) do copy nul %%n.txt
```

```
Command Prompt
C:\Users\david\Documents\Batch Files>count
        1 file(s) copied.
        1 file(s) copied.
        1 file(s) copied.
        1 file(s) copied.
        1 file(s) copied.
        1 file(s) copied.
        1 file(s) copied.
        1 file(s) copied.
        1 file(s) copied.
        1 file(s) copied.
        1 file(s) copied.
        1 file(s) copied.
        1 file(s) copied.
        1 file(s) copied.
        1 file(s) copied.
        1 file(s) copied.
        1 file(s) copied.
        1 file(s) copied.
        1 file(s) copied.
        1 file(s) copied.
        1 file(s) copied.
        1 file(s) copied.
        1 file(s) copied.
        1 file(s) copied.
        1 file(s) copied.
C:\Users\david\Documents\Batch Files>
```

STEP 7

If you open Windows Explorer, and navigate to the Batch Files directory where you're working from, you can now see 25 text files all neatly numbered. Of course, you can append the file name with something like user1.txt and so on by altering the code to read:

```
@echo off
for /L %%n in (1,1,25) do copy nul User%%n.txt
```

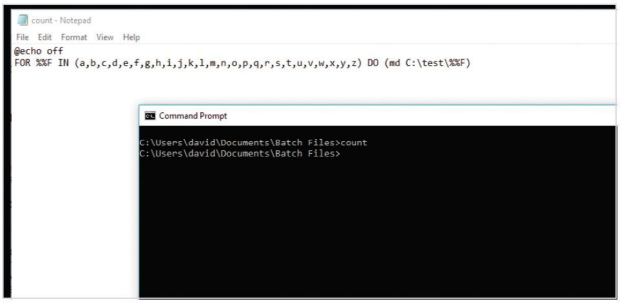


Name	Date modified	Type	Size
count	01/05/2018 10:03	Windows Batch File	1 KB
dirview	30/04/2018 13:21	Windows Batch File	1 KB
list	30/04/2018 15:15	Windows Batch File	1 KB
menu	01/05/2018 09:09	Windows Batch File	1 KB
name	30/04/2018 14:10	Windows Batch File	1 KB
seek	01/05/2018 08:39	Windows Batch File	1 KB
Test	30/04/2018 12:29	Windows Batch File	1 KB
User1	01/05/2018 10:03	Text Document	0 KB
User2	01/05/2018 10:03	Text Document	0 KB
User3	01/05/2018 10:03	Text Document	0 KB
User4	01/05/2018 10:03	Text Document	0 KB
User5	01/05/2018 10:03	Text Document	0 KB
User6	01/05/2018 10:03	Text Document	0 KB
User7	01/05/2018 10:03	Text Document	0 KB
User8	01/05/2018 10:03	Text Document	0 KB
User9	01/05/2018 10:03	Text Document	0 KB
User10	01/05/2018 10:03	Text Document	0 KB
User11	01/05/2018 10:03	Text Document	0 KB
User12	01/05/2018 10:03	Text Document	0 KB
User13	01/05/2018 10:03	Text Document	0 KB
User14	01/05/2018 10:03	Text Document	0 KB
User15	01/05/2018 10:03	Text Document	0 KB
User16	01/05/2018 10:03	Text Document	0 KB
User17	01/05/2018 10:03	Text Document	0 KB

STEP 8

There are different ways of using the For loop. In this example, the code creates 26 directories, one for each letter of the alphabet, within the directory c:\test which the batch file makes using the MD command:

```
@echo off
FOR %%F IN (a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,
s,t,u,v,w,x,y,z) DO (md C:\test\%%F)
```

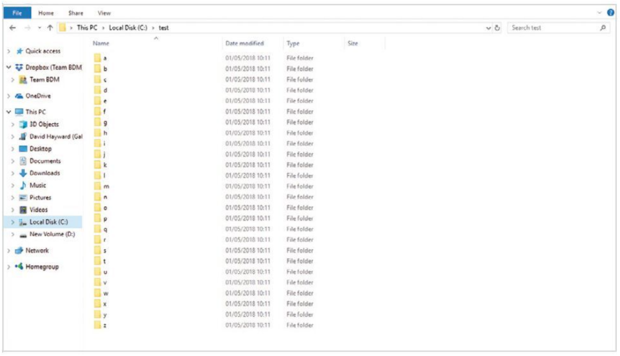


```
count - Notepad
File  Edit  Format  View  Help
@echo off
FOR %%F IN (a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z) DO (md C:\test\%%F)
```

```
Command Prompt
C:\Users\david\Documents\Batch Files>count
C:\Users\david\Documents\Batch Files>
```

STEP 9

Loops can be powerful and extremely useful elements in a batch file. While creating 26 directories may not sound too helpful, imagine having to create 1,000 users on a network and assign each one their own set of unique directories. This is where a batch file saves an immense amount of time.

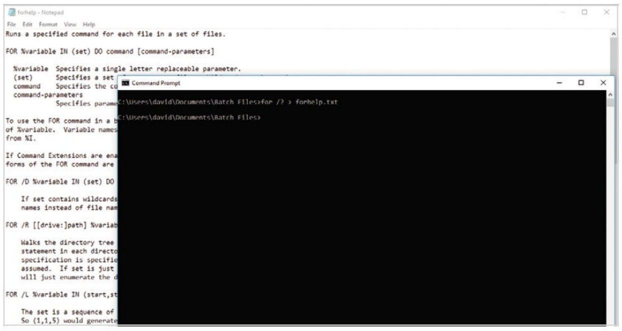


Name	Date modified	Type	Size
a	01/05/2018 10:11	File folder	
b	01/05/2018 10:11	File folder	
c	01/05/2018 10:11	File folder	
d	01/05/2018 10:11	File folder	
e	01/05/2018 10:11	File folder	
f	01/05/2018 10:11	File folder	
g	01/05/2018 10:11	File folder	
h	01/05/2018 10:11	File folder	
i	01/05/2018 10:11	File folder	
j	01/05/2018 10:11	File folder	
k	01/05/2018 10:11	File folder	
l	01/05/2018 10:11	File folder	
m	01/05/2018 10:11	File folder	
n	01/05/2018 10:11	File folder	
o	01/05/2018 10:11	File folder	
p	01/05/2018 10:11	File folder	
q	01/05/2018 10:11	File folder	
r	01/05/2018 10:11	File folder	
s	01/05/2018 10:11	File folder	
t	01/05/2018 10:11	File folder	
u	01/05/2018 10:11	File folder	
v	01/05/2018 10:11	File folder	
w	01/05/2018 10:11	File folder	
x	01/05/2018 10:11	File folder	
y	01/05/2018 10:11	File folder	
z	01/05/2018 10:11	File folder	

STEP 10

Should you ever get stuck when using the various commands within a batch file, drop into the command prompt and enter the command followed by a question switch. For example, for /? or if /?. You get an on-screen help file detailing the commands' use. For easier reading, pipe it to a text file:

```
For /? > forhelp.txt
```



```
forhelp - Notepad
File  Edit  Format  View  Help
Runs a specified command for each file in a set of files.

FOR %variable IN (set) DO command [command-parameters]

  %variable  Specifies a single letter replaceable parameter.
  (set)      Specifies a set
  command    Specifies the co
  command-parameters
             Specifies parame

To use the FOR command in a b
of %variable.  Variable names
from %I.

If Command Extensions are ena
forms of the FOR command are

FOR /D %variable IN (set) DO

    If set contains wildcards
    names instead of file nam

FOR /R [[drive:]path] %variab

    Walks the directory tree
    statement in each directo
    specification is specifie
    assumed.  If set is just
    will just enumerate the d

FOR /L %variable IN (start,st

    The set is a sequence of
    So (1,1,5) would generate
```

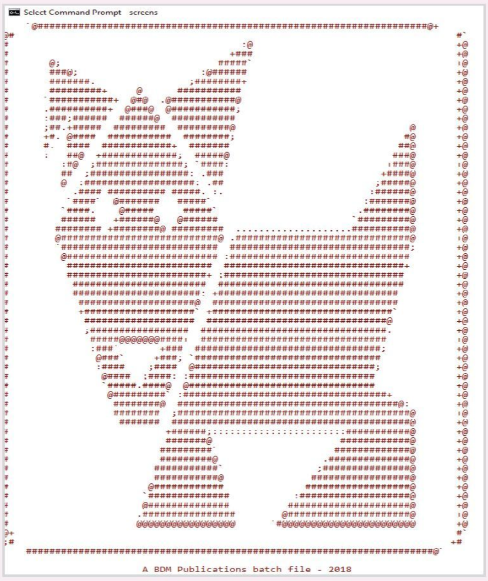
```
Command Prompt
C:\Users\david\Documents\Batch Files>for /? > forhelp.txt

C:\Users\david\Documents\Batch Files>
```

Creating a Batch File Game

Based on what you've looked at so far with batch files, you can probably come up with your own simple text adventure or multiple-choice game. Here's one that we created, with which you're free to edit and make your own.

Make up your own questions but how about also including an introductory or loading screen? Make your loading screen in a separate batch file and save it as screens.bat (for example). Then, from the main game batch file, you can load it at the beginning of the file with the 'call' command followed by colour to reset the game's colours:

```
@echo off
Cls
Call screens.bat
color
:start
set /a score=0
set /a question=0
cls
set /p name= What is your name?
```



Select Command Prompt - screens

A BDM Publications batch file - 2018

```
@echo off
Cls
:start
set /a score=0
set /a question=0
cls
set /p name= What is your name?

:menu
cls
echo.
echo ***********************************************
   **
echo.
echo Welcome %name% to the super-cool trivia game.
echo.
echo Press 1 to get started
echo.
echo Press 2 for instructions
echo.
echo Press Q to quit
echo.
echo ***********************************************
   **
choice /C 12Q
if errorlevel 3 goto :eof
if errorlevel 2 goto Instructions
if errorlevel 1 goto Game

:Instructions
cls
echo.
echo ***********************************
echo.
echo The instructions are simple. Answer the
   questions correctly.
echo.
echo ***********************************
pause
cls
goto menu

:Game
set /a question=%question%+1
cls
if %question% ==5 (goto end) else (echo you are on
   question %question%)
echo.
echo get ready for the question...
echo.
timeout /T 5 /nobreak > nul
if %question% ==5 (goto end) else (goto %question%)

:1
cls
echo.
echo ***************************
echo.
```

```
echo Your current score is %score%
echo.
echo ******************************
echo.
echo.
echo Question %question%.
echo.
echo Which of the following version of Windows is the
  best?
echo.
echo A. Windows 10
echo.
echo B. Windows ME
echo.
echo C. Windows Vista
echo.
choice /C abc
if errorlevel 3 goto wrong
if errorlevel 2 goto wrong
if errorlevel 1 goto correct

:2
cls
echo.
echo ******************************
echo.
echo Your current score is %score%
echo.
echo ******************************
echo.
echo.
echo Question %question%.
echo.
echo Which of the following version of Windows is the
  most stable?
echo.
echo A. Windows 10
echo.
echo B. Windows 95
echo.
echo C. Windows ME
echo.
choice /C abc
if errorlevel 3 goto wrong
if errorlevel 2 goto wrong
if errorlevel 1 goto correct

:3
cls
echo.
echo ******************************
echo.
echo Your current score is %score%
echo.
echo ******************************
echo.
echo.
echo Question %question%.
echo.
echo Which of the following Windows version is the
  latest?
echo.
echo A. Windows 10
echo.
echo B. Windows 98
echo.
echo C. Windows 7
echo.
choice /C abc
if errorlevel 3 goto wrong
if errorlevel 2 goto wrong
if errorlevel 1 goto correct

:4
cls
echo.
echo ******************************
echo.
echo Your current score is %score%
echo.
echo ******************************
echo.
echo.
echo Question %question%.
echo.
echo Which of the following Windows uses DirectX 12?
echo.
echo A. Windows 10
echo.
echo B. Windows 3.11
echo.
echo C. Windows XP
echo.
choice /C abc
if errorlevel 3 goto wrong
if errorlevel 2 goto wrong
if errorlevel 1 goto correct

:Wrong
cls
echo ******************
echo.
echo WRONG!!!!
echo.
echo ******************
set /a score=%score%-1
pause
goto :game

:correct
cls
echo ******************
echo.
echo CORRECT. YIPEE!!!
echo.
echo ******************
set /a score=%score%+1
pause
goto :game

:end
cls
echo ******************************
echo.
echo Well done, %name%, you have answered all the
  questions
echo.
echo And your final score is....
echo.
echo %score%
echo.
echo ********************************
choice /M “play again? Y/N”
if errorlevel 2 goto :eof
if errorlevel 1 goto start
```

Coding on Linux

Linux is such a versatile operating system that's both malleable and powerful, while offering the programmer a perfect foundation on which to build their skills. While all the popular and mainstream programming languages are available on Linux, as they are on Windows and macOS, Linux also utilises its own coding language, called scripting.

Bash scripting on Linux can be used to create a wealth of useful, real-world programs that interact with the user, or simply work in the background based on a pre-defined schedule. Scripting is a powerful interface to the Linux system, so we've crafted this section to help you get to grips with how everything fits together, and how to make some amazing Linux scripts.

Why Linux?

For many of its users, Linux means freedom. Freedom from the walled-gardened 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.

FREE AND OPEN

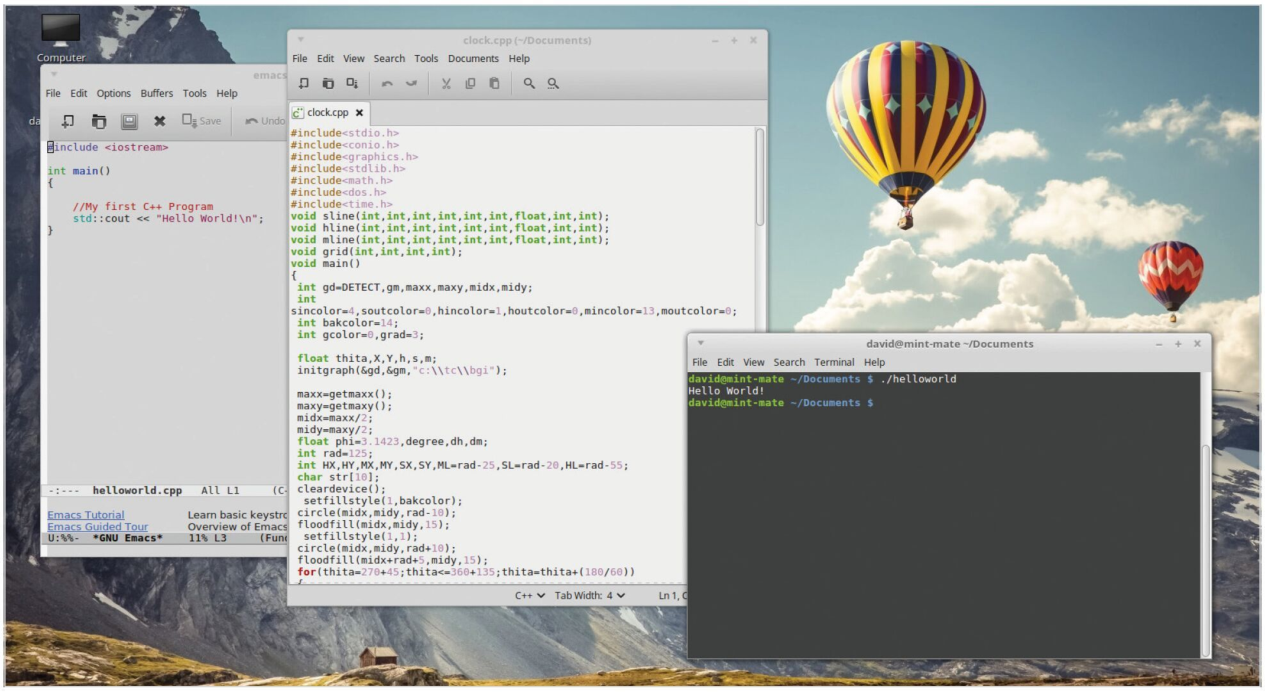
Linux is a fantastic fit for those who want something different; the efficiency of the system, the availability of applications and its stability, are just a few reasons why.

The first thing we need to address is that there is no basic operating system called Linux. Fundamentally, Linux is the operating system kernel, the core component of an OS. When talking about Linux we are, in fact, referring to 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 offers the user something a little different. While each has the Linux kernel at its core, they offer a different looking desktop environment, different pre-loaded 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 very differently to Windows or macOS. For a start, it's 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 distros available on the Internet have a certain 'spin', some offer increased security, a fancy-

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


Computer
emacs
File Edit Options Buffers Tools Help
Save
Undo
clock.cpp (~/Documents)
File Edit View Search Tools Documents Help
clock.cpp
C++ Tab Width: 4 Ln 1,
david@mint-mate ~/Documents
File Edit View Search Terminal Help

```
#include <iostream>

int main()
{

    //My first C++ Program
    std::cout << "Hello World!\n";
}
```

```
-:---  helloworld.cpp    All L1
Emacs Tutorial      Learn basic keystro
Emacs Guided Tour   Overview of Emacs
U:%%-  *GNU Emacs*   11% L3    (Fun
```

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<stdlib.h>
#include<math.h>
#include<dos.h>
#include<time.h>
void sline(int,int,int,int,int,int,float,int,int);
void hline(int,int,int,int,int,int,float,int,int);
void mline(int,int,int,int,int,int,float,int,int);
void grid(int,int,int,int);
void main()
{
 int gd=DETECT,gm,maxx,maxy,midx,midy;
 int
sincolor=4,soutcolor=0,hincolor=1,houtcolor=0,mincolor=13,moutcolor=0;
 int bakcolor=14;
 int gcolor=0,grad=3;

 float thita,X,Y,h,s,m;
 initgraph(&gd,&gm,"c:\\tc\\bgi");

 maxx=getmaxx();
 maxy=getmaxy();
 midx=maxx/2;
 midy=maxy/2;
 float phi=3.1423,degree,dh,dm;
 int rad=125;
 int HX,HY,MX,MY,SX,SY,ML=rad-25,SL=rad-20,HL=rad-55;
 char str[10];
 cleardevice();
  setfillstyle(1,bakcolor);
 circle(midx,midy,rad-10);
 floodfill(midx,midy,15);
  setfillstyle(1,1);
 circle(midx,midy,rad+10);
 floodfill(midx+rad+5,midy,15);
 for(thita=270+45;thita<=360+135;thita=thita+(180/60))
```

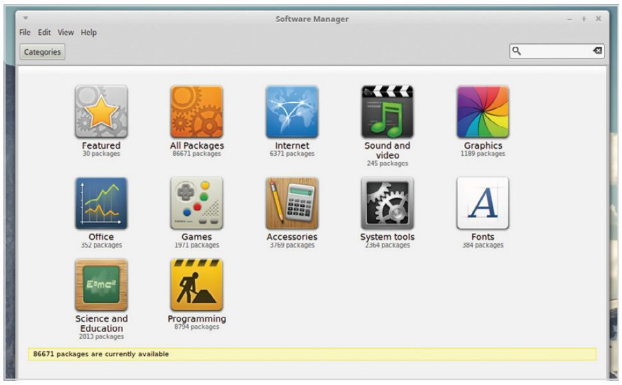
```
david@mint-mate ~/Documents $ ./helloworld
Hello World!
david@mint-mate ~/Documents $
```

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 use with Python, web development, C++, Java and so on, or even 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 pre-installed in a high percentage of Linux distros, 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), as the Linux code has been streamlined and is free from third-party 'bloatware', which hogs those systems resources. A more efficient system, means more available resources for the coding and testing environment and the programs you will eventually create. Less resource use 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.

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



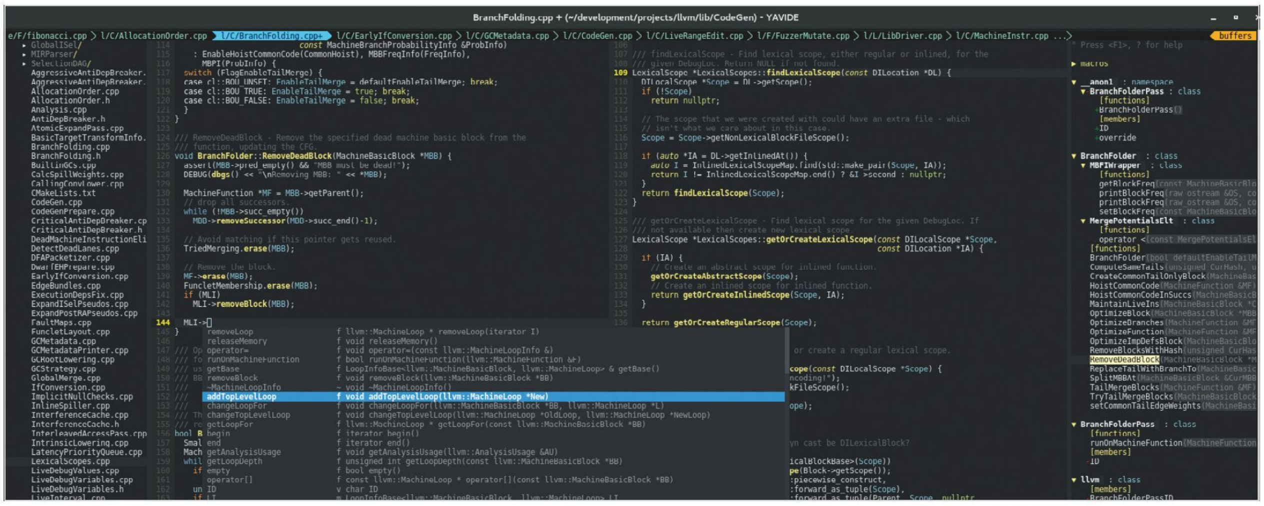
There are thousands of free packages available for programmers under Linux.

So rather than throwing away an old computer, it can be reused with a Linux distro.

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

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

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



BranchFolding.cpp + (~/development/projects/llvm/lib/CodeGen) - YAVIDE

```
e/F/fibonacci.cpp > l/C/AllocationOrder.cpp > l/C/BranchFolding.cpp+ > l/C/EarlyIfConversion.cpp > l/C/GCMetadata.cpp > l/C/CodeGen.cpp > l/C/LiveRangeEdit.cpp > l/F/FuzzerMutate.cpp > l/L/LibDriver.cpp > l/C/MachineInstr.cpp ...>
```

```
                                   const MachineBranchProbabilityInfo &ProbInfo)
    : EnableHoistCommonCode(CommonHoist), MBBFreqInfo(FreqInfo),
      MBPI(ProbInfo) {
  switch (FlagEnableTailMerge) {
  case cl::BOU_UNSET: EnableTailMerge = defaultEnableTailMerge; break;
  case cl::BOU_TRUE: EnableTailMerge = true; break;
  case cl::BOU_FALSE: EnableTailMerge = false; break;
  }
}

/// RemoveDeadBlock - Remove the specified dead machine basic block from the
/// function, updating the CFG.
void BranchFolder::RemoveDeadBlock(MachineBasicBlock *MBB) {
  assert(MBB->pred_empty() && "MBB must be dead!");
  DEBUG(dbgs() << "\nRemoving MBB: " << *MBB);

  MachineFunction *MF = MBB->getParent();
  // drop all successors.
  while (!MBB->succ_empty())
    MBB->removeSuccessor(MBB->succ_end()-1);

  // Avoid matching if this pointer gets reused.
  TriedMerging.erase(MBB);

  // Remove the block.
  MF->erase(MBB);
  FuncletMembership.erase(MBB);
  if (MLI)
    MLI->removeBlock(MBB);

  MLI->
}
     removeLoop                   f llvm::MachineLoop * removeLoop(iterator I)
     releaseMemory                f void releaseMemory()
     operator=                    f void operator=(const llvm::MachineLoopInfo &)
     runOnMachineFunction         f bool runOnMachineFunction(llvm::MachineFunction &F)
     getBase                      f LoopInfoBase<llvm::MachineBasicBlock, llvm::MachineLoop> & getBase()
     removeBlock                  f void removeBlock(llvm::MachineBasicBlock *BB)
     ~MachineLoopInfo             ~ void ~MachineLoopInfo()
     addTopLevelLoop              f void addTopLevelLoop(llvm::MachineLoop *New)
     changeLoopFor                f void changeLoopFor(llvm::MachineBasicBlock *BB, llvm::MachineLoop *L)
     changeTopLevelLoop           f void changeTopLevelLoop(llvm::MachineLoop *OldLoop, llvm::MachineLoop *NewLoop)
     getLoopFor                   f llvm::MachineLoop * getLoopFor(const llvm::MachineBasicBlock *BB)
```

```
/// findLexicalScope - Find lexical scope, either regular or inlined, for the
/// given DebugLoc. Return NULL if not found.
LexicalScope *LexicalScopes::findLexicalScope(const DILocation *DL) {
  DILocalScope *Scope = DL->getScope();
  if (!Scope)
    return nullptr;

  // The scope that we were created with could have an extra file - which
  // isn't what we care about in this case.
  Scope = Scope->getNonLexicalBlockFileScope();

  if (auto *IA = DL->getInlinedAt()) {
    auto I = InlinedLexicalScopeMap.find(std::make_pair(Scope, IA));
    return I != InlinedLexicalScopeMap.end() ? &I->second : nullptr;
  }
  return findLexicalScope(Scope);
}

/// getOrCreateLexicalScope - Find lexical scope for the given DebugLoc. If
/// not available then create new lexical scope.
LexicalScope *LexicalScopes::getOrCreateLexicalScope(const DILocalScope *Scope,
                                                     const DILocation *IA) {
  if (IA) {
    // Create an abstract scope for inlined function.
    getOrCreateAbstractScope(Scope);
    // Create an inlined scope for inlined function.
    return getOrCreateInlinedScope(Scope, IA);
  }

  return getOrCreateRegularScope(Scope);
```

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

The Best Linux Distributions

There are numerous 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.

LINUX MINT – LINUXMINT.COM

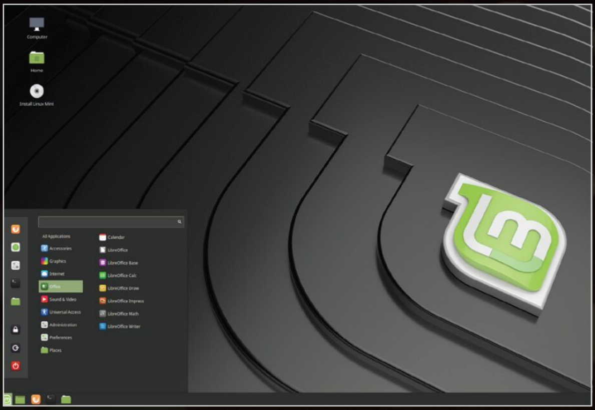
By far the most popular Linux distro (distribution) is Linux Mint. Mint began life back in 2006, as an alternative to the then, most popular distro, Ubuntu. Although based on Ubuntu’s Long Term Support build, Linux Mint took a different direction to offer the user a better overall experience.

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, although MATE and Xfce models are also available.

Cinnamon is a graphically rich desktop environment, while MATE uses less fancy graphics but is more stable on a wider variety of desktop systems and finally Xfce is an extremely streamlined desktop environment that is 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.

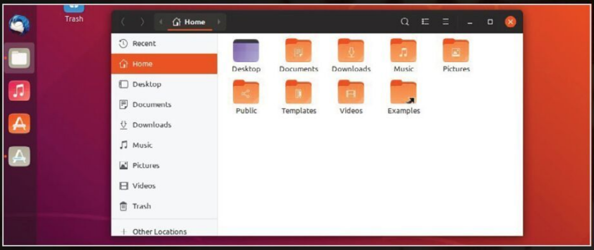
In fact, it’s recommended that you spend at least some time trying different environments and even different distros, to see which suits you and your computer best.



UBUNTU – UBUNTU.COM

The second most popular distro available is Ubuntu: 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, sadly, some wrong choices along the way with regards to its presentation, along with some unfavourable and controversial elements involving privacy, 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, it’s impressive although it can be a little confusing for former Windows users



and it’s a little heavy on system resources (especially if you plan to install 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 the complete Linux experience.

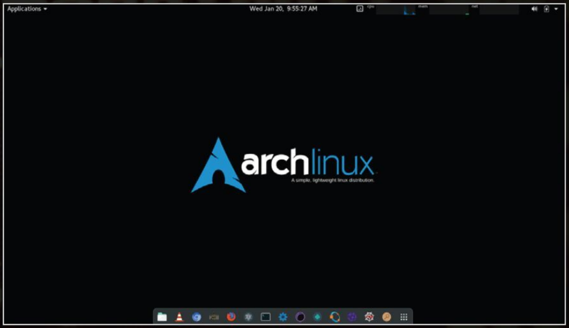
ARCH – ARCHLINUX.ORG

Arch is one of the longest running Linux distributions and forms the basis of many other versions of Linux. So 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 that will help you get started.

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

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

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

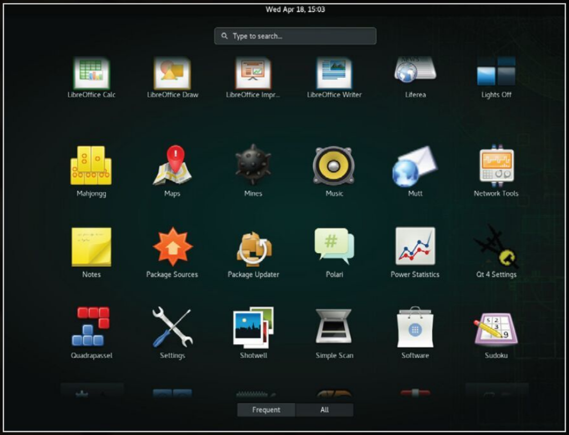


OPENSUSE – OPENSUSE.ORG

Most Linux distributions fall into two camps: those with the latest features and technology like Ubuntu and Mint and those with fewer new features, but rock-solid reliability, like Debian.

Meanwhile, openSUSE attempts to cover both bases. OpenSUSE Leap is a rock-solid system. It's developed openly by a community, along with SUSE employees. They develop an enterprise-level operating system: SUSE, which 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.

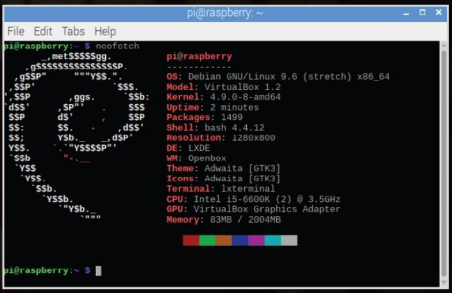
openSUSE is a highly respected Linux distribution and many of its core contributors work on the Linux Kernel, LibreOffice, Gnome and other key Linux areas. In short: openSUSE is where you'll find the pros hanging out.



RASPBERRY PI DESKTOP – RASPBERRYPI.ORG/DOWNLOADS/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 since it was introduced six years ago.

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 employ it to build electronics, 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.



```
pi@raspberry:~ $ neofetch
       _,met$$$$$gg.          pi@raspberry
    ,g$$$$$$$$$$$$$$$P.       ------------
  ,g$$P"     """Y$$.".        OS: Debian GNU/Linux 9.6 (stretch) x86_64
 ,$$P'              `$$$.     Model: VirtualBox 1.2
',$$P       ,ggs.     `$$b:   Kernel: 4.9.0-8-amd64
`d$$'     ,$P"'   .    $$$    Uptime: 2 minutes
 $$P      d$'     ,    $$P    Packages: 1499
 $$:      $$.   -    ,d$$'    Shell: bash 4.4.12
 $$;      Y$b._   _,d$P'      Resolution: 1280x800
 Y$$.    `.`"Y$$$$P"'         DE: LXDE
 `$$b      "-.__              WM: Openbox
  `Y$$                        Theme: Adwaita [GTK3]
   `Y$$.                      Icons: Adwaita [GTK3]
     `$$b.                    Terminal: lxterminal
       `Y$$b.                 CPU: Intel i5-6600K (2) @ 3.5GHz
          `"Y$b._             GPU: VirtualBox Graphics Adapter
              `"""            Memory: 83MB / 2004MB

pi@raspberry:~ $
```

Originally, Raspbian was exclusive to the Pi hardware, as 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.

As with 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 definitely one of the top distros to consider.

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

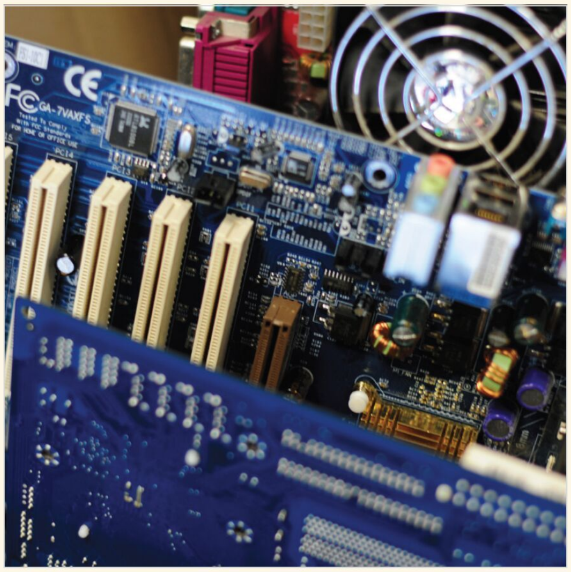
Here's what you'll need to install and run Linux Mint as we work through this book. You have several choices available, so take your time and see which works best for you.

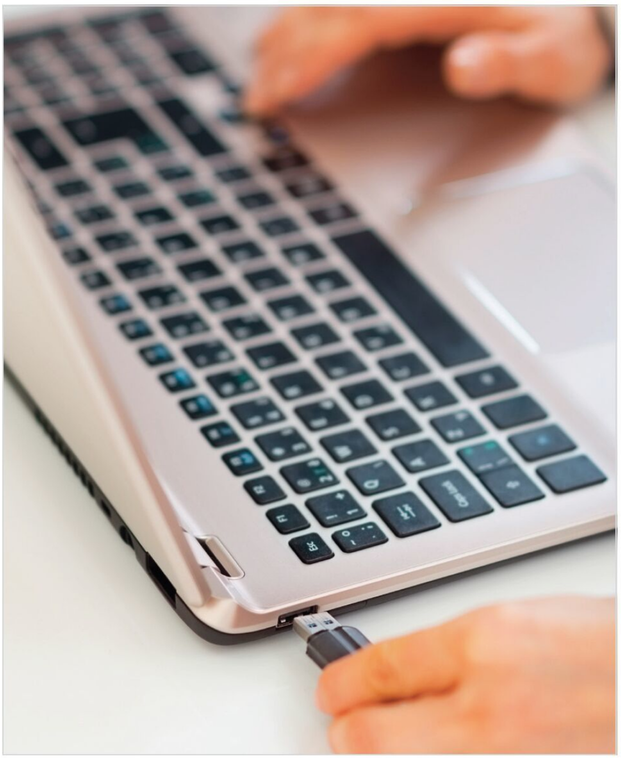
SYSTEM REQUIREMENTS

The minimum system requirements for Linux Mint are as follows:

CPU	700MHz
RAM	512MB
Hard Drive Space	9GB (20GB recommended)
Monitor	1024 x 768 resolution

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



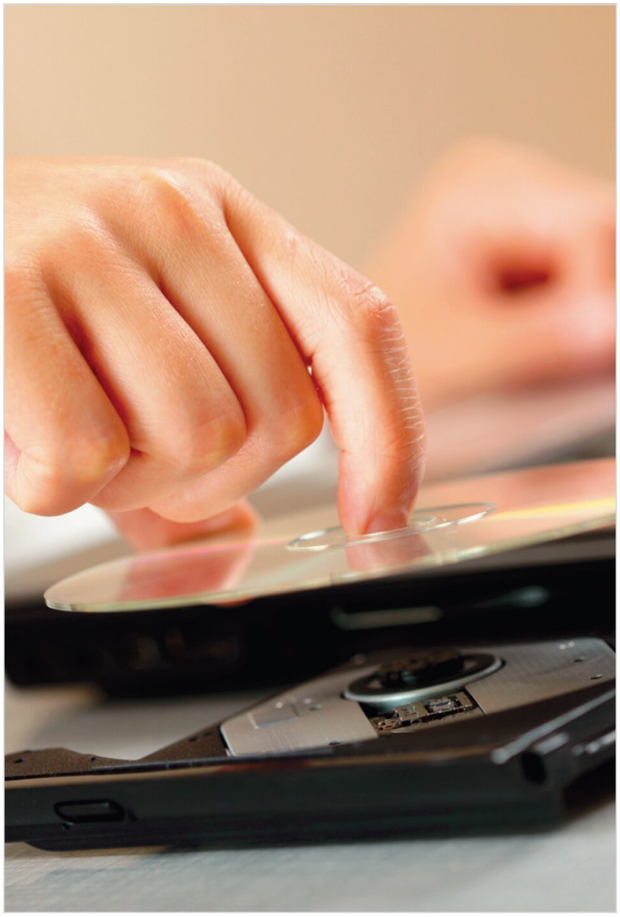


USB INSTALLATION

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

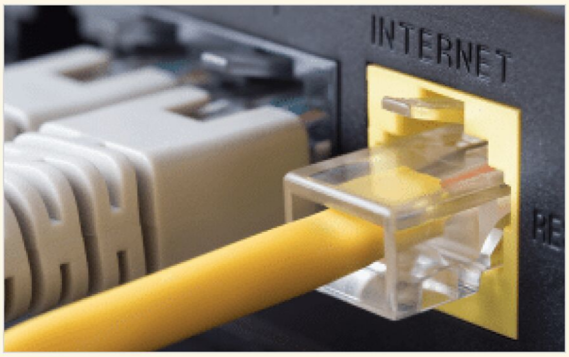
DVD INSTALLATION

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



INTERNET CONNECTION

It goes without saying, that an Internet connection is vital for making sure that Linux Mint is resourced 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 looking to breathe new life into an older Mac, make sure it's an Intel CPU model and not the Power-PC models. However, be aware, it's not as pain-free as installing on to a PC.

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, more on that later. There are many different virtual environment apps available, however VirtualBox, from Oracle, is one of the easiest. You can get the latest version from **www.virtualbox.org**.

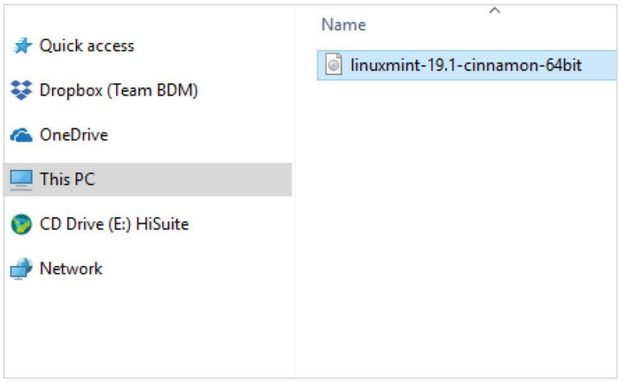
Creating a Linux Installer on Windows

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

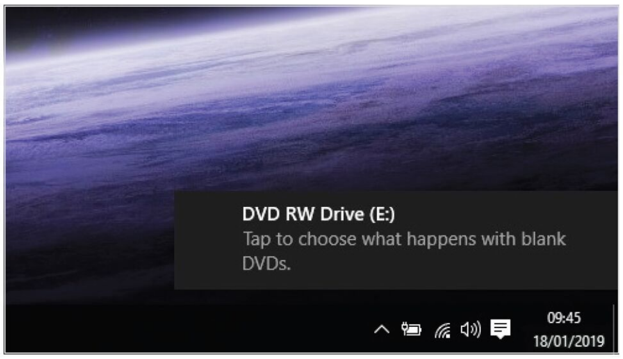
DVD BOOTABLE MEDIA

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

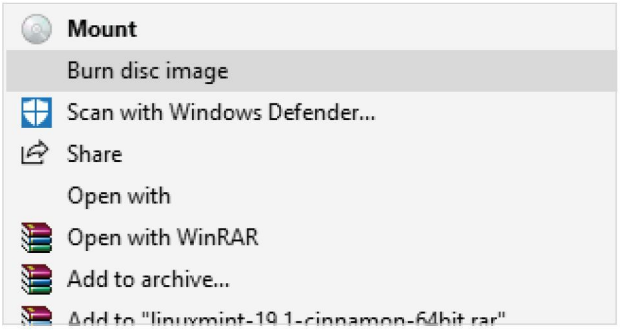
STEP 1 First locate the ISO image of Linux you've already downloaded. In Windows 7, 8.1 and 10 computers, you'll usually find this in the Downloads folder unless, when saving it, you've specified a different location.



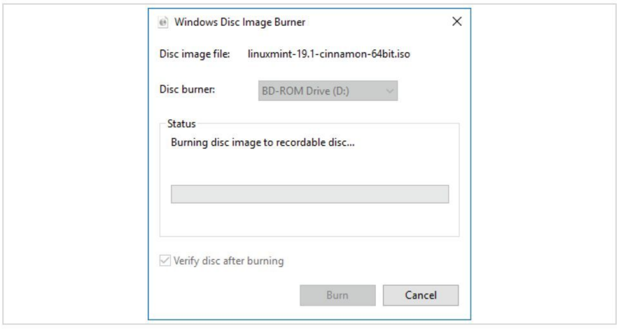
STEP 2 Next, insert a recordable DVD disc into your computer's optical drive. After a few seconds, while the disc is read, Windows will display a pop-up message asking you what to do with the newly inserted disc, ignore this as we're going to use the built-in image burning function.


DVD RW Drive (E:)
Tap to choose what happens with blank DVDs.

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.


Mount
Burn disc image
Scan with Windows Defender...
Share
Open with
Open with WinRAR
Add to archive...
Add to "linuxmint-19.1-cinnamon-64bit.rar"

STEP 4 With the Windows Disc Image Burner dialogue box open, click on the 'Verify disc after burning' tick box, followed by 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 will run through the verification stage and when done, the optical drive should auto-eject the disc for you.


Windows Disc Image Burner

Disc image file: linuxmint-19.1-cinnamon-64bit.iso

Disc burner: BD-ROM Drive (D:)

Status

Burning disc image to recordable disc...

[✓] Verify disc after burning

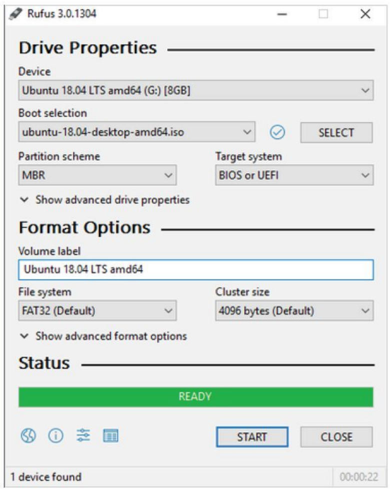
Burn | Cancel

USB BOOTABLE MEDIA

USB media is faster than a DVD and it's often more convenient as most modern PCs don't have an optical drive installed. The process of transferring the image is easy, but you'll 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 **https://rufus.akeo.ie/**. Scroll down the page a little and you'll come to a Download heading, under which you'll see the latest version of Rufus. Left-click on the link to start the download.



Rufus 3.0.1304

Drive Properties

Device: Ubuntu 18.04 LTS amd64 (G:) [8GB]

Boot selection: ubuntu-18.04-desktop-amd64.iso

Partition scheme: MBR — Target system: BIOS or UEFI

Format Options

Volume label: Ubuntu 18.04 LTS amd64

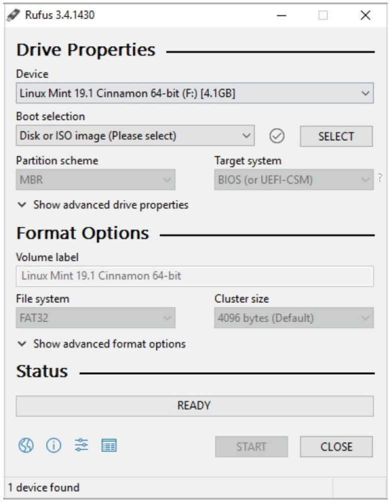
File system: FAT32 (Default) — Cluster size: 4096 bytes (Default)

Status: READY

1 device found 00:00:22

STEP 2

Double-click the downloaded Rufus executable; you can 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).



Rufus 3.4.1430

Drive Properties

Device: Linux Mint 19.1 Cinnamon 64-bit (F:) [4.1GB]

Boot selection: Disk or ISO image (Please select)

Partition scheme: MBR — Target system: BIOS (or UEFI-CSM)

Format Options

Volume label: Linux Mint 19.1 Cinnamon 64-bit

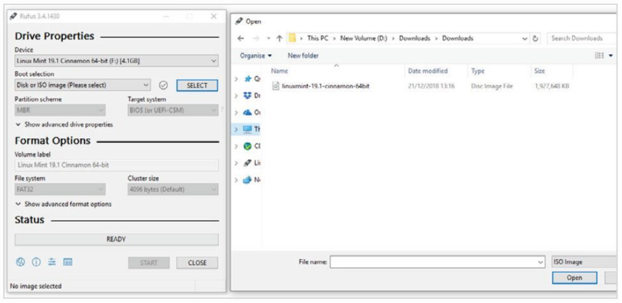
File system: FAT32 — Cluster size: 4096 bytes (Default)

Status: READY

1 device found

STEP 3

At first glance the Rufus interface can look a little confusing, don't worry though, 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 will launch a Windows Explorer window from where you can locate and select the Linux ISO.



Rufus 3.4.1430

Drive Properties

Device: Linux Mint 19.1 Cinnamon 64-bit (F:) [4.1GB]

Boot selection: Disk or ISO image (Please select)

Partition scheme: MBR — Target system: BIOS (or UEFI-CSM)

Format Options

Volume label: Linux Mint 19.1 Cinnamon 64-bit

File system: FAT32 — Cluster size: 4096 bytes (Default)

Status: READY

No image selected

Open

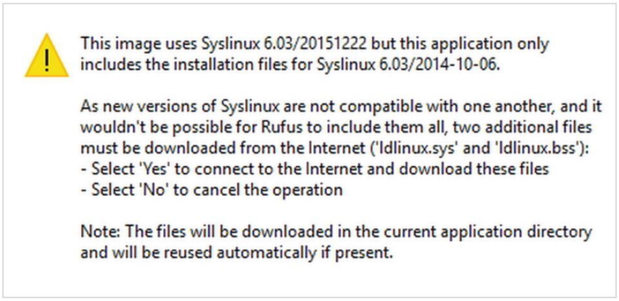
This PC › New Volume (D:) › Downloads › Downloads

Name	Date modified	Type	Size
linuxmint-19.1-cinnamon-64bit	21/12/2018 13:16	Disc Image File	1,927,648 KB

File name: — ISO Image

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.



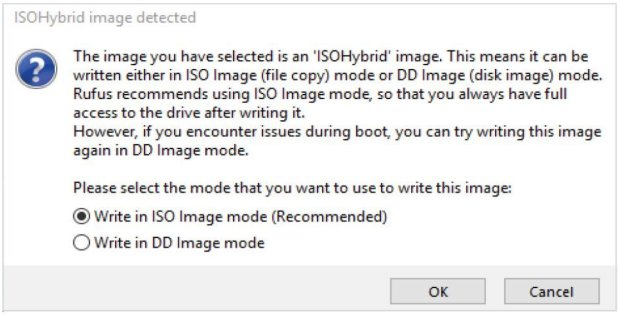
This image uses Syslinux 6.03/20151222 but this application only includes the installation files for Syslinux 6.03/2014-10-06.

As new versions of Syslinux are not compatible with one another, and it wouldn't be possible for Rufus to include them all, two additional files must be downloaded from the Internet ('ldlinux.sys' and 'ldlinux.bss'):
- Select 'Yes' to connect to the Internet and download these files
- Select 'No' to cancel the operation

Note: The files will be downloaded in the current application directory and will be reused automatically if present.

STEP 5

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



ISOHybrid image detected

The image you have selected is an 'ISOHybrid' image. This means it can be written either in ISO Image (file copy) mode or DD Image (disk image) mode. Rufus recommends using ISO Image mode, so that you always have full access to the drive after writing it.
However, if you encounter issues during boot, you can try writing this image again in DD Image mode.

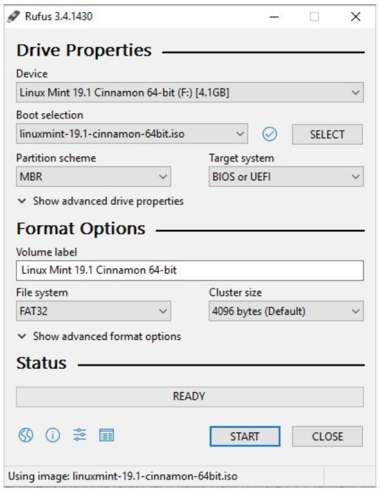
Please select the mode that you want to use to write this image:

● Write in ISO Image mode (Recommended)

○ Write in DD Image mode

STEP 6

The Linux ISO is now being transferred to the USB flash drive. The process shouldn't take too long, depending on the speed of the USB device and the PC. During the process, you may find Rufus will auto-open the USB drive in Windows Explorer, don't worry you can minimise or close it if you want. Click on the Close button once the process is complete.



Rufus 3.4.1430

Drive Properties

Device: Linux Mint 19.1 Cinnamon 64-bit (F:) [4.1GB]

Boot selection: linuxmint-19.1-cinnamon-64bit.iso

Partition scheme: MBR — Target system: BIOS or UEFI

Format Options

Volume label: Linux Mint 19.1 Cinnamon 64-bit

File system: FAT32 — Cluster size: 4096 bytes (Default)

Status: READY

Using image: linuxmint-19.1-cinnamon-64bit.iso

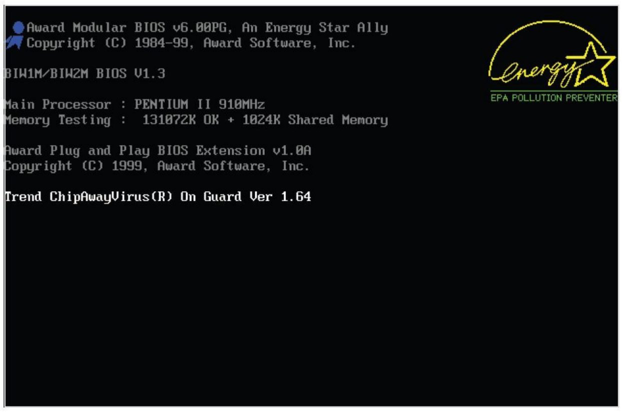
Installing Linux on a PC

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 you've just created. Let's see how that works and how you go about installing Linux from there.

UEFI BIOS

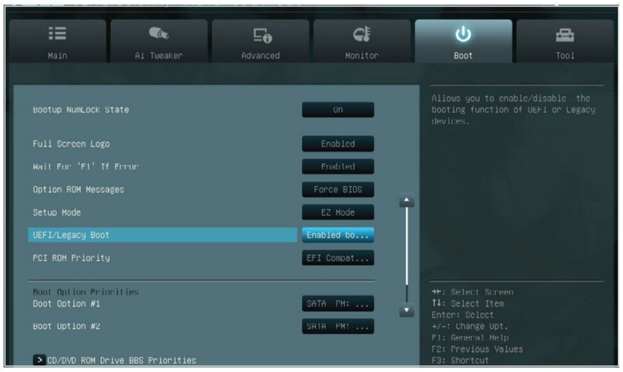
The Unified Extensible Firmware Interface (UEFI) is used to identify hardware and protect a PC during its boot-up process. It replaces the traditional BIOS, but can cause issues when installing Linux.

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; these could be, for example, **F2**, **Del** or even **F12**.



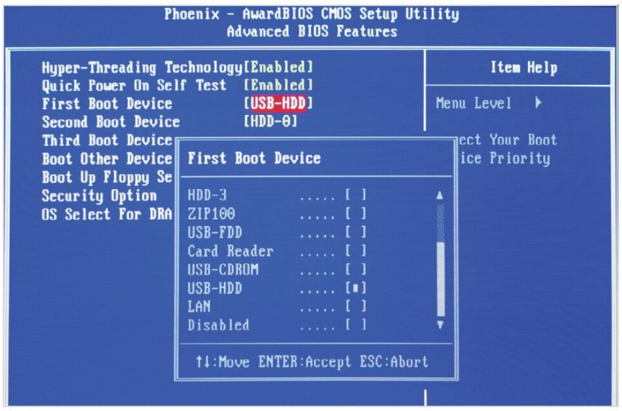
```
Award Modular BIOS v6.00PG, An Energy Star Ally
Copyright (C) 1984-99, Award Software, Inc.

BIW1M/BIW2M BIOS V1.3

Main Processor : PENTIUM II 910MHz
Memory Testing :  131072K OK + 1024K Shared Memory

Award Plug and Play BIOS Extension v1.0A
Copyright (C) 1999, Award Software, Inc.

Trend ChipAwayVirus(R) On Guard Ver 1.64
```

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'll 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.



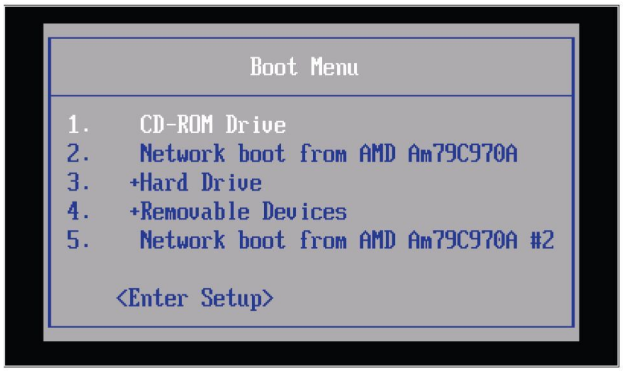
```
Bootup NumLock State            On

Full Screen Logo                Enabled
Wait For 'F1' If Error          Enabled
Option ROM Messages             Force BIOS
Setup Mode                      EZ Mode
UEFI/Legacy Boot                Enabled bo...
PCI ROM Priority                EFI Compat...

Boot Option Priorities
Boot Option #1                  SATA  PM: ...
Boot Option #2                  SATA  PM: ...

> CD/DVD ROM Drive BBS Priorities

Allows you to enable/disable the booting function of UEFI or Legacy devices.

→←: Select Screen
↑↓: Select Item
Enter: Select
+/-: Change Opt.
F1: General Help
F2: Previous Values
F3: Shortcut
```

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.



```
Phoenix - AwardBIOS CMOS Setup Utility
Advanced BIOS Features

Hyper-Threading Technology[Enabled]        Item Help
Quick Power On Self Test  [Enabled]
First Boot Device         [USB-HDD]        Menu Level  ▶
Second Boot Device        [HDD-0]
Third Boot Device                          ect Your Boot
Boot Other Device  First Boot Device       ice Priority
Boot Up Floppy Se
Security Option    HDD-3        ..... [ ]
OS Select For DRA  ZIP100       ..... [ ]
                   USB-FDD      ..... [ ]
                   Card Reader  ..... [ ]
                   USB-CDROM    ..... [ ]
                   USB-HDD      ..... [■]
                   LAN          ..... [ ]
                   Disabled     ..... [ ]

                   ↑↓:Move ENTER:Accept ESC:Abort
```

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**.

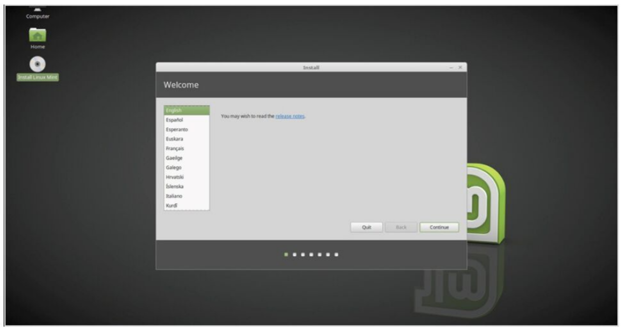


```
Boot Menu

1.   CD-ROM Drive
2.   Network boot from AMD Am79C970A
3.  +Hard Drive
4.  +Removable Devices
5.   Network boot from AMD Am79C970A #2

  <Enter Setup>
```

INSTALLING LINUX

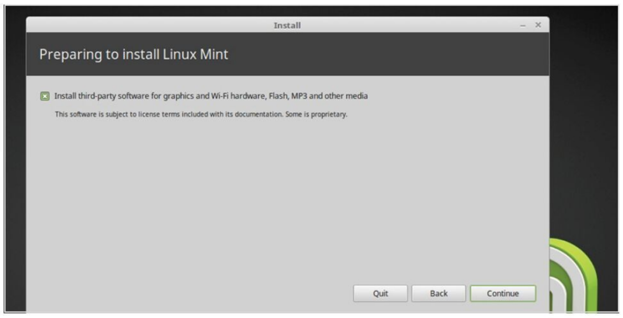
Once the Live Environment has booted, you will see the option to install the distro to your computer. Have a look around and when you're ready, look for the Install option on the desktop.

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 will display their own name, of course, but the process is the same. Click **Continue** when you're ready.



You may wish to read the release notes.

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, or 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.

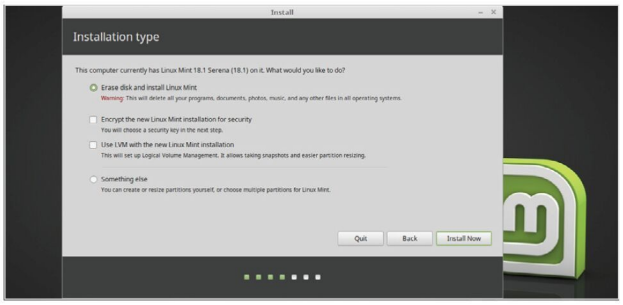


Preparing to install Linux Mint

[✓] Install third-party software for graphics and Wi-Fi hardware, Flash, MP3 and other media

This software is subject to license terms included with its documentation. Some is proprietary.

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 will completely wipe Windows 10 from your computer; make sure you have backups of all your personal files and data.**



Installation type

This computer currently has Linux Mint 18.1 Serena (18.1) on it. What would you like to do?

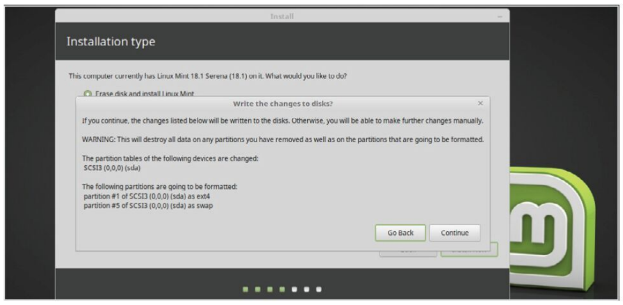
● Erase disk and install Linux Mint
Warning: This will delete all your programs, documents, photos, music, and any other files in all operating systems.

☐ Encrypt the new Linux Mint installation for security
You will choose a security key in the next step.

☐ Use LVM with the new Linux Mint installation
This will set up Logical Volume Management. It allows taking snapshots and easier partition resizing.

○ Something else
You can create or resize partitions yourself, or choose multiple partitions for Linux Mint.

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 back up your files remove the Linux disc/USB, reboot, back up and start again.



Installation type

This computer currently has Linux Mint 18.1 Serena (18.1) on it. What would you like to do?

Erase disk and install Linux Mint

Write the changes to disks?

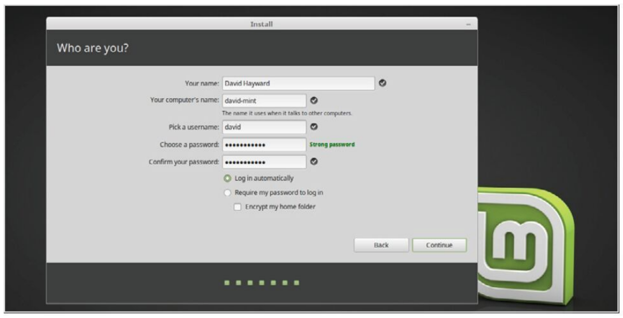
If you continue, the changes listed below will be written to the disks. Otherwise, you will be able to make further changes manually.

WARNING: This will destroy all data on any partitions you have removed as well as on the partitions that are going to be formatted.

The partition tables of the following devices are changed:
SCSI3 (0,0,0) (sda)

The following partitions are going to be formatted:
partition #1 of SCSI3 (0,0,0) (sda) as ext4
partition #5 of SCSI3 (0,0,0) (sda) as swap

STEP 5 Eventually you will be asked to set up your Linux username and password. Enter your Name to begin with, then **Computer Name** – which is the name used to identify it on the network. 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.

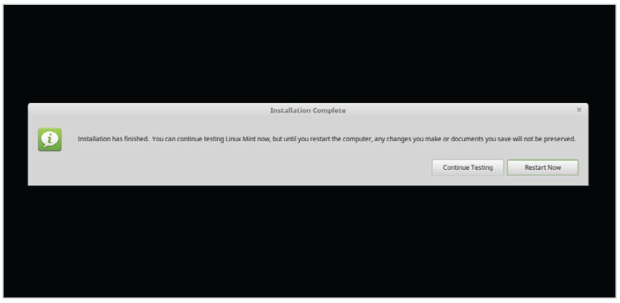


Who are you?

Your name:	David Hayward
Your computer's name:	david-mint
	The name it uses when it talks to other computers.
Pick a username:	david
Choose a password:	•••••••••••• Strong password
Confirm your password:	••••••••••••

● Log in automatically
○ Require my password to log in
☐ Encrypt my home folder

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 will 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**.



Installation Complete

Installation has finished. You can continue testing Linux Mint now, but until you restart the computer, any changes you make or documents you save will not be preserved.

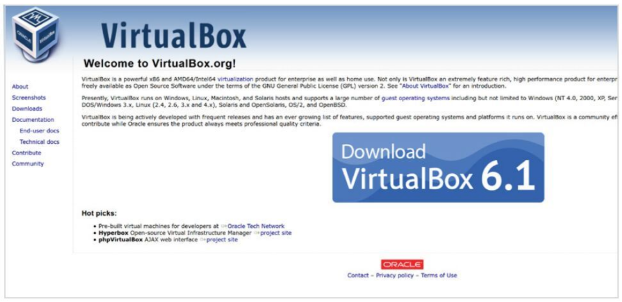
Installing a Virtual Environment

A Virtual Environment is a simulated computer system. Using a Virtual Machine, you can mimic a standard PC and install an entire operating system on 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) will take resources from your computer: memory, hard drive space, processor usage and so on. So make sure you've got 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 will take you to the main download page. Locate the correct host for your system: Windows or Mac – the Host is the current installed operating system and click the link to begin the download.


VirtualBox

Welcome to VirtualBox.org!

About
Screenshots
Downloads
Documentation
End-user docs
Technical docs
Contribute
Community

VirtualBox is a powerful x86 and AMD64/Intel64 virtualization product for enterprise as well as home use. Not only is VirtualBox an extremely feature rich, high performance product for enterpr
freely available as Open Source Software under the terms of the GNU General Public License (GPL) version 2. See "About VirtualBox" for an introduction.

Presently, VirtualBox runs on Windows, Linux, Macintosh, and Solaris hosts and supports a large number of guest operating systems including but not limited to Windows (NT 4.0, 2000, XP, Ser
DOS/Windows 3.x, Linux (2.4, 2.6, 3.x and 4.x), Solaris and OpenSolaris, OS/2, and OpenBSD.

VirtualBox is being actively developed with frequent releases and has an ever growing list of features, supported guest operating systems and platforms it runs on. VirtualBox is a community ef
contribute while Oracle ensures the product always meets professional quality criteria.

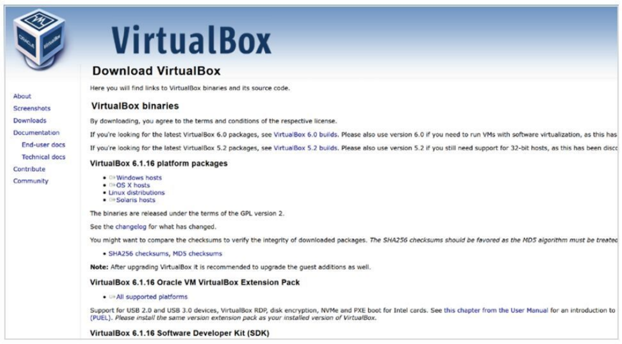
Download VirtualBox 6.1

Hot picks:

- Pre-built virtual machines for developers at Oracle Tech Network
- **Hyperbox** Open-source Virtual Infrastructure Manager project site
- **phpVirtualBox** AJAX web interface project site

ORACLE
Contact – Privacy policy – Terms of Use

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.


VirtualBox

Download VirtualBox

About
Screenshots
Downloads
Documentation
End-user docs
Technical docs
Contribute
Community

Here you will find links to VirtualBox binaries and its source code.

VirtualBox binaries

By downloading, you agree to the terms and conditions of the respective license.

If you're looking for the latest VirtualBox 6.0 packages, see VirtualBox 6.0 builds. Please also use version 6.0 if you need to run VMs with software virtualization, as this has

If you're looking for the latest VirtualBox 5.2 packages, see VirtualBox 5.2 builds. Please also use version 5.2 if you still need support for 32-bit hosts, as this has been disc

VirtualBox 6.1.16 platform packages

- Windows hosts
- OS X hosts
- Linux distributions
- Solaris hosts

The binaries are released under the terms of the GPL version 2.

See the changelog for what has changed.

You might want to compare the checksums to verify the integrity of downloaded packages. *The SHA256 checksums should be favored as the MD5 algorithm must be treate*

- SHA256 checksums, MD5 checksums

Note: After upgrading VirtualBox it is recommended to upgrade the guest additions as well.

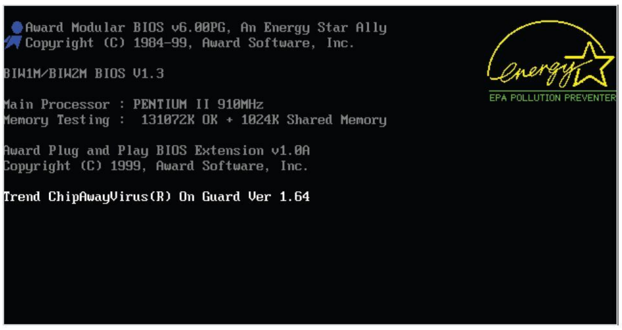
VirtualBox 6.1.16 Oracle VM VirtualBox Extension Pack

- All supported platforms

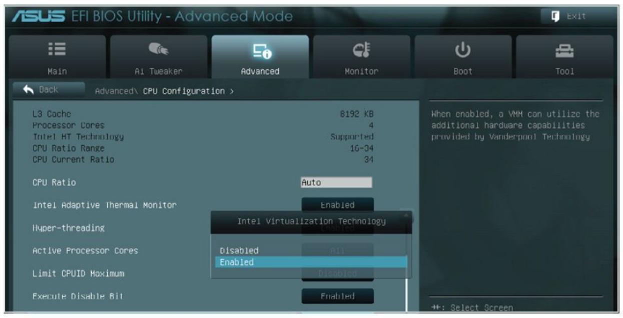
Support for USB 2.0 and USB 3.0 devices, VirtualBox RDP, disk encryption, NVMe and PXE boot for Intel cards. See this chapter from the User Manual for an introduction to
(PUEL). *Please install the same version extension pack as your installed version of VirtualBox.*

VirtualBox 6.1.16 Software Developer Kit (SDK)

STEP 3 With the correct packages downloaded, before we install anything, we 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. As the computer starts up, press the **Del**, **F2**, or whichever key is necessary to **Enter Setup**.


```
Award Modular BIOS v6.00PG, An Energy Star Ally
Copyright (C) 1984-99, Award Software, Inc.

BIW1M/BIW2M BIOS V1.3

Main Processor : PENTIUM II 910MHz
Memory Testing :  131072K OK + 1024K Shared Memory

Award Plug and Play BIOS Extension v1.0A
Copyright (C) 1999, Award Software, Inc.

Trend ChipAwayVirus(R) On Guard Ver 1.64
```

STEP 4 As each BIOS is laid out differently, 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, 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.

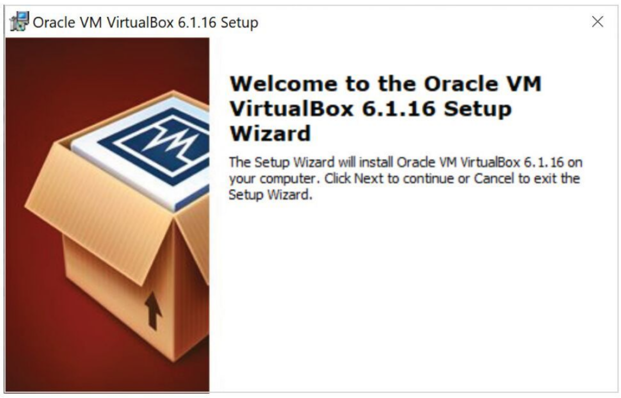


```
Advanced\ CPU Configuration >

L3 Cache                                   8192 KB
Processor Cores                                  4
Intel HT Technology                      Supported
CPU Ratio Range                              16-34
CPU Current Ratio                               34

CPU Ratio                               Auto

Intel Adaptive Thermal Monitor          Enabled

Hyper-threading         Intel Virtualization Technology

Active Processor Cores  Disabled
                        Enabled
Limit CPUID Maximum

Execute Disable Bit                     Enabled
```

When enabled, a VMM can utilize the additional hardware capabilities provided by Vanderpool Technology

→←: Select Screen

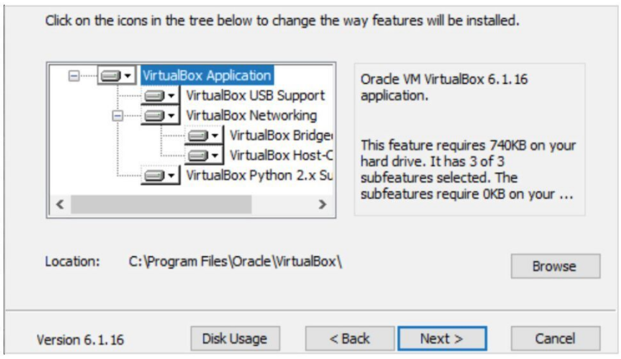
STEP 5 With the computer back up and running, locate the downloaded main VirtualBox application and double-click to begin the installation process. Click Next to continue, when you're ready.


Oracle VM VirtualBox 6.1.16 Setup

Welcome to the Oracle VM VirtualBox 6.1.16 Setup Wizard

The Setup Wizard will install Oracle VM VirtualBox 6.1.16 on your computer. Click Next to continue or Cancel to exit the Setup Wizard.

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. Also, make sure that all the icons in the VirtualBox feature tree are selected and none of them have a red X next to them. Click Next to move on.


Click on the icons in the tree below to change the way features will be installed.

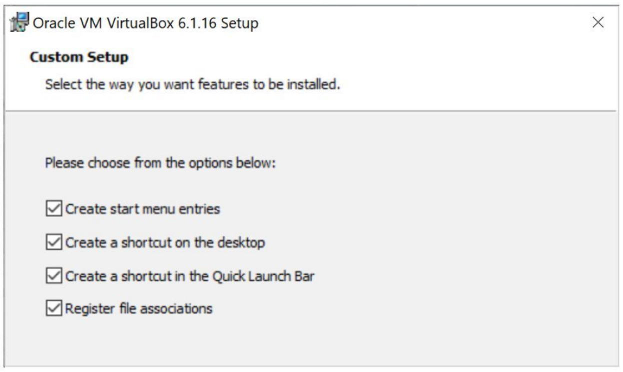
- VirtualBox Application
  - VirtualBox USB Support
  - VirtualBox Networking
    - VirtualBox Bridge
    - VirtualBox Host-C
  - VirtualBox Python 2.x Su

Oracle VM VirtualBox 6.1.16 application.

This feature requires 740KB on your hard drive. It has 3 of 3 subfeatures selected. The subfeatures require 0KB on your ...

Location: C:\Program Files\Oracle\VirtualBox\

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


Oracle VM VirtualBox 6.1.16 Setup

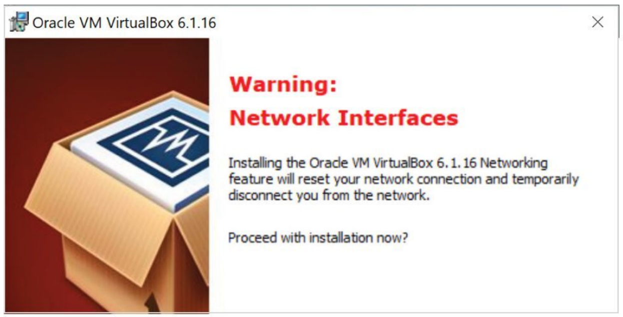
Custom Setup

Select the way you want features to be installed.

Please choose from the options below:

☑ Create start menu entries
☑ Create a shortcut on the desktop
☑ Create a shortcut in the Quick Launch Bar
☑ Register file associations

STEP 8 When installing VirtualBox, your network connection will be disabled for a very brief period. This is due to VirtualBox creating a linked, virtual network connection so that any VM installed will be 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.

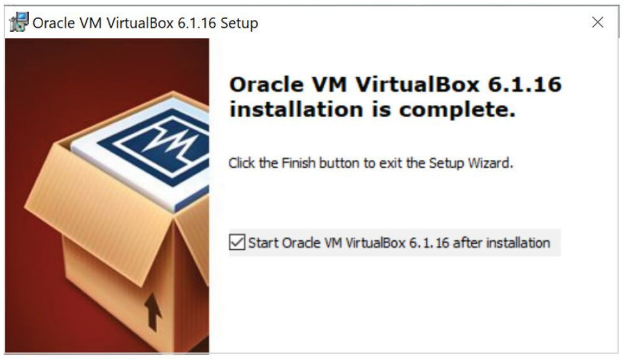

Oracle VM VirtualBox 6.1.16

Warning:
Network Interfaces

Installing the Oracle VM VirtualBox 6.1.16 Networking feature will reset your network connection and temporarily disconnect you from the network.

Proceed with installation now?

STEP 9 You'll probably be asked by Windows to accept a security notification, click Yes for this and next 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.

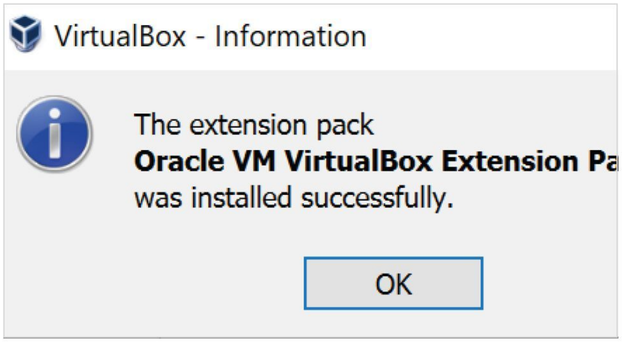

Oracle VM VirtualBox 6.1.16 Setup

Oracle VM VirtualBox 6.1.16 installation is complete.

Click the Finish button to exit the Setup Wizard.

☑ Start Oracle VM VirtualBox 6.1.16 after installation

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'll eventually receive a message to install it; click Install to begin the process, scroll down the next screen to accept the agreement and click 'I Agree'.


VirtualBox - Information

The extension pack
Oracle VM VirtualBox Extension Pa
was installed successfully.

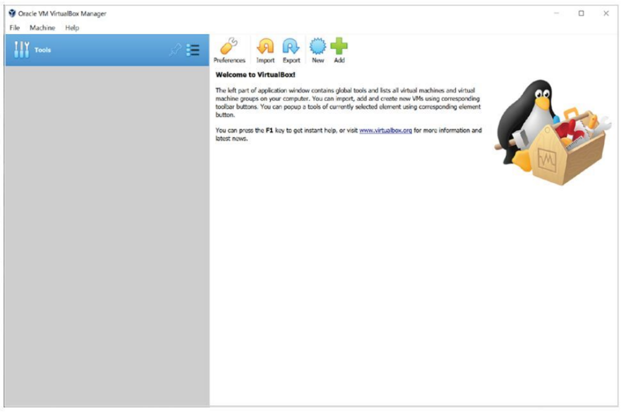
Installing Linux in a Virtual Environment

With Oracle's VirtualBox now up and running, the next task is to create the Virtual Machine (VM) environment into which you will 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, we'll setup a VM adequate to run the excellent Linux Mint, and perform well.

STEP 1 With VirtualBox open, click on the **New** icon in the top-middle of the right-hand panel of the app. This will open the new VM Wizard.

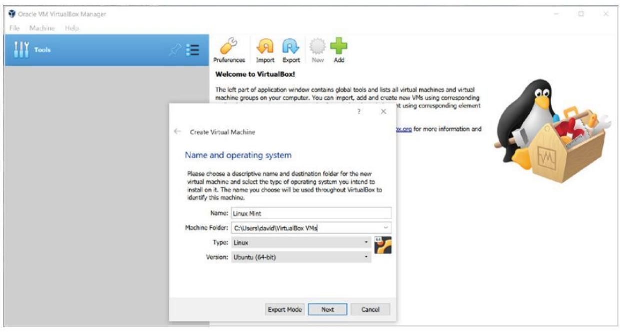

Oracle VM VirtualBox Manager
File Machine Help
Tools
Preferences Import Export New Add

Welcome to VirtualBox!

The left part of application window contains global tools and lists all virtual machines and virtual machine groups on your computer. You can import, add and create new VMs using corresponding toolbar buttons. You can popup a tools of currently selected element using corresponding element button.

You can press the **F1** key to get instant help, or visit www.virtualbox.org for more information and latest news.

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.

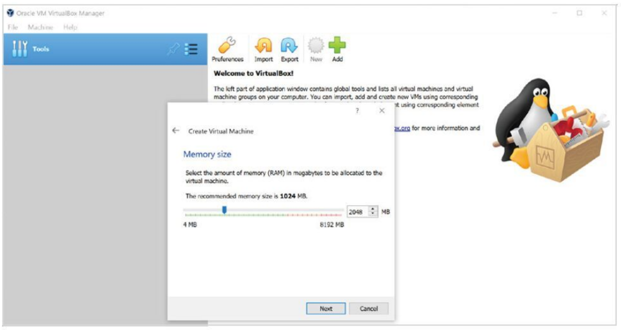


Name and operating system

Please choose a descriptive name and destination folder for the new virtual machine and select the type of operating system you intend to install on it. The name you choose will be used throughout VirtualBox to identify this machine.

Name: Linux Mint
Machine Folder: C:\Users\david\VirtualBox VMs
Type: Linux
Version: Ubuntu (64-bit)

STEP 3 The next section will define the amount of system memory, or RAM, the VM has allocated. Remember this amount will be taken from the available memory installed in your computer, so don't give the VM too much. For example, we have 8GB of memory installed and we're giving 2GB (2048MB) to the VM. When you're ready, click **Next** to continue.



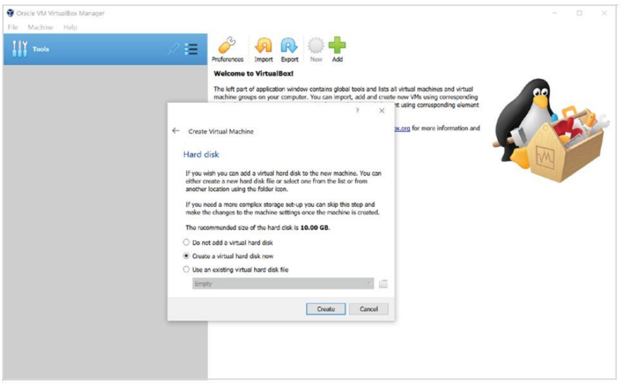
Memory size

Select the amount of memory (RAM) in megabytes to be allocated to the virtual machine.

The recommended memory size is **1024** MB.

2048 MB

STEP 4 This section is where you'll start to create the virtual hard disk that the VM will use 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.


Oracle VM VirtualBox Manager
File Machine Help
Tools
Preferences Import Export New Add
Create Virtual Machine
Create
Cancel

Hard disk

If you wish you can add a virtual hard disk to the new machine. You can either create a new hard disk file or select one from the list or from another location using the folder icon.

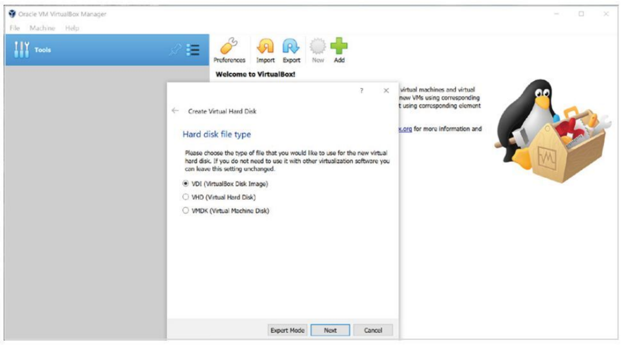
If you need a more complex storage set-up you can skip this step and make the changes to the machine settings once the machine is created.

The recommended size of the hard disk is **10.00 GB**.

- Do not add a virtual hard disk
- [✓ Create a virtual hard disk now]
- Use an existing virtual hard disk file

Empty

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**.

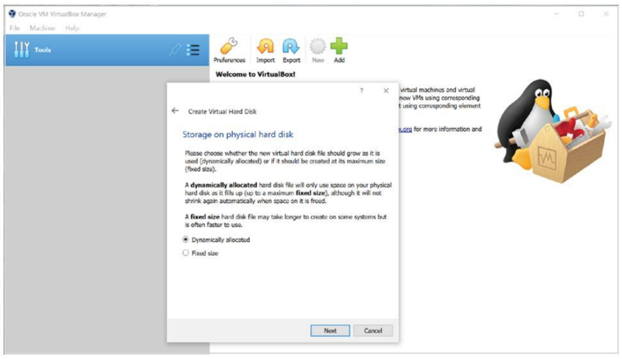

Create Virtual Hard Disk

Hard disk file type

Please choose the type of file that you would like to use for the new virtual hard disk. If you do not need to use it with other virtualization software you can leave this setting unchanged.

- [✓ VDI (VirtualBox Disk Image)]
- VHD (Virtual Hard Disk)
- VMDK (Virtual Machine Disk)

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. Basically, a Dynamically Allocated virtual hard disk is a more flexible storage management option. It won't take up much space within your physical hard disk to begin with either. Ensure **Dynamically Allocated** is selected, and click **Next**.


Create Virtual Hard Disk

Storage on physical hard disk

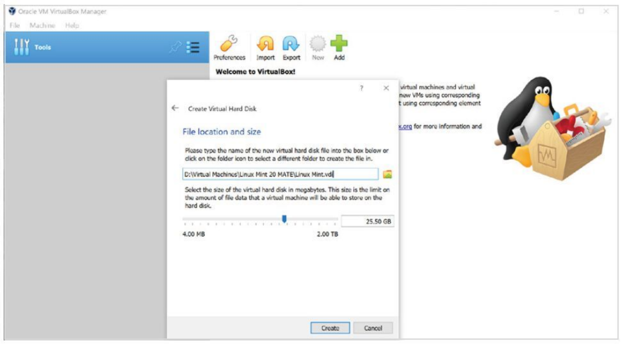
Please choose whether the new virtual hard disk file should grow as it is used (dynamically allocated) or if it should be created at its maximum size (fixed size).

A **dynamically allocated** hard disk file will only use space on your physical hard disk as it fills up (up to a maximum **fixed size**), although it will not shrink again automatically when space on it is freed.

A **fixed size** hard disk file may take longer to create on some systems but is often faster to use.

- [✓ Dynamically allocated]
- Fixed size

STEP 7 The virtual hard disk will be 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.


Create Virtual Hard Disk

File location and size

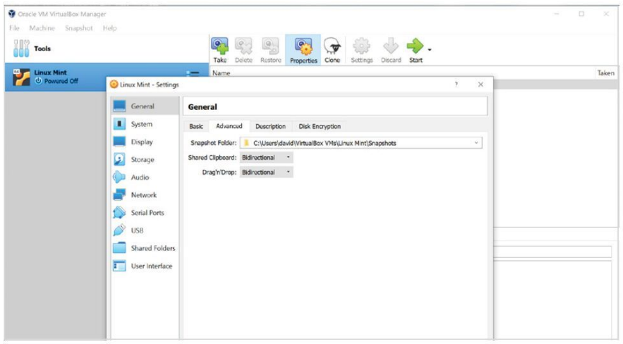
Please type the name of the new virtual hard disk file into the box below or click on the folder icon to select a different folder to create the file in.

D:\Virtual Machines\Linux Mint 20 MATE\Linux Mint.vdi

Select the size of the virtual hard disk in megabytes. This size is the limit on the amount of file data that a virtual machine will be able to store on the hard disk.

25.50 GB (4.00 MB – 2.00 TB)

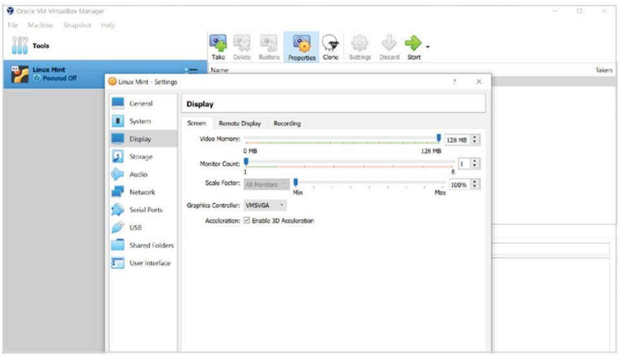
STEP 8 After clicking Create the initial setup of the VM is complete; you should now be looking at the newly created VM within the VirtualBox application. Before you begin though, click the **Settings** button from the top of the right-hand panel, and within the General section click the **Advanced**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


Linux Mint - Settings

General — Advanced

Snapshot Folder: C:\Users\david\VirtualBox VMs\Linux Mint\Snapshots
Shared Clipboard: Bidirectional
Drag'n'Drop: Bidirectional

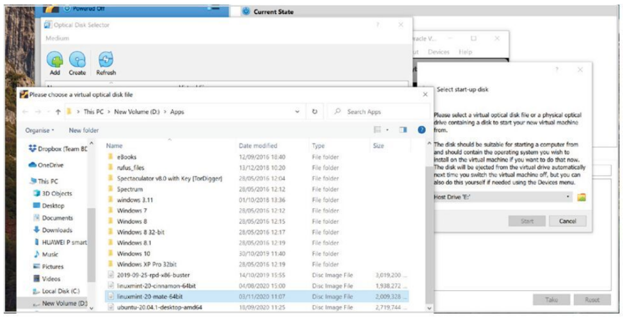
STEP 9 Follow that by clicking on the **System** section, then the **Processor** tab. Depending on your CPU allocate as many cores as you can without detriment to your host system; we've opted for two CPUs. Now click on the **Display** section, slide the **Video Memory** up to the maximum, and tick 'Enable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


Linux Mint - Settings

Display — Screen

Video Memory: 128 MB (0 MB – 128 MB)
Monitor Count: 1 (1 – 8)
Scale Factor: All Monitors 100% (Min – Max)
Graphics Controller: VMSVGA
Acceleration: [✓ Enable 3D Acceleration]

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


Please choose a virtual optical disk file

This PC > New Volume (D:) > Apps

Name	Date modified	Type	Size
eBooks	12/09/2016 18:40	File folder	
rufus_files	13/12/2018 10:20	File folder	
Spectaculator v8.0 with Key [TorDigger]	28/05/2016 12:04	File folder	
Spectrum	28/05/2016 12:12	File folder	
windows 3.11	01/10/2018 13:36	File folder	
Windows 7	28/05/2016 12:12	File folder	
Windows 8	28/05/2016 12:15	File folder	
Windows 8 32-bit	28/05/2016 12:17	File folder	
Windows 8.1	28/05/2016 12:19	File folder	
Windows 10	30/10/2019 11:40	File folder	
Windows XP Pro 32bit	28/05/2016 12:19	File folder	
2019-09-25-rpd-x86-buster	14/10/2019 15:55	Disc Image File	3,019,200 ...
linuxmint-20-cinnamon-64bit	04/08/2020 15:00	Disc Image File	1,938,272 ...
linuxmint-20-mate-64bit	03/11/2020 11:07	Disc Image File	2,009,328 ...
ubuntu-20.04.1-desktop-amd64	18/09/2020 11:25	Disc Image File	2,719,744 ...

Select start-up disk

Please select a virtual optical disk file or a physical optical drive containing a disk to start your new virtual machine from.

The disk should be suitable for starting a computer from and should contain the operating system you wish to install on the virtual machine if you want to do that now. The disk will be ejected from the virtual drive automatically next time you switch the virtual machine off, but you can also do this yourself if needed using the Devices menu.

Host Drive 'E:'

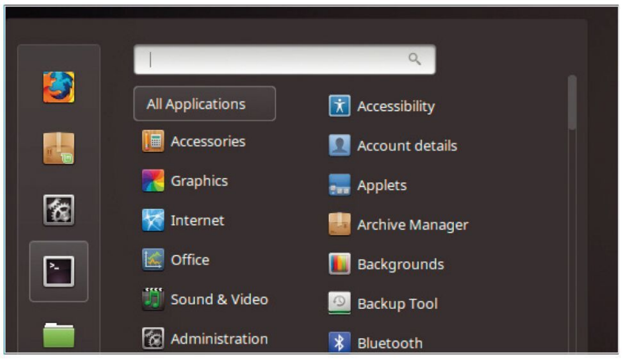
Getting Ready to Code in Linux

Coding in Linux mostly happens in the Terminal or the Command Line. While it can be a scary looking place to begin with, the Terminal is an extremely powerful environment. Before you can start to code, it's best to master the Terminal.

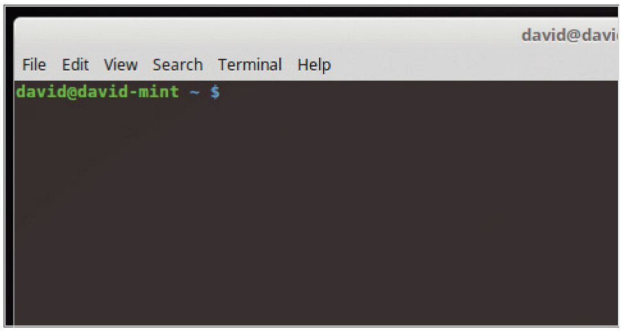
TAKING COMMAND

The command line is at the core of Linux and when you program with it, this is called scripting. These are self-contained programs designed to be run in the Terminal.

STEP 1 The Terminal is where you begin your journey with Linux, through the command line and thus any scripting from. In Linux Mint, it can be accessed by clicking on the Menu followed by the Terminal icon in the panel, or entering 'Terminal' into the search bar.

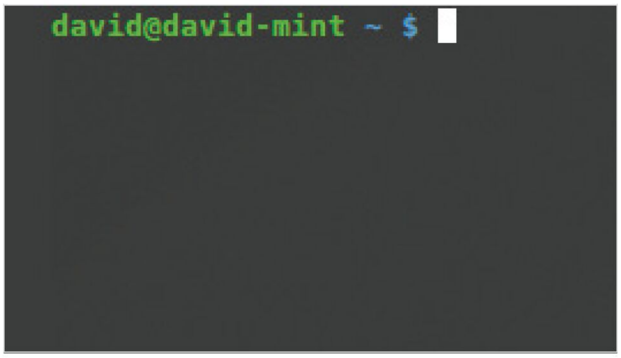


STEP 2 The Terminal will give you access to the Linux Mint Shell, called BASH; this gives you access to the underlying operating system, which is why scripting is such a powerful language to learn and use. Everything in Mint, and Linux as a whole, including the desktop and GUI, is a module running from the command line.



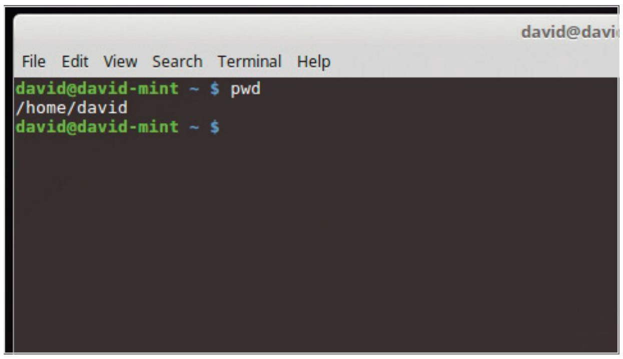
```
david@david-mint ~ $
```

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



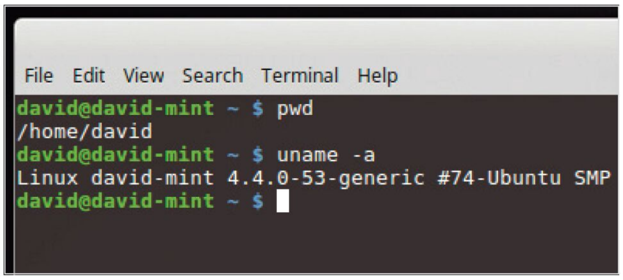
```
david@david-mint ~ $
```

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



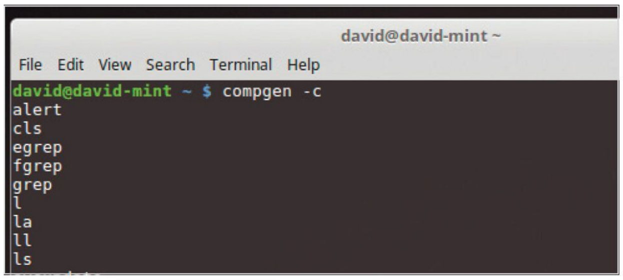
```
david@david-mint ~ $ pwd
/home/david
david@david-mint ~ $
```

STEP 5 All the commands you enter will 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. Now type: `uname -a` and press Enter. This will display information regarding Linux Mint. In scripting, you can use all the Linux command-line commands within your own scripts.



```
david@david-mint ~ $ pwd
/home/david
david@david-mint ~ $ uname -a
Linux david-mint 4.4.0-53-generic #74-Ubuntu SMP
david@david-mint ~ $
```

STEP 6 The list of available Linux commands is vast, with some simply returning the current working directory, while others are capable of deleting the entire system in an instant. Getting to know the commands is part of learning how to script. By using the wrong command, you could end up wiping your computer. Type `compgen -c` to view the available commands.

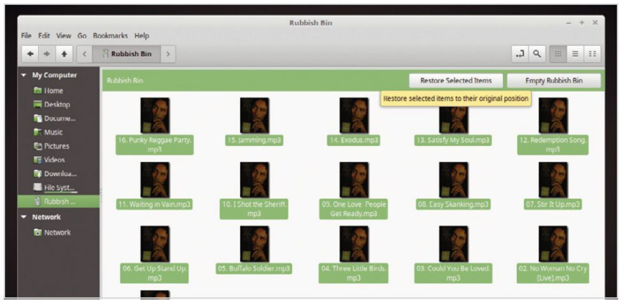


```
david@david-mint ~ $ compgen -c
alert
cls
egrep
fgrep
grep
l
la
ll
ls
```

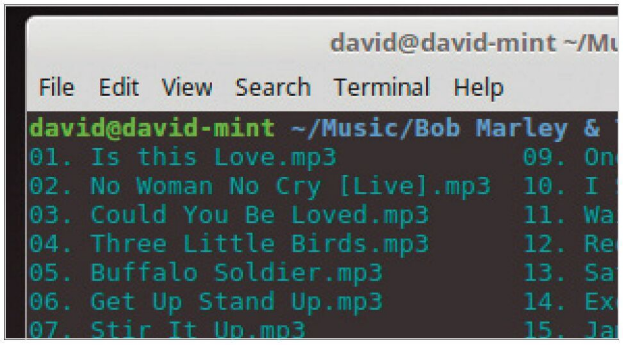
HERE BE DRAGONS!

There's an urban myth on the Internet that an employee at Disney Pixar nearly ruined the animated movie Toy Story by inadvertently entering the wrong Linux command and deleting the entire system the film was stored on.

STEP 1 Having access to the Terminal means you're bypassing the GUI desktop method of working with the system. The Terminal is a far more powerful environment than the desktop, which has several safeguards in place in case you accidentally delete all your work, such as Rubbish Bin to recover deleted files.

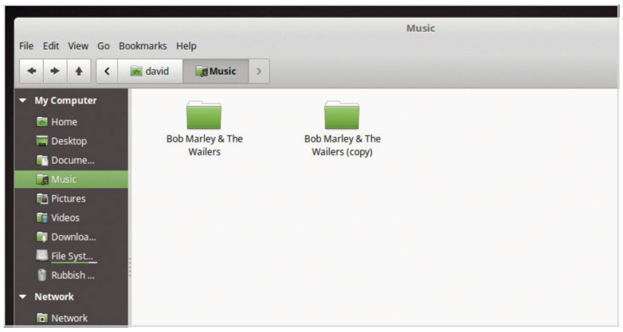


STEP 2 However, the Terminal doesn't offer that luxury. If you were to access a folder with files within via the Terminal and then enter the command: `rm *.*`, all the files in that folder would be instantly deleted. They won't appear in the Rubbish Bin either, they're gone for good.

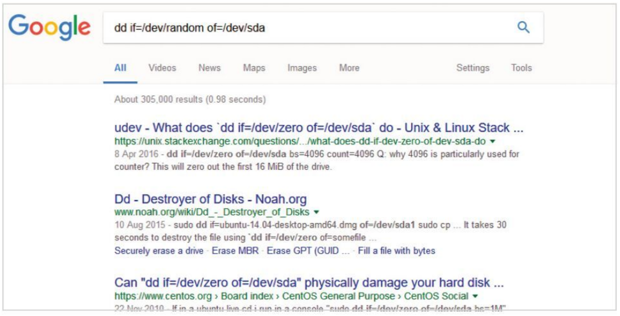


```
david@david-mint ~/Music/Bob Marley &
01. Is this Love.mp3              09. On
02. No Woman No Cry [Live].mp3    10. I
03. Could You Be Loved.mp3        11. Wa
04. Three Little Birds.mp3        12. Re
05. Buffalo Soldier.mp3           13. Sa
06. Get Up Stand Up.mp3           14. Ex
07. Stir It Up.mp3                15. Ja
```

STEP 3 Therefore it's always a good idea to work in the Terminal using a two-pronged approach. First, use the desktop to make regular backups of the folders you're working in when in the Terminal. This way, should anything go wrong, there's a quick and handy backup waiting for you.



STEP 4 Second, research before blindly entering a command you've seen on the Internet. If you see the command: `sudo dd if=/dev/random of=/dev/sda` and use it in a script, you'll soon come to regret the action as the command will wipe the entire hard drive and fill it with random data. Take a moment to Google the command and see what it does.



About 305,000 results (0.98 seconds)

udev - What does `dd if=/dev/zero of=/dev/sda` do - Unix & Linux Stack ...
https://unix.stackexchange.com/questions/.../what-does-dd-if-dev-zero-of-dev-sda-do
8 Apr 2016 - dd if=/dev/zero of=/dev/sda bs=4096 count=4096 Q: why 4096 is particularly used for counter? This will zero out the first 16 MiB of the drive.

Dd - Destroyer of Disks - Noah.org
www.noah.org/wiki/Dd_-_Destroyer_of_Disks
10 Aug 2015 - sudo dd if=ubuntu-14.04-desktop-amd64.dmg of=/dev/sda1 sudo cp ... It takes 30 seconds to destroy the file using `dd if=/dev/zero of=somefile ...
Securely erase a drive · Erase MBR · Erase GPT (GUID ... · Fill a file with bytes

Can "dd if=/dev/zero of=/dev/sda" physically damage your hard disk ...
https://www.centos.org › Board index › CentOS General Purpose › CentOS Social
22 Nov 2010 - If in a ubuntu live cd i run in a console "sudo dd if=/dev/zero of=/dev/sda bs=1M"

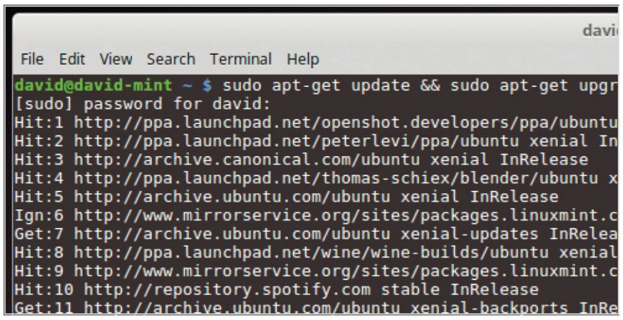
Creating Bash Scripts – Part 1

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

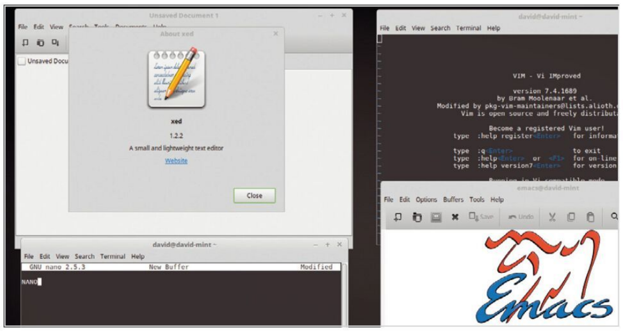
A Bash script is simply a series of commands that Mint will run through to complete a certain task. They can be simple or remarkably complex, it all depends on the situation.

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`.



```
david@david-mint ~ $ sudo apt-get update && sudo apt-get upgr
[sudo] password for david:
Hit:1 http://ppa.launchpad.net/openshot.developers/ppa/ubuntu
Hit:2 http://ppa.launchpad.net/peterlevi/ppa/ubuntu xenial In
Hit:3 http://archive.canonical.com/ubuntu xenial InRelease
Hit:4 http://ppa.launchpad.net/thomas-schiex/blender/ubuntu x
Hit:5 http://archive.ubuntu.com/ubuntu xenial InRelease
Ign:6 http://www.mirrorservice.org/sites/packages.linuxmint.c
Get:7 http://archive.ubuntu.com/ubuntu xenial-updates InRelea
Hit:8 http://ppa.launchpad.net/wine/wine-builds/ubuntu xenial
Hit:9 http://www.mirrorservice.org/sites/packages.linuxmint.c
Hit:10 http://repository.spotify.com stable InRelease
Get:11 http://archive.ubuntu.com/ubuntu xenial-backports InRe
```

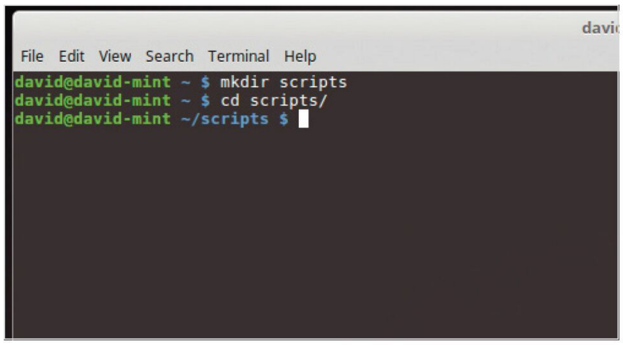
STEP 2 There are several text editors we can use to create a Bash script: Xed, Vi, Nano, Vim, GNU Emacs and so on. In the end it all comes down to personal preference. Our use of Xed is purely due to making it easier to read the script in the screenshots you see below.


Unsaved Document 1
File Edit View Search Tools Documents Help
Unsaved Docu
About xed
xed
1.2.2
A small and lightweight text editor
Website
Close
david@david-mint ~
File Edit View Search Terminal Help
emacs@david-mint
File Edit Options Buffers Tools Help
Save
Undo
Emacs

```
 GNU nano 2.5.3              New Buffer                  Modified

NANO
```

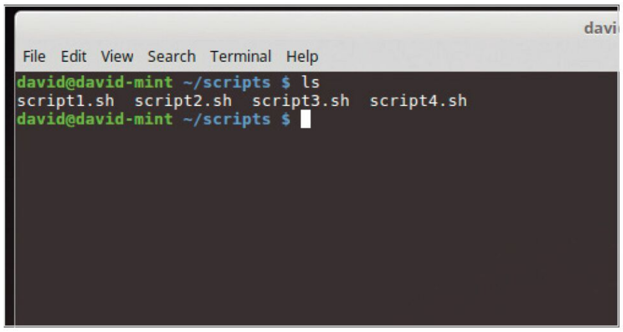
```
                 VIM - Vi IMproved

                 version 7.4.1689
            by Bram Moolenaar et al.
   Modified by pkg-vim-maintainers@lists.alioth.
      Vim is open source and freely distribut

            Become a registered Vim user!
  type  :help register<Enter>   for informa

  type  :q<Enter>               to exit
  type  :help<Enter>  or  <F1>  for on-line
  type  :help version7<Enter>   for version
```

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.



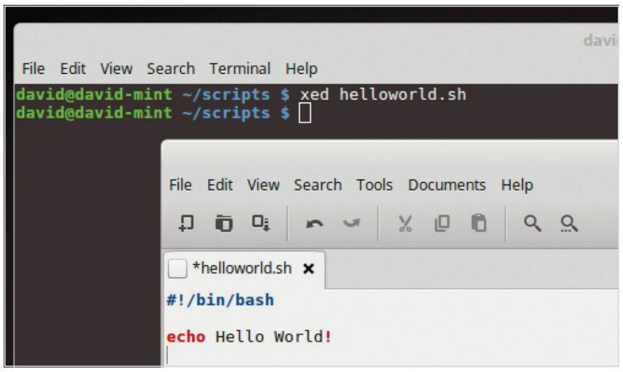
```
david@david-mint ~ $ mkdir scripts
david@david-mint ~ $ cd scripts/
david@david-mint ~/scripts $
```

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.



```
david@david-mint ~/scripts $ ls
script1.sh  script2.sh  script3.sh  script4.sh
david@david-mint ~/scripts $
```

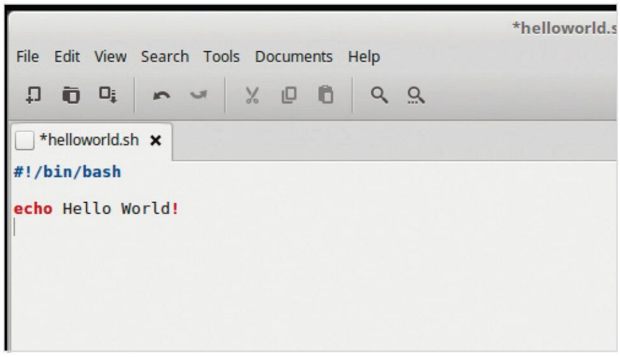
STEP 5 Let's start with a simple script to output something to the Terminal. Enter `xed helloworld.sh`. This will launch Xed and create a file called helloworld.sh. In Xed, enter the following: `#!/bin/bash`, then on a new line: `echo Hello World!`.


File Edit View Search Terminal Help
File Edit View Search Tools Documents Help
*helloworld.sh

```
david@david-mint ~/scripts $ xed helloworld.sh
david@david-mint ~/scripts $ 
```

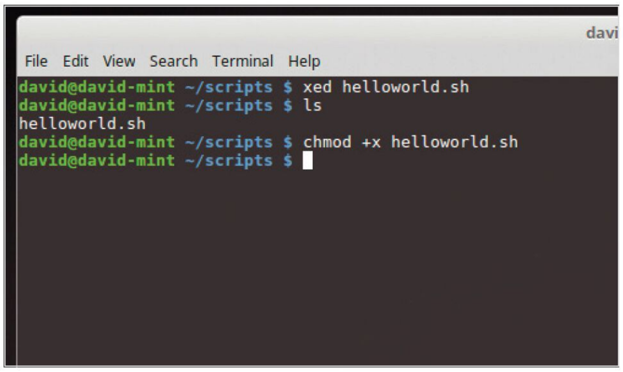
```
#!/bin/bash

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.



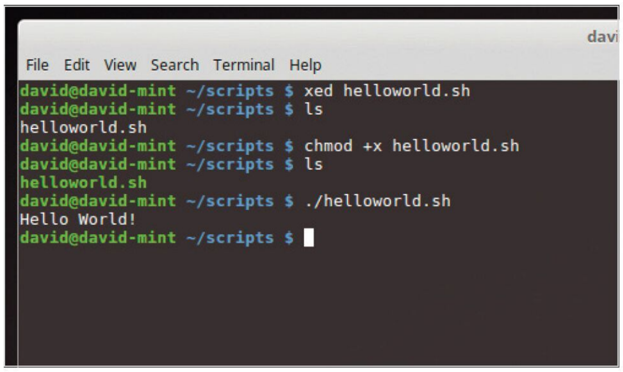
```
#!/bin/bash

echo Hello World!
```

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.



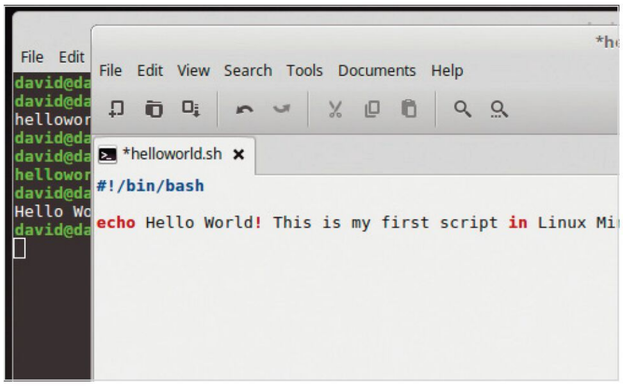
```
david@david-mint ~/scripts $ xed helloworld.sh
david@david-mint ~/scripts $ ls
helloworld.sh
david@david-mint ~/scripts $ chmod +x helloworld.sh
david@david-mint ~/scripts $ 
```

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`.



```
david@david-mint ~/scripts $ xed helloworld.sh
david@david-mint ~/scripts $ ls
helloworld.sh
david@david-mint ~/scripts $ chmod +x helloworld.sh
david@david-mint ~/scripts $ ls
helloworld.sh
david@david-mint ~/scripts $ ./helloworld.sh
Hello World!
david@david-mint ~/scripts $ 
```

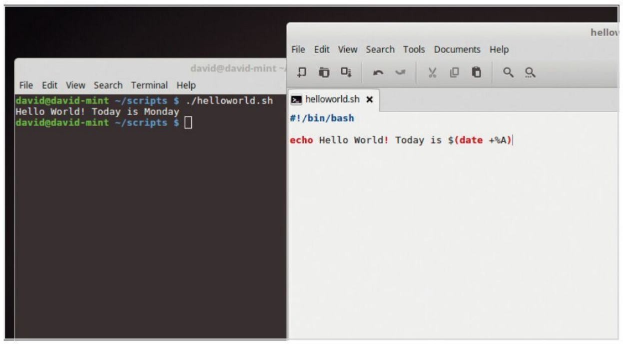
STEP 9 Although it's not terribly exciting, the words 'Hello World!' should now be displayed in the Terminal. The echo command is responsible for outputting the words after it in the Terminal, as we move on you can make the echo command output to other sources.



```
david@da
david@da
hellowor
david@da
david@da
hellowor
david@da
Hello Wo
david@da
```

```
#!/bin/bash

echo Hello World! This is my first script in Linux Mi
```

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.


hellov
File Edit View Search Tools Documents Help
helloworld.sh
david@david-mint ~
File Edit View Search Terminal Help

```
david@david-mint ~/scripts $ ./helloworld.sh
Hello World! Today is Monday
david@david-mint ~/scripts $ 
```

```
#!/bin/bash

echo Hello World! Today is $(date +%A)
```

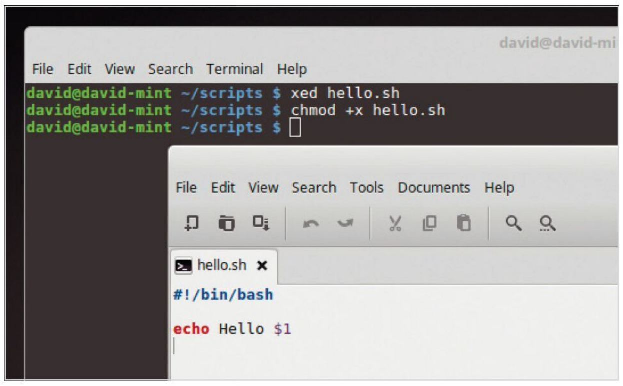
Creating Bash Scripts – Part 2

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.

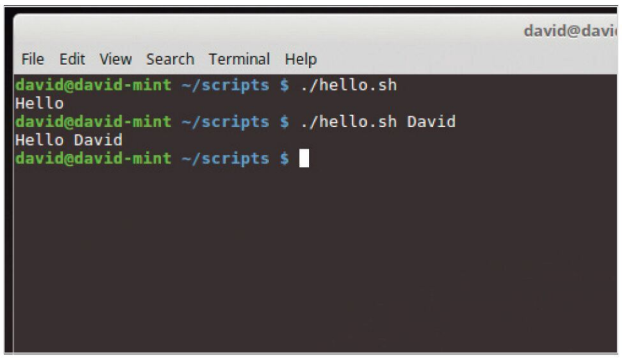
STEP 1 Let's start by creating a new script called hello.sh; `xed hello.sh`. In it enter: `#!/bin/bash`, then, echo `Hello $1`. Save the file and exit Xed. Back in the Terminal make the script executable with: `chmod +x hello.sh`.



```
david@david-mint ~/scripts $ xed hello.sh
david@david-mint ~/scripts $ chmod +x hello.sh
david@david-mint ~/scripts $ 
```

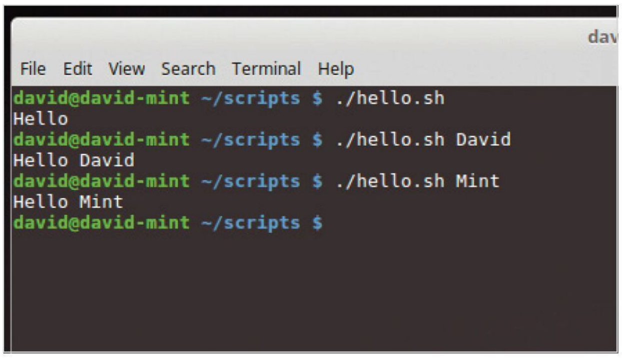
```
#!/bin/bash

echo Hello $1
```

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`.



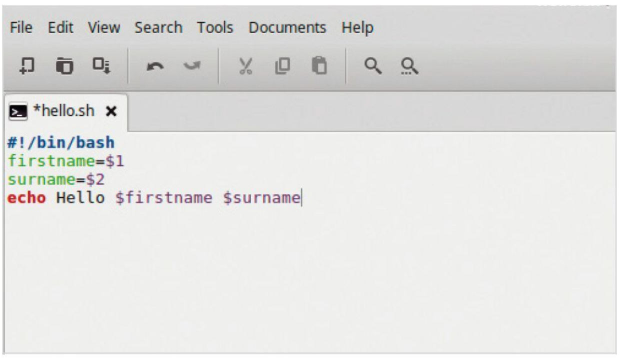
```
david@david-mint ~/scripts $ ./hello.sh
Hello
david@david-mint ~/scripts $ ./hello.sh David
Hello David
david@david-mint ~/scripts $ 
```

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`.



```
david@david-mint ~/scripts $ ./hello.sh
Hello
david@david-mint ~/scripts $ ./hello.sh David
Hello David
david@david-mint ~/scripts $ ./hello.sh Mint
Hello Mint
david@david-mint ~/scripts $
```

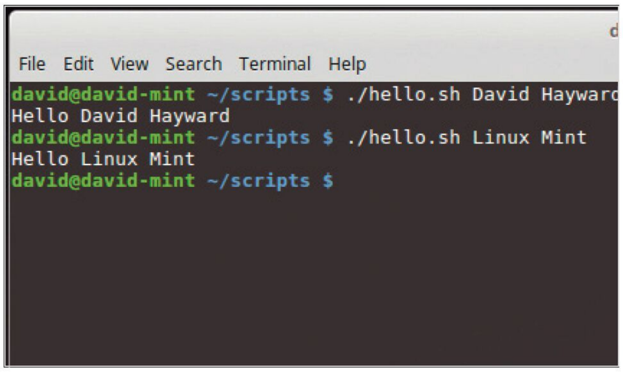
STEP 4 You can even rename variables. Modify the hello.sh script with the following: `firstname=$1, surname=$2, echo Hello $firstname $surname`. Putting each statement on a new line. Save the script and exit back into the Terminal.



```
#!/bin/bash
firstname=$1
surname=$2
echo Hello $firstname $surname
```

STEP 5

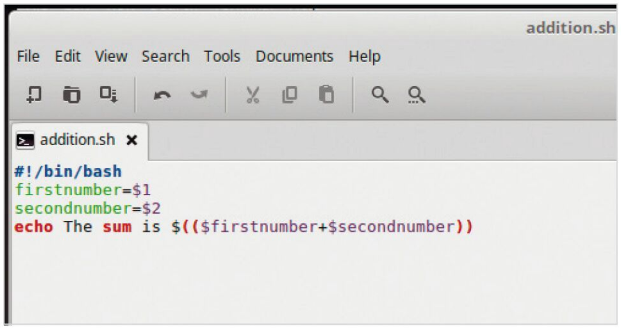
When you run the script now you can use two custom variables: `./hello.sh David Hayward`. Naturally change the two variables with your own name; unless you're also called David Hayward. At the moment we're just printing the contents, so let's expand the two-variable use a little.



```
david@david-mint ~/scripts $ ./hello.sh David Haywarc
Hello David Hayward
david@david-mint ~/scripts $ ./hello.sh Linux Mint
Hello Linux Mint
david@david-mint ~/scripts $
```

STEP 6

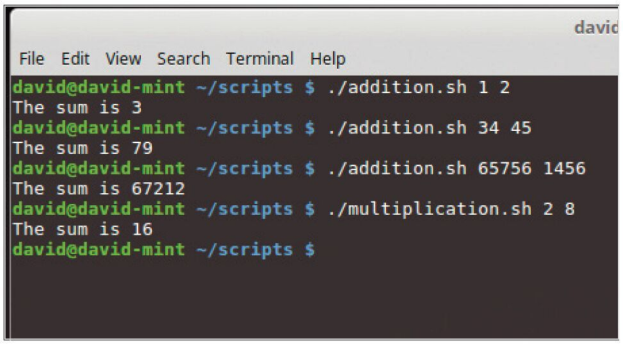
Create a new script called `addition.sh`, using the same format as the hello.sh script, but changing the variable names. Here we've added `firstnumber` and `secondnumber`, and used the echo command to output some simple arithmetic by placing an integer expression, `echo The sum is $(($firstnumber+$secondnumber))`.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



```
#!/bin/bash
firstnumber=$1
secondnumber=$2
echo The sum is $(($firstnumber+$secondnumber))
```

STEP 7

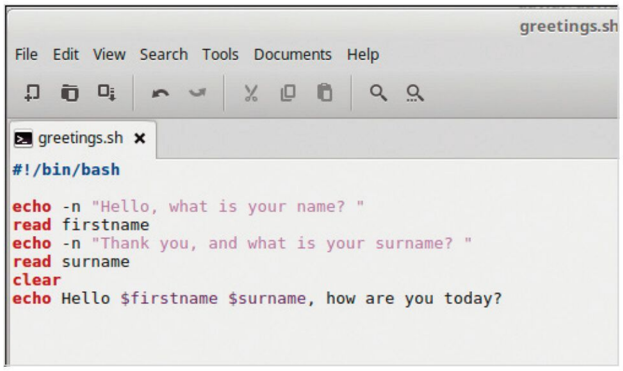
When you now run the addition.sh script we can enter two numbers: `./addition.sh 1 2`. The result will hopefully be 3, with the Terminal displaying 'The sum is 3'. Try it with a few different numbers and see what happens. See also if you can alter the script and rename it do multiplication, and subtraction.



```
david@david-mint ~/scripts $ ./addition.sh 1 2
The sum is 3
david@david-mint ~/scripts $ ./addition.sh 34 45
The sum is 79
david@david-mint ~/scripts $ ./addition.sh 65756 1456
The sum is 67212
david@david-mint ~/scripts $ ./multiplication.sh 2 8
The sum is 16
david@david-mint ~/scripts $
```

STEP 8

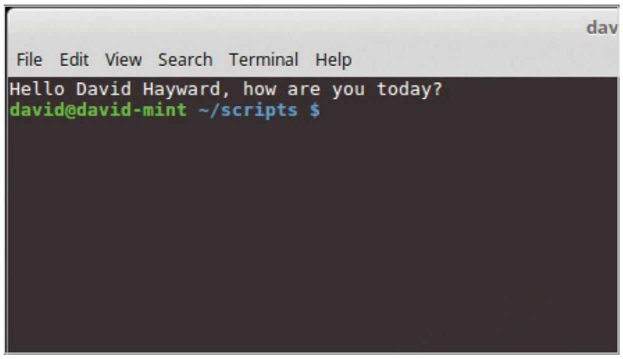
Let's expand things further. Create a new script called `greetings.sh`. Enter the scripting as below in the screenshot, save it and make it executable with the `chmod` command. You can see that there are a few new additions to the script now.



```
#!/bin/bash

echo -n "Hello, what is your name? "
read firstname
echo -n "Thank you, and what is your surname? "
read surname
clear
echo Hello $firstname $surname, how are you today?
```

STEP 9

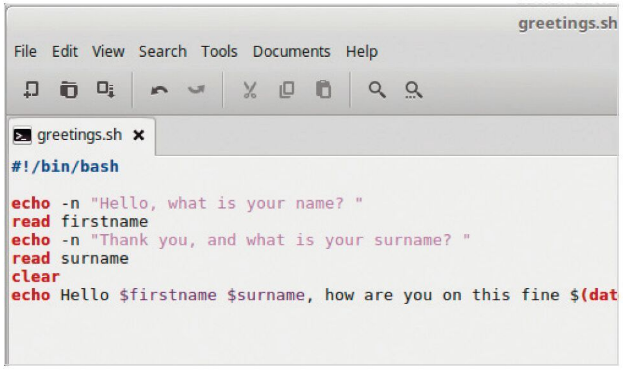
We've added a `-n` to the echo command here which will leave the cursor on the same line as the question, instead of a new line. The `read` command stores the users' input as the variables firstname and surname, to then read back later in the last `echo` line. And the `clear` command clears the screen.



```
Hello David Hayward, how are you today?
david@david-mint ~/scripts $
```

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.



```
#!/bin/bash

echo -n "Hello, what is your name? "
read firstname
echo -n "Thank you, and what is your surname? "
read surname
clear
echo Hello $firstname $surname, how are you on this fine $(dat
```

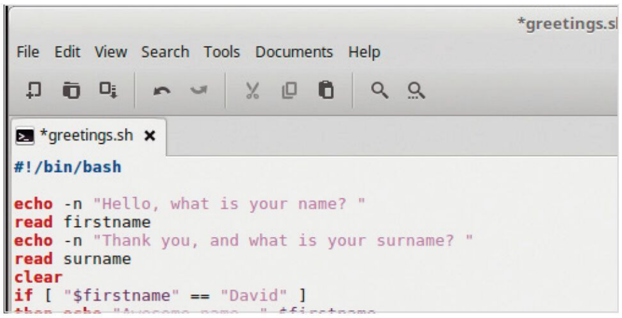
Creating Bash Scripts – Part 3

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

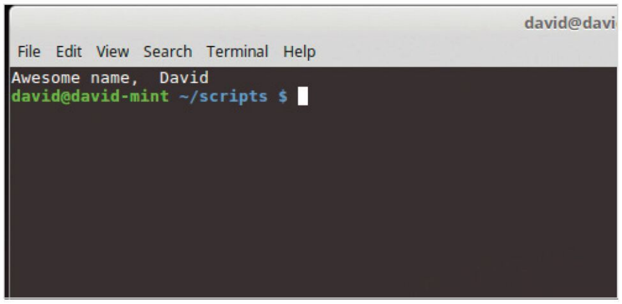
With most programming structures there will come a time where you need to loop through the commands you've entered to create better functionality, and ultimately a better program.

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.



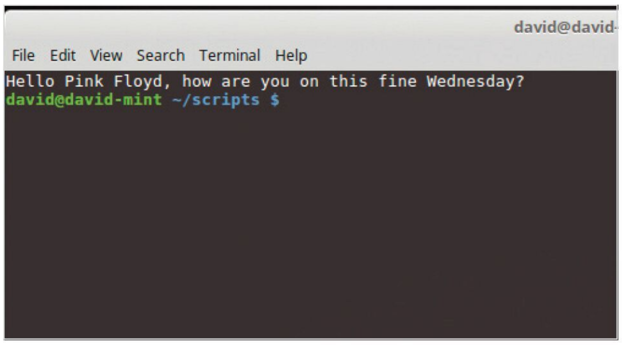
```
#!/bin/bash

echo -n "Hello, what is your name? "
read firstname
echo -n "Thank you, and what is your surname? "
read surname
clear
if [ "$firstname" == "David" ]
```

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).



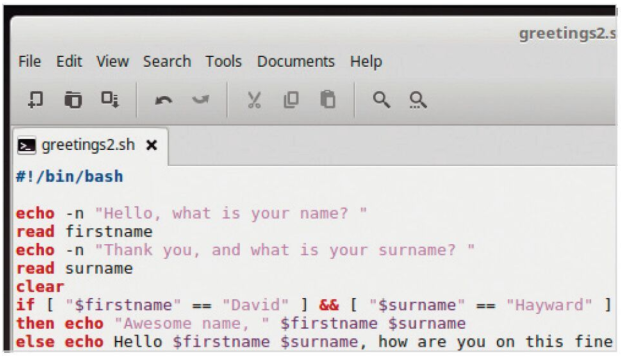
```
Awesome name,  David
david@david-mint ~/scripts $ 
```

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.



```
Hello Pink Floyd, how are you on this fine Wednesday?
david@david-mint ~/scripts $
```

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.

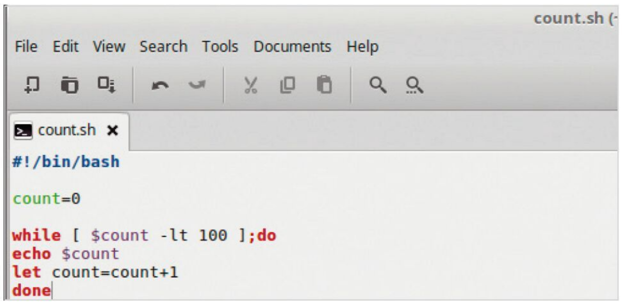


```
#!/bin/bash

echo -n "Hello, what is your name? "
read firstname
echo -n "Thank you, and what is your surname? "
read surname
clear
if [ "$firstname" == "David" ] && [ "$surname" == "Hayward" ]
then echo "Awesome name, " $firstname $surname
else echo Hello $firstname $surname, how are you on this fine
```

MORE LOOPING

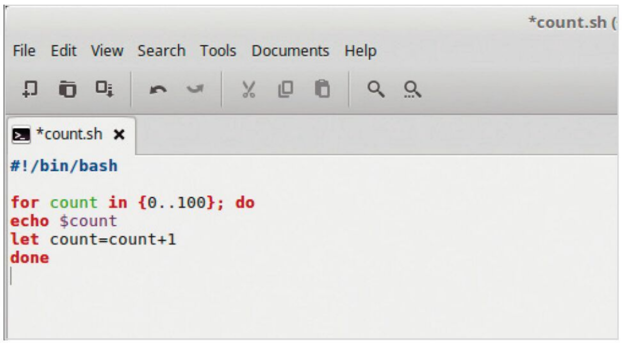
You can loop over data using the FOR, WHILE and UNTIL statements. These can be handy if you're batch naming, copying or running a script where a counter is needed.

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.



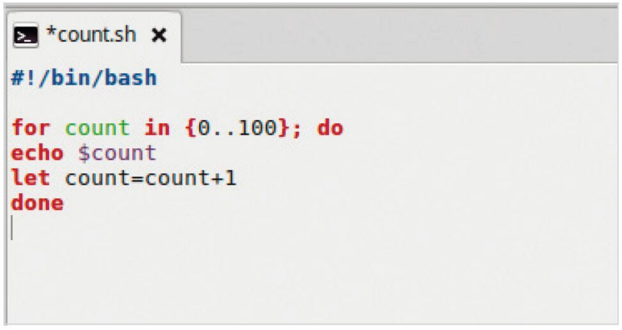
```
#!/bin/bash

count=0

while [ $count -lt 100 ];do
echo $count
let count=count+1
done
```

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.



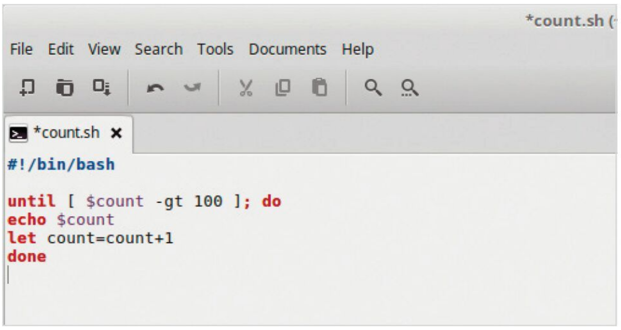
```
#!/bin/bash

for count in {0..100}; do
echo $count
let count=count+1
done
```

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.



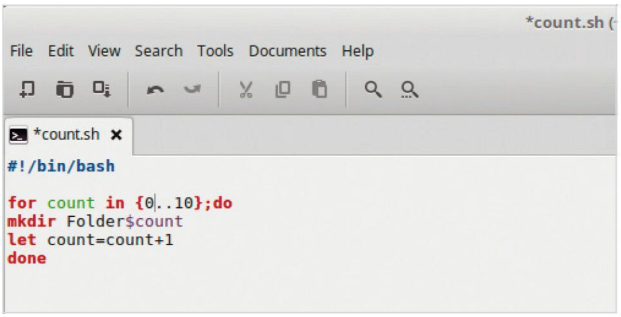
```
#!/bin/bash

for count in {0..100}; do
echo $count
let count=count+1
done
```

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.



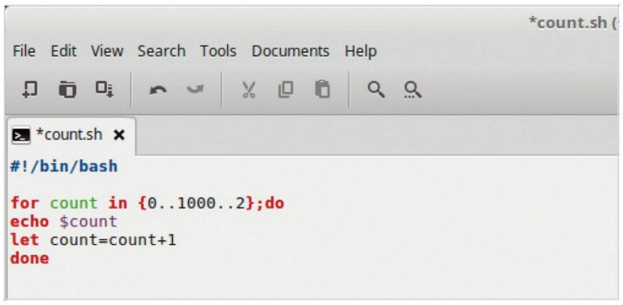
```
#!/bin/bash

until [ $count -gt 100 ]; do
echo $count
let count=count+1
done
```

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.



```
#!/bin/bash

for count in {0..10};do
mkdir Folder$count
let count=count+1
done
```

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`.



```
#!/bin/bash

for count in {0..1000..2};do
echo $count
let count=count+1
done
```

Creating Bash Scripts – Part 4

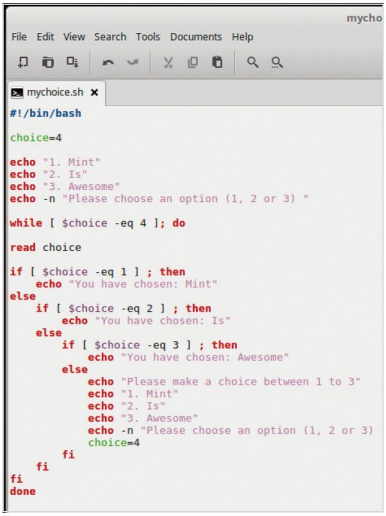
You've encountered user interaction with your scripts, asking what the user's name is and so on. You've also looked at creating loops within the script to either count or simply do something several times. Let's combine and expand some more.

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

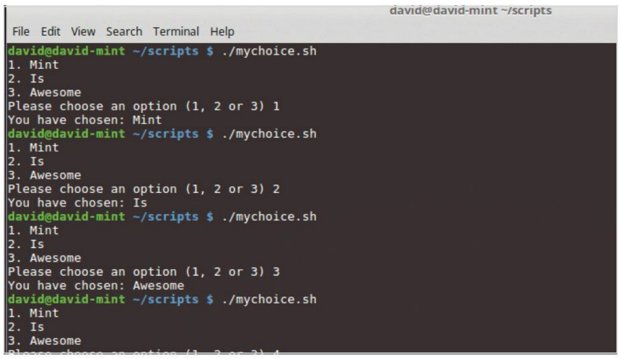
The mychoice.sh script is beginning to look a lot more complex. What we have here is a list of four choices, with three possible options. The options: Mint, Is, and Awesome will be displayed if the user presses the correct option key. If not, then the menu will reappear, the fourth choice.



```
#!/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

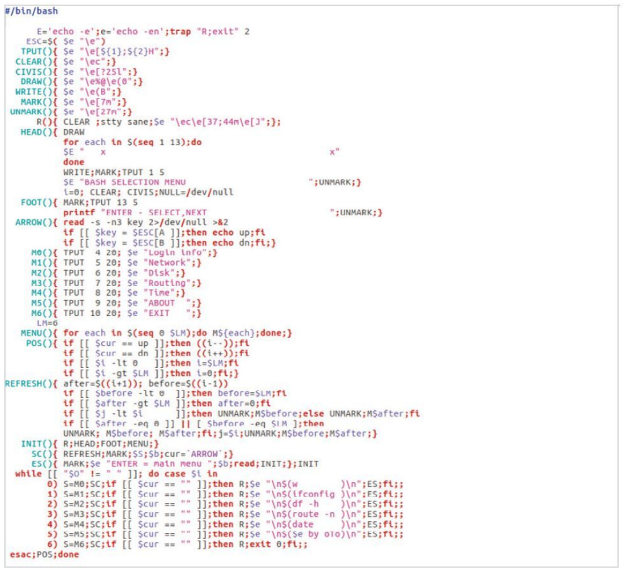
If you follow the script through you soon get the hang of what's going on, based on what we've already covered. WHILE, IF, and ELSE, with the FI closing loop statement will run through the options and bring you back to the start if you pick the wrong option.


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

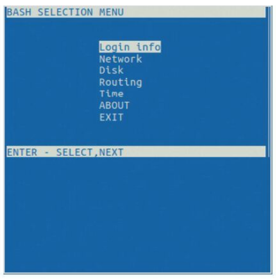
You can, of course, increase the number of choices but you need to make sure that you match the number of choices to the number of IF statements. The script can quickly become a very busy screen to look at. This lengthy script is another way of displaying a menu, this time with a fancy colour scheme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

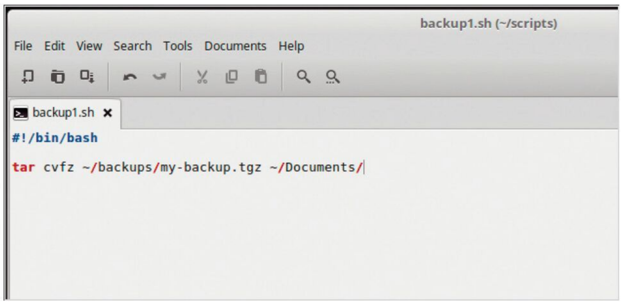


```
BASH SELECTION MENU

                    Login info
                    Network
                    Disk
                    Routing
                    Time
                    ABOUT
                    EXIT

ENTER - SELECT,NEXT
```

CREATING A BACKUP TASK SCRIPT

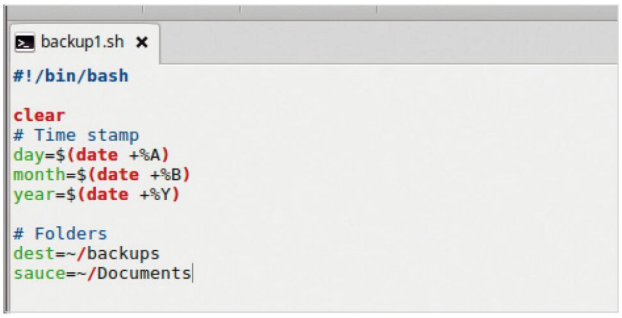
One of the most well used examples of Bash scripting is the creation of a backup routine, one that automates the task as well as adding some customisations along the way.

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.



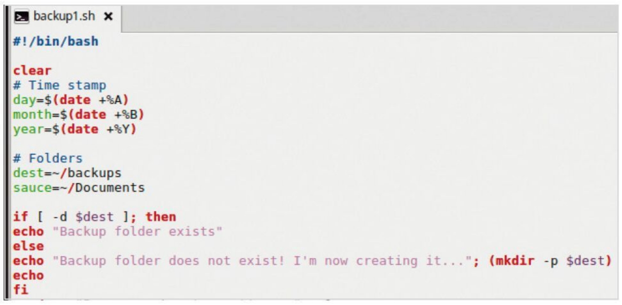
```
#!/bin/bash

tar cvfz ~/backups/my-backup.tgz ~/Documents/
```

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.



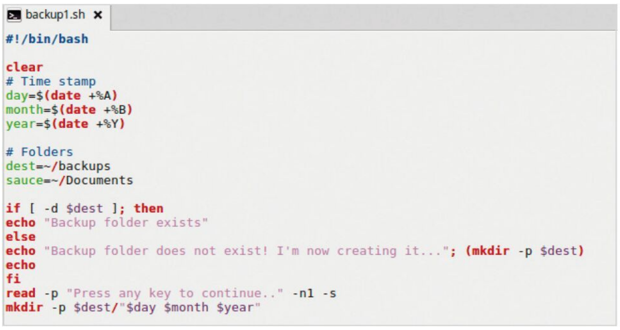
```
#!/bin/bash

clear
# Time stamp
day=$(date +%A)
month=$(date +%B)
year=$(date +%Y)

# Folders
dest=~/backups
sauce=~/Documents
```

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.



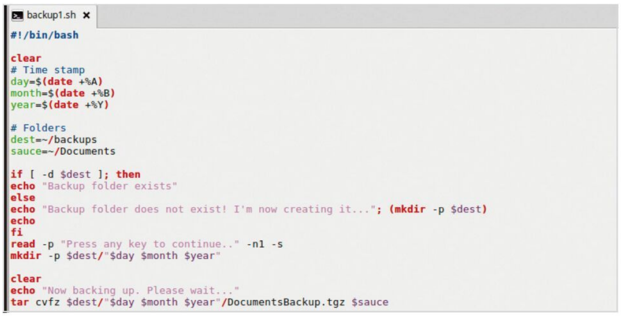
```
#!/bin/bash

clear
# Time stamp
day=$(date +%A)
month=$(date +%B)
year=$(date +%Y)

# Folders
dest=~/backups
sauce=~/Documents

if [ -d $dest ]; then
echo "Backup folder exists"
else
echo "Backup folder does not exist! I'm now creating it..."; (mkdir -p $dest)
echo
fi
```

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.



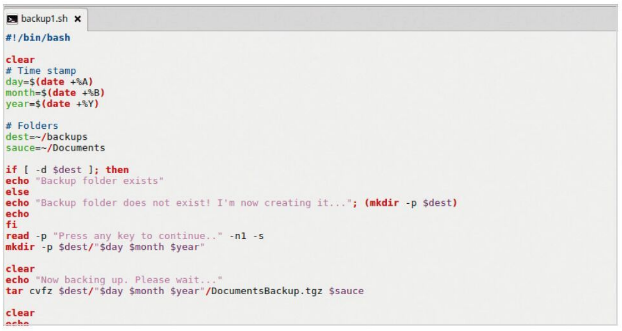
```
#!/bin/bash

clear
# Time stamp
day=$(date +%A)
month=$(date +%B)
year=$(date +%Y)

# Folders
dest=~/backups
sauce=~/Documents

if [ -d $dest ]; then
echo "Backup folder exists"
else
echo "Backup folder does not exist! I'm now creating it..."; (mkdir -p $dest)
echo
fi
read -p "Press any key to continue.." -n1 -s
mkdir -p $dest/"$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.



```
#!/bin/bash

clear
# Time stamp
day=$(date +%A)
month=$(date +%B)
year=$(date +%Y)

# Folders
dest=~/backups
sauce=~/Documents

if [ -d $dest ]; then
echo "Backup folder exists"
else
echo "Backup folder does not exist! I'm now creating it..."; (mkdir -p $dest)
echo
fi
read -p "Press any key to continue.." -n1 -s
mkdir -p $dest/"$day $month $year"

clear
echo "Now backing up. Please wait..."
tar cvfz $dest/"$day $month $year"/DocumentsBackup.tgz $sauce
```

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.



```
#!/bin/bash

clear
# Time stamp
day=$(date +%A)
month=$(date +%B)
year=$(date +%Y)

# Folders
dest=~/backups
sauce=~/Documents

if [ -d $dest ]; then
echo "Backup folder exists"
else
echo "Backup folder does not exist! I'm now creating it..."; (mkdir -p $dest)
echo
fi
read -p "Press any key to continue.." -n1 -s
mkdir -p $dest/"$day $month $year"

clear
echo "Now backing up. Please wait..."
tar cvfz $dest/"$day $month $year"/DocumentsBackup.tgz $sauce

clear
echo
```

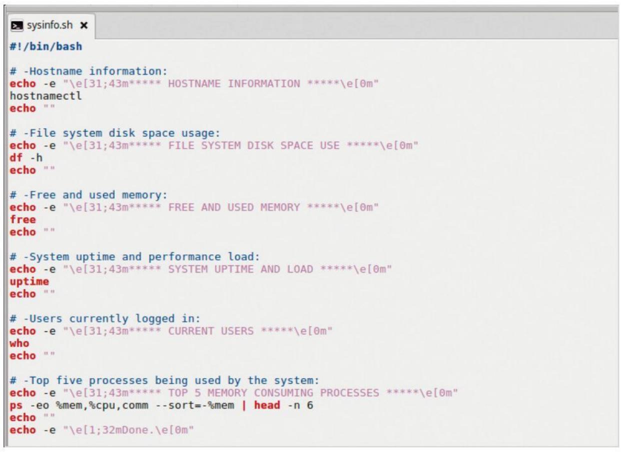
Creating Bash Scripts – Part 5

The backup script we looked at previously can be further amended to incorporate choices; or in other words, 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

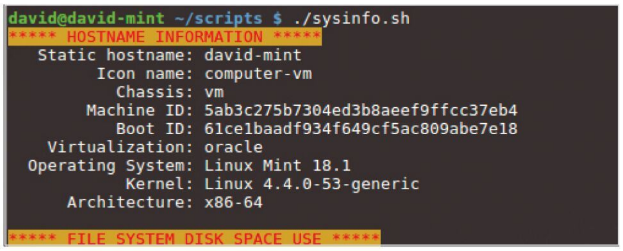
Are you entering line after line of commands to retrieve system information, find a file or rename a batch of files? A script is a better answer.

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.



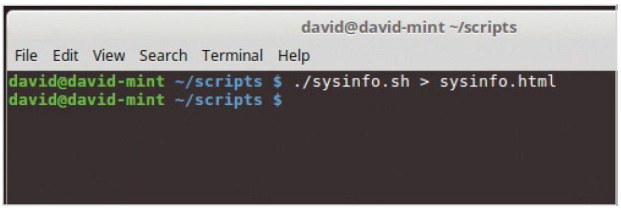
```
#!/bin/bash

# -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 *****
```

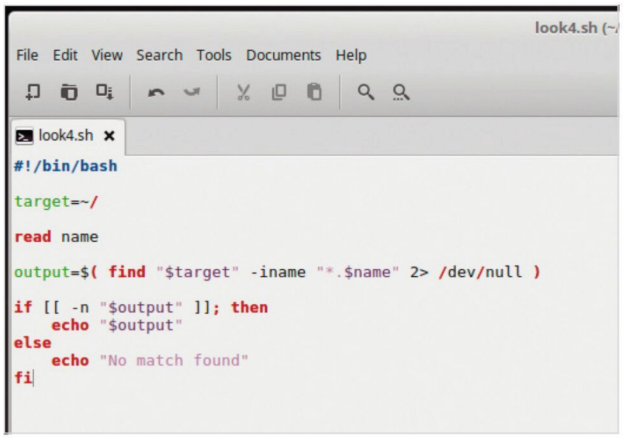
STEP 3 Each of the sections runs a different Terminal command, outputting the results under the appropriate heading. You can include a lot more, such as the current aliases being used in the system, the current time and date and so on. Plus, you could also pipe all that information into a handy HTML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



david@david-mint ~/scripts
File Edit View Search Terminal Help

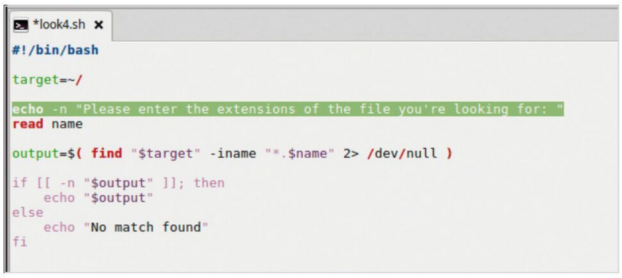
```
david@david-mint ~/scripts $ ./sysinfo.sh > sysinfo.html
david@david-mint ~/scripts $
```

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.



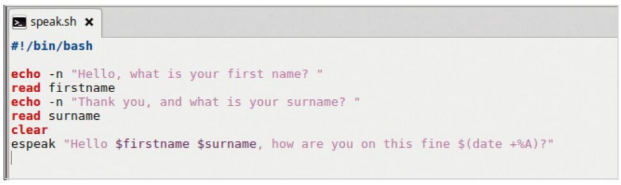
```
#!/bin/bash

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.



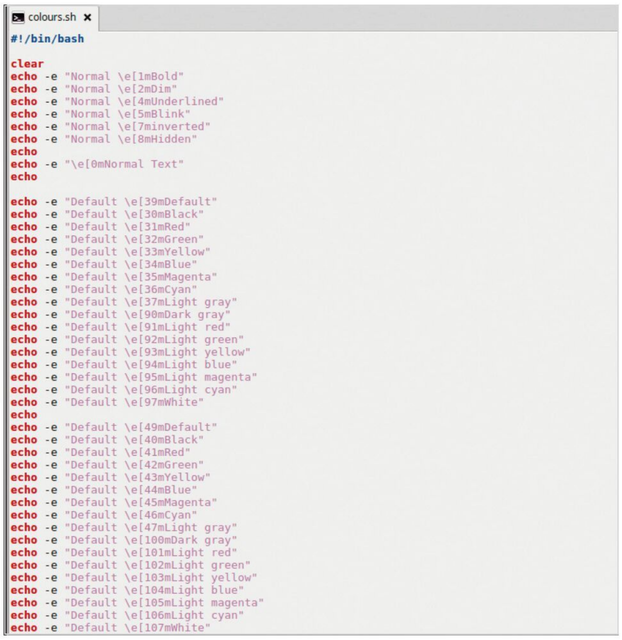
```
#!/bin/bash

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.



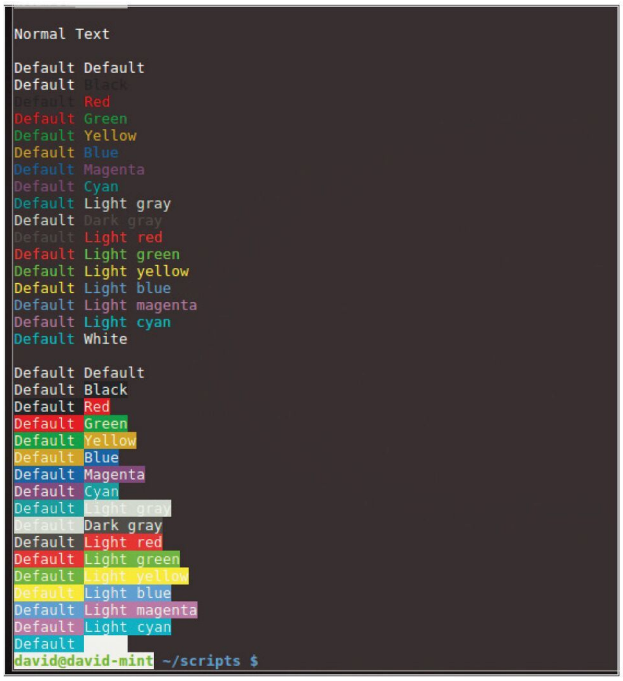
```
#!/bin/bash

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.



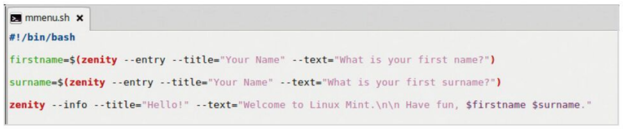
```
#!/bin/bash

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.



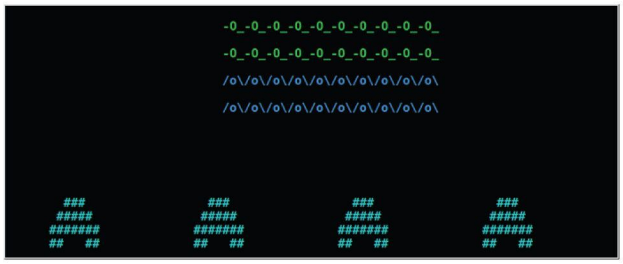
```
Normal Text

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.



```
#!/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. Movement-based games are diffiuclt, and sometimes buggy, however a good text adventure, or Fighting Fantasy type game is a perfect choice for gaming in the Terminal. give it a go, and let us know how you get on.



```
                                 -0_-0_-0_-0_-0_-0_-0_-0_-0_-0_

                                 -0_-0_-0_-0_-0_-0_-0_-0_-0_-0_

                                 /o\/o\/o\/o\/o\/o\/o\/o\/o\/o\

                                 /o\/o\/o\/o\/o\/o\/o\/o\/o\/o\




     ###                  ###                  ###                  ###
    #####                #####                #####                #####
   #######              #######              #######              #######
   ##   ##              ##   ##              ##   ##              ##   ##
```

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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).

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.

uname-a

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

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.

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`.

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.

adduser

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

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.

delgroup

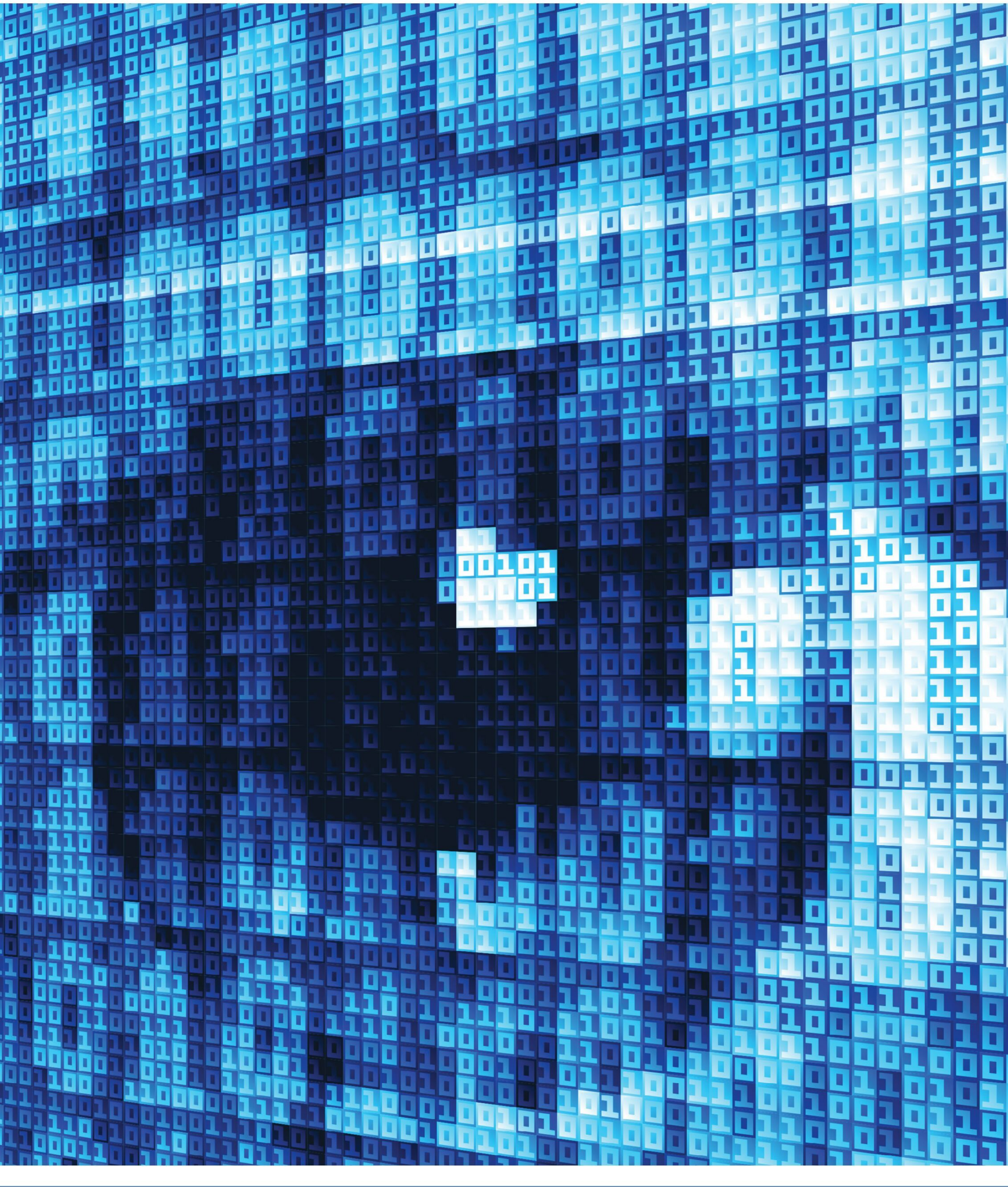
The `delgroup` command removes a group from the system. You cannot remove a group that is the primary group of any users.

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.

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.



Code Projects and Ideas

To help you on your way to becoming a better coder, we've included some great coding projects and ideas that you can apply to your own code, use as a demo for those who are starting to code, or improve and send out into the coding community. Want to learn how to craft some retro code? How about learning how to code a text adventure? These and more are all found in this final section.

No matter where you go next in coding, whether it's on to learning more languages, creating amazing apps and games, or helping out and teaching new coders to avoid the pitfalls you made, remember the basics, and have fun!

From here, it's down to you and your imagination. So dream it, and code it.

Planning Your Code

Knowing how to code is great, but it's not going to get you very far if you don't know what it is you're supposed to be coding. Small samples of code don't need much planning, but bigger projects do, so here are some tips on planning your code.

You will find that nearly all the professional developers out there will set aside some time to plan out how their code is going to fit into the wider project on which they're working. Regardless of whether they're working as part of a team on the latest game, developing a workflow user interface for the company, or creating a set of automated tasks that can be used across a number of different platforms, each developer will set out their code in a plan.

The planning stages can be unique to the developer, unique to the company for which they're working, or the project on which they're working. Planning meetings will often sketch out a rough plan of how the end product will function, with each developer team then working on their element within the project. Lone developers will sometimes go back to basics, grab paper and a pencil and start getting their ideas down. The end result is always the same, though, a plan of action that enables the developer to plan without worrying about the actual code at this early stage.

PSEUDOCODE

```
Line 1  RECEIVE userName FROM (STRING) KEYBOARD
Line 2  RECEIVE pinNumber FROM (STRING) KEYBOARD
Line 3  IF userName VALID OR pinNumber VALID THEN
Line 4          Allow access to network
Line 5  ELSE
Line 6          SEND "Access Denied" TO SCREEN
Line 7  END IF
```

One of the more popular methods of planning to code is using pseudocode. It's terms used in programming circles that allows the developer to represent the implementation of code, the sequence of the actions, using plain English and a mixture of code examples together with a flowchart or two to help 'picture' loops within the coding project.

For example, a developer who is planning to write a program that will check user input for odd or even numbers will probably begin with a basic statement of pseudocode:

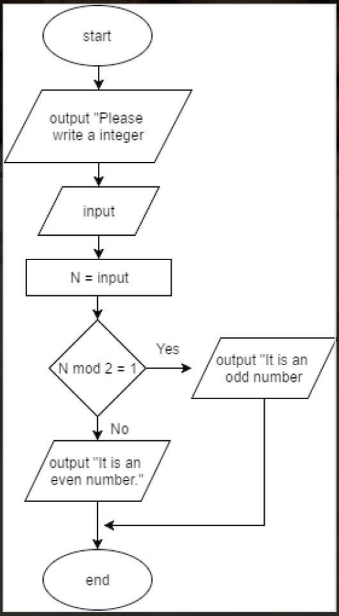
A program to allow a user to check if the number is odd or even. Then the basic statement can be expanded:

Code Example:

```
If "1"
        Print response
        "I am result 1"

If "2"
        Print response
        "I am result 2"
```

It's very simplistic, but from there the developer can continue to plan the remainder of the code needed without taking up too much valuable time – but while also having a readable plan of action for how the code is going to be formed.

The actual writing of the code, therefore, takes place later on in the project. Once there's a plan of action in place, the

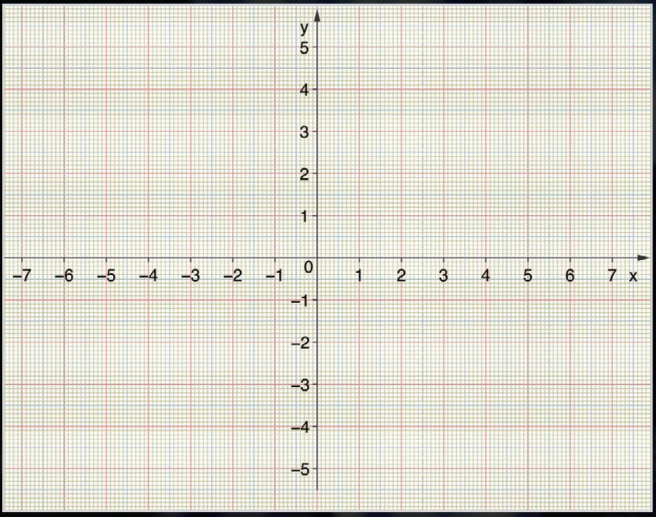

start
output "Please write a integer
input
N = input
N mod 2 = 1
Yes
output "It is an odd number
No
output "It is an even number."
end

developer can share the plan with their colleagues, lead developers, managers, and so on. Then the plan can be further tweaked to include other aspects that the original developer may have missed or may have been removed due to being being classed as unnecessary.

This process will not only trim the fat off the code before it's even written but will also help plug any gaps in the project before the hard work of entering the code begins.

GRAPH PAPER

If you're planning to include graphics in your project, such as a platform game, then one of the best resources you can use – and one that has been used since the early days of coding games – is graph paper.



Super Mario Bros., launched for the Nintendo Entertainment System in 1985, was nearly exclusively drawn out – both characters and levels – on graph paper before being applied to code. Entire game maps were often created this way, especially on the older 8-bit machines such as the Commodore 64 and Sinclair ZX Spectrum home computers. 8-Bit graphics were ideal for graph paper since the sprites used then were made up of eight or sixteen blocks; the developer could then visualise the layout of the sprite using the drawing on the graph paper and apply that to the code. The end result, of course, is the character appearing on the screen.

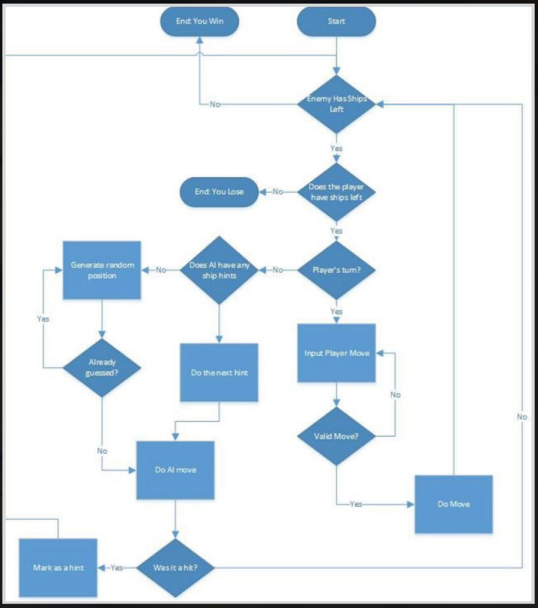
Once the graphical planning stage is complete, it's then down to you as the developer to fill in the blanks, as it were. This means creating the code that will place and animate your characters within the game, as well as developing interactions, collision detection, number of lives, and so on. With the help of some pseudocode, you could easily plan out what modules you'd use or create, the number of variables needed, and the various endgame states that will arise when someone plays your game.

GET PLANNING

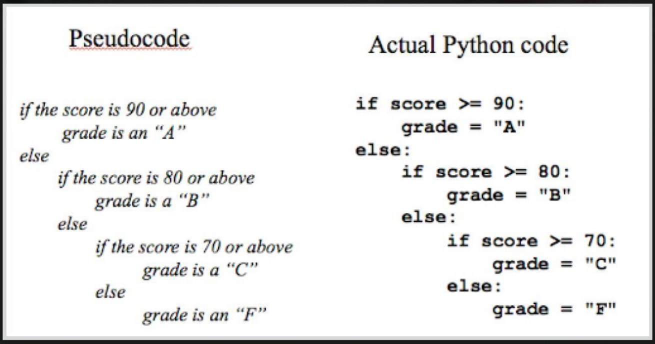
The whole point of planning is to create something easy to follow that will help you write the code for your project. Planning isn't supposed to be a chore; it's there to aid you when you get stuck trying to work out which variables you've used, where you are in the game, with what you're left and how the game will eventually pan out and end.

When planning, begin with small steps. Coding a Hangman game, for example, begins with the plan to get the player's name, then includes the word bank, drawing out the stages of the Hangman, asking the right questions, and finishing the game in one of two states – a win or a lose – then asking if the player wants another go.

Once you're used to planning it'll come naturally, and you'll find yourself thankful for all those post-it notes stuck to a sheet of A3 paper, as well as the reams of graph paper that detail your characters and level designs.



Pseudocode flowcharts can be simple, or as complex as necessary



Pseudocode

```
if the score is 90 or above
    grade is an "A"
else
    if the score is 80 or above
        grade is a "B"
    else
        if the score is 70 or above
            grade is a "C"
        else
            grade is an "F"
```

Actual Python code

```
if score >= 90:
    grade = "A"
else:
    if score >= 80:
        grade = "B"
    else:
        if score >= 70:
            grade = "C"
        else:
            grade = "F"
```

Writing pseudocode will help you visualise the actual code

Finding Resources

The Python community is huge and spans countless websites and forums on the Internet, so finding extra help here and there isn't too much of a problem. However, finding good Python resources can be, so here are some ideas for you to try.

You will reach a point in your Python coding experience when you need to rely on some external resources to help you through a sticky patch in your project or give you a bit of inspiration. Python resources are available throughout the Internet, but not all of them are any good. If you spend any time looking up code for a particular project, then you're likely to come across copious examples, yet not all of them seem to work.

That's one of the main issues with resources for Python. With the language being twenty-nine years old, and many different versions under the bridge, there's a significant amount of content out there that's simply out of date – or just wrong for the version you're using. Newer versions, even by one number out, can have different outputs for your code from a previous version – it's not always that case, but it can happen. Therefore, you need to find yourself some trusted Python resources that work, or at least 90% would work, with the version you're using and the code within.

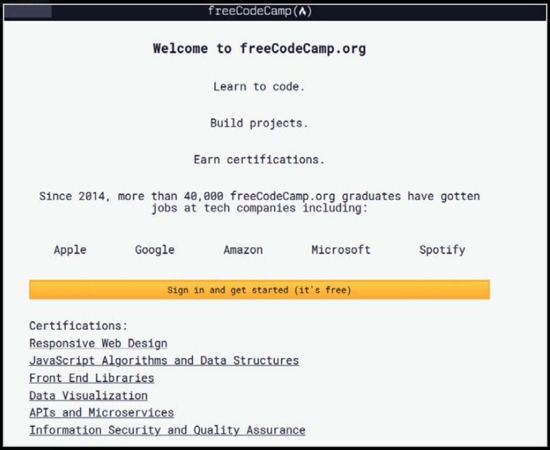
THE HITCHHIKER'S GUIDE TO PYTHON

One of the first places that's worth turning to is: freeCodeCamp.org. This website features a wealth of code that covers most aspects of the learning experience, even for those who are experts at Python. You'll find content that will take your Python journey to new heights; with code snippets to help you access networking elements, military-style encryption and more, and you can even gain certification through the site.

Programiz is another useful resource that features Python code examples covering a wide range of concepts. There's everything in here from the original "Hello, world!" code, through to code that will transpose a matrix or find the hash of a file. There's also an online compiler and in-depth looks into further reading for aspiring data scientists.

If you're looking for resources about using graphics in Python, then ProgramArcadeGames.com is a good place to start. This site has plenty of code for the Pygame module, including moving an object around the screen using a mouse or game controller, creating fractals, moving sprites, and an example of a classic Pong game for you to try out.

GitHub is always worth a look. However, the code featured here isn't always up to date, and you may find yourself having to convert Python 2 code into Python 3. Nevertheless, you do come across some gems from time to time, especially if you look up the Pygame examples that users have uploaded. We recommend the excellent Solarwolf, by Pete Shinners.


freeCodeCamp(🔥)

Welcome to freeCodeCamp.org

Learn to code.

Build projects.

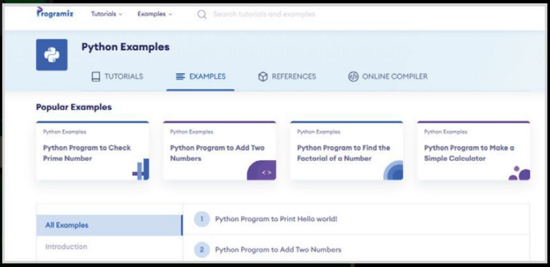
Earn certifications.

Since 2014, more than 40,000 freeCodeCamp.org graduates have gotten jobs at tech companies including:

Apple Google Amazon Microsoft Spotify

Sign in and get started (it's free)

Certifications:
Responsive Web Design
JavaScript Algorithms and Data Structures
Front End Libraries
Data Visualization
APIs and Microservices
Information Security and Quality Assurance



Python Examples

Popular Examples

Python Examples
Python Program to Check Prime Number

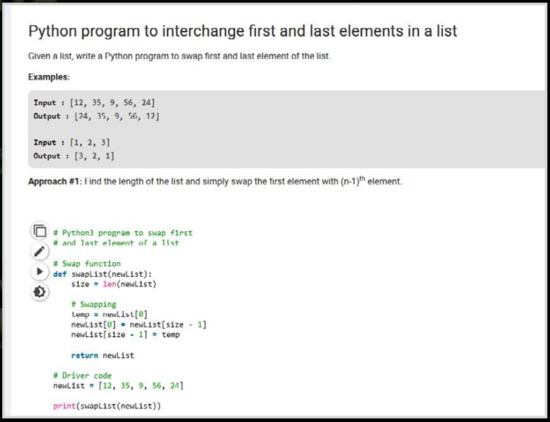
Python Examples
Python Program to Add Two Numbers

Python Examples
Python Program to Find the Factorial of a Number

Python Examples
Python Program to Make a Simple Calculator

1 Python Program to Print Hello world!

2 Python Program to Add Two Numbers


Python program to interchange first and last elements in a list

Given a list, write a Python program to swap first and last element of the list.

Examples:

```
Input : [12, 35, 9, 56, 24]
Output : [24, 35, 9, 56, 12]

Input : [1, 2, 3]
Output : [3, 2, 1]
```

Approach #1: Find the length of the list and simply swap the first element with (n-1)th element.

```
# Python3 program to swap first
# and last element of a list

# Swap function
def swapList(newList):
    size = len(newList)

    # Swapping
    temp = newList[0]
    newList[0] = newList[size - 1]
    newList[size - 1] = temp

    return newList

# Driver code
newList = [12, 35, 9, 56, 24]

print(swapList(newList))
```

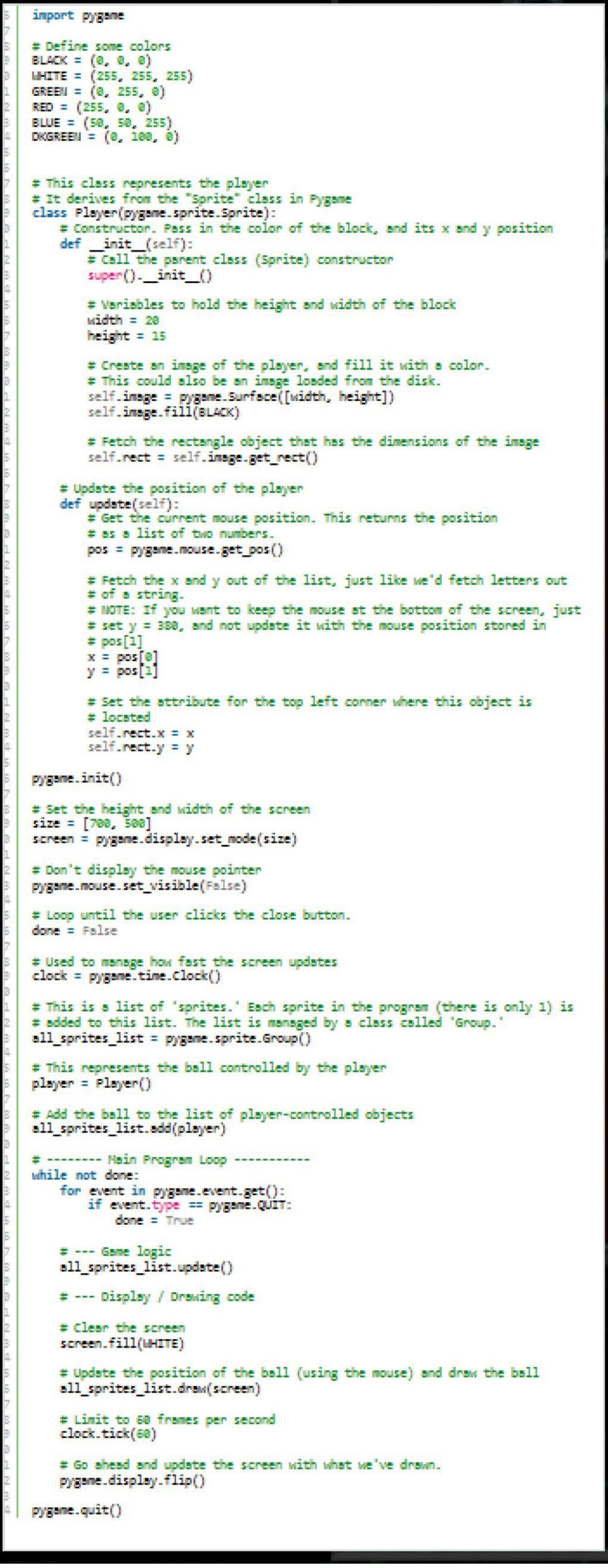
Guardian Info
The Guardians protect the Power Cubes on every level. They will become more aggressive in the later levels.
Don't get too close, they like to shoot point blank.

STRONG GOOGLE-FU

There's no trick to finding good Python resources online; most of the time, it's a case of trial and error. You'll need to make sure you're searching for the right content, though, just to save you some time. If it's a Pygame specific set of resources you're looking for then you'll need to specify the Python version you're using; include Pygame, and include what it is you're looking for – it may sound like we're teaching you to suck eggs, but effective searching is often overlooked.

Again, it's also worth looking up working examples of a type of game if you're looking for Pygame content. If you want to learn how to engineer sprites moving from one side of the screen to the other, then look up some examples of Space Invaders, which feature such movements. The same goes if you're making a platform game, examine some classic games that involved platforming, like Mario and so on.

One last resource is our very own code portal. Log in to **https://bdmpublications.com/code-portal/**, and you'll find a continually growing list of Python code and resources that you can freely add to your own code, then chop, change, and use for whatever projects you have in mind.



```
import pygame

# Define some colors
BLACK = (0, 0, 0)
WHITE = (255, 255, 255)
GREEN = (0, 255, 0)
RED = (255, 0, 0)
BLUE = (50, 50, 255)
DKGREEN = (0, 100, 0)


# This class represents the player
# It derives from the "Sprite" class in Pygame
class Player(pygame.sprite.Sprite):
    # Constructor. Pass in the color of the block, and its x and y position
    def __init__(self):
        # Call the parent class (Sprite) constructor
        super().__init__()

        # Variables to hold the height and width of the block
        width = 20
        height = 15

        # Create an image of the player, and fill it with a color.
        # This could also be an image loaded from the disk.
        self.image = pygame.Surface([width, height])
        self.image.fill(BLACK)

        # Fetch the rectangle object that has the dimensions of the image
        self.rect = self.image.get_rect()

    # Update the position of the player
    def update(self):
        # Get the current mouse position. This returns the position
        # as a list of two numbers.
        pos = pygame.mouse.get_pos()

        # Fetch the x and y out of the list, just like we'd fetch letters out
        # of a string.
        # NOTE: If you want to keep the mouse at the bottom of the screen, just
        # set y = 380, and not update it with the mouse position stored in
        # pos[1]
        x = pos[0]
        y = pos[1]

        # Set the attribute for the top left corner where this object is
        # located
        self.rect.x = x
        self.rect.y = y

pygame.init()

# Set the height and width of the screen
size = [700, 500]
screen = pygame.display.set_mode(size)

# Don't display the mouse pointer
pygame.mouse.set_visible(False)

# Loop until the user clicks the close button.
done = False

# Used to manage how fast the screen updates
clock = pygame.time.Clock()

# This is a list of 'sprites.' Each sprite in the program (there is only 1) is
# added to this list. The list is managed by a class called 'Group.'
all_sprites_list = pygame.sprite.Group()

# This represents the ball controlled by the player
player = Player()

# Add the ball to the list of player-controlled objects
all_sprites_list.add(player)

# -------- Main Program Loop -----------
while not done:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            done = True

    # --- Game logic
    all_sprites_list.update()

    # --- Display / Drawing code

    # Clear the screen
    screen.fill(WHITE)

    # Update the position of the ball (using the mouse) and draw the ball
    all_sprites_list.draw(screen)

    # Limit to 60 frames per second
    clock.tick(60)

    # Go ahead and update the screen with what we've drawn.
    pygame.display.flip()

pygame.quit()
```

Creating a Loading Screen

If you're looking to add a little something extra to your Python code, then consider including a loading screen. The loading screen is a short introduction, or piece of art, that appears before the main part of your code.

LOAD""

Back in the 80s, in the 8-bit home computing era, loading screens were often used to display the cover of a game as it loaded from tape. The image would load itself in, usually one-line-at-a-time, then proceed to colour itself in while the loading raster bars danced around in the borders of the screen.

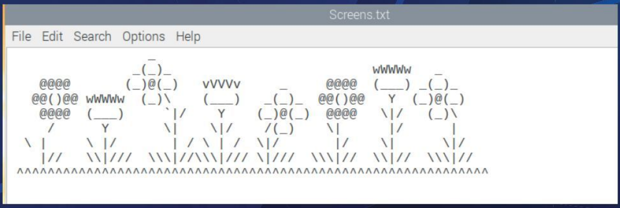
Loading screens were a part of the package and often the buy-in for the whole game as an experience. Some loading screens featured animations, or a countdown for time remaining as the game loads, while others even went so far as to include some kind of playable game. The point being: a loading screen is not just an artistic part of computing history, but an introduction to the program that's about to be run.

While these days loading screens may no longer be with us, in terms of modern gaming we can still include them in our own Python content. Either just for fun, or to add a little retro-themed spice to the mix.

SCREEN$

Creating a loading screen in Python is remarkably easy. You have several options to hand: you can create a tkinter window and display an image, followed by a brief pause, before starting your main code, or you could opt for a console-based ASCII art version that loads one-line-at-a-time. Let's look at the latter and see how it works.

First you'll need some ASCII art, you can look up plenty of examples online, or use an image to ASCII Art converter to transform any images you have to ASCII. When you have your ASCII art, drop it into a newly created folder inside a normal text file.

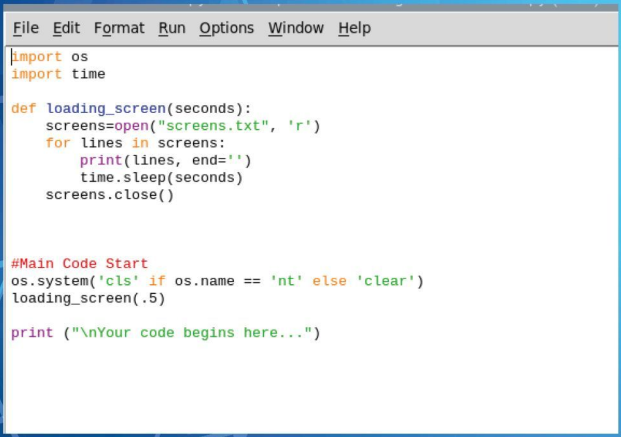


```
                  _
                _(_)_                           wWWWw    _
   @@@@        (_)@(_)   vVVVv      _     @@@@  (___) _(_)_
  @@()@@ wWWWw    (_)\    (___)   _(_)_  @@()@@   Y  (_)@(_)
   @@@@  (___)      `|/    Y    (_)@(_)  @@@@   \|/   (_)\
    /      Y        \|    \|/    /(_)    \|      |/      |
 \ |     \ |/        | / \ | /  \|/        |/    \|      \|/
   |//   \\|///   \\\|//\\\|/// \|///   \\\|//  \\|//  \\\|//
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

THE CODE

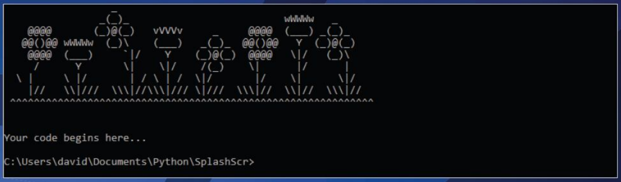
Launch Python and enter the following code to a New File:

```
import os
import time

def loading_screen(seconds):
    screens=open("screens.txt", 'r')
    for lines in screens:
        print(lines, end='')
        time.sleep(seconds)
    screens.close()

#Main Code Start
os.system('cls' if os.name == 'nt' else
'clear')
loading_screen(.5)

print ("\nYour code begins here...")
```



```
import os
import time

def loading_screen(seconds):
    screens=open("screens.txt", 'r')
    for lines in screens:
        print(lines, end='')
        time.sleep(seconds)
    screens.close()



#Main Code Start
os.system('cls' if os.name == 'nt' else 'clear')
loading_screen(.5)

print ("\nYour code begins here...")
```

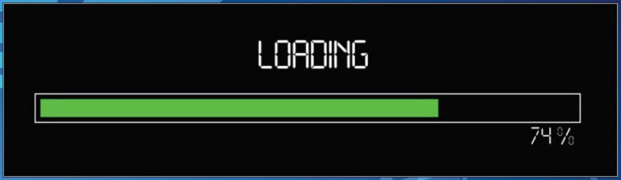
The code is quite simple: import the OS and Time modules and then create a Python function called loading_screen with a (seconds) option. Within the function, open the text file with the ASCII art as read-only and create a `For` loop that'll read the text file one-line-at-a-time. Next, print the lines – incidentally, the `lines, end=''` element will strip the newline from the text document, without it you'll end up with a double-line spaced display. Include the timing in seconds and close the text file buffer. The final part of the code, `#Main Code Start`, is where you'll clear the screen (CLS for Windows, Clear for Linux (Raspberry Pi) and macOS) and call the function, together with the output number of seconds to wait for each line to be written to the screen – in this case, .5 of a second.



```
Your code begins here...

C:\Users\david\Documents\Python\SplashScr>
```

Save the code as screens.py, drop into a Command Prompt or Terminal and execute it. The screen will clear and your ASCII art inside the text file will load in line-by-line, creating a loading screen effect.

LOADING…



Another favourite introduction screen is that of a simple loading animation, where the word loading is displayed, followed by some characters, and a percentage of the program loaded. While it may not take long for your Python code to load, the effect can be interesting.

Create a New File in Python and enter the following code:

```
import os
import time

def loading_bar(seconds):
  for loading in range(0,seconds+1):
    percent = (loading * 100) // seconds
    print("\n")
    print("Loading...")
    print("<" + ("-" * loading) + (" " * (seconds-
loading)) + "> " + str(percent) + "%")
    print("\n")
    time.sleep(1)
    os.system('cls' if os.name == 'nt' else
'clear')

#Main Code Start
loading_bar(10)
print ("\nYour code begins here...")
```

The code works in much the same way as the previous, except, instead of reading from a text file, it's running through a For loop that prints Loading… followed by an animation of sorts, along with a percentage counter; clearing the screen every second and displaying the new results. It's simple, yes, but quite effective in its design.

COMBINING THE TWO

How about combining the two elements we've looked at? Let's begin with a Loading… progress bar, followed by the loading screen. After that, you can include your own code and continue your program. Here's the code:

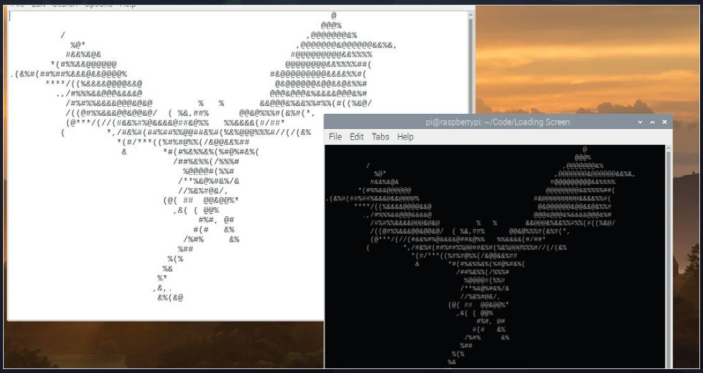
```
import os
import time

def loading_bar(seconds):
  for loading in range(0,seconds+1):
    percent = (loading * 100) // seconds
    print("\n")
    print("Loading...")
    print("<" + ("-" * loading) + (" " * (seconds-
loading)) + "> " + str(percent) + "%")
    print("\n")
    time.sleep(1)
    os.system('cls' if os.name == 'nt' else
'clear')

def loading_screen(seconds):
    screens=open("screens.txt", 'r')
    for lines in screens:
        print(lines, end='')
        time.sleep(seconds)
    screens.close()

#Main Code Start
loading_bar(10)
os.system('cls' if os.name == 'nt' else 'clear')
loading_screen(.5)

print ("\nYour code begins here...")
```

You can, of course, pull those functions up wherever and whenever you want in your code and they'll display, as they should, at the beginning. Remember to have the ASCII text file in the same folder as the Python code, or, at the screens=open("screens.txt", 'r') part of the code, point to where the text file is located.

ADVENTURE TIME

A good example of using loading screen, ASCII art text images is when coding a text adventure. Once you've established your story, created the characters, events and so on, you could easily incorporate some excellently designed ASCII art to your game.

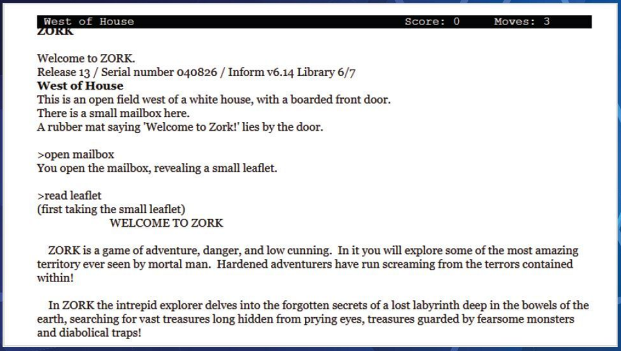
Imagine coming across a dragon, in game, and displaying its representation as ASCII. You can then load up the image lines, one-by-one, and continue with the rest of the adventure code. It's certainly worth having a play around with and it'll definitely add a little something else extra.



Planning a Text Adventure

Planning a text adventure is one of the best ways of teaching yourself some new Python tricks and essential code. A good text adventure contains many aspects, all of which will need to be carefully planned and structured, to get the best from the code.

If you can think back to some of the classic text adventures you may have played in the past, consider the amount of work that was needed to get them up and running. If you're not familiar with a text adventure, then open a browser and go to **http://textadventures.co.uk/games/play/5zyoqrsugeopel3ffhz_vq**. This is an online version of the classic text adventure Zork. Zork was created for the DEC PDP-10 workstations, using the MDL programming language back in the late seventies. Despite its age, it's still considered one of the best text adventures to play.

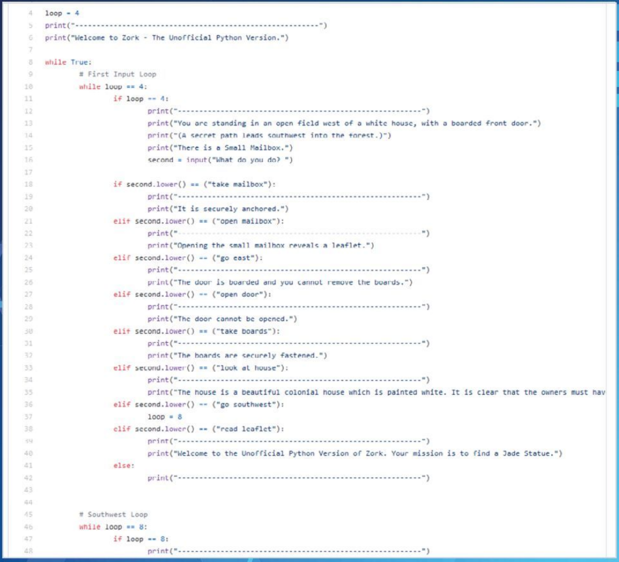


```
 West of House                                    Score: 0      Moves: 3
ZORK

Welcome to ZORK.
Release 13 / Serial number 040826 / Inform v6.14 Library 6/7
West of House
This is an open field west of a white house, with a boarded front door.
There is a small mailbox here.
A rubber mat saying 'Welcome to Zork!' lies by the door.

>open mailbox
You open the mailbox, revealing a small leaflet.

>read leaflet
(first taking the small leaflet)
                    WELCOME TO ZORK

   ZORK is a game of adventure, danger, and low cunning.  In it you will explore some of the most amazing
territory ever seen by mortal man.  Hardened adventurers have run screaming from the terrors contained
within!

   In ZORK the intrepid explorer delves into the forgotten secrets of a lost labyrinth deep in the bowels of the
earth, searching for vast treasures long hidden from prying eyes, treasures guarded by fearsome monsters
and diabolical traps!
```

Zork, or any other text adventure game, features several elements necessary to make the game both interesting and playable.

INGREDIENTS FOR A COMPELLING ADVENTURE

The first ingredient is a story, a plot that will take the player on a ride through their own imagination. It doesn't necessarily have to include dragons, barbarians, spaceships, or other such things, but it needs to be engaging. Perhaps it could follow the theme of a murder mystery, much in the same vein as an Agatha Christie novel, or it could be something that explores the base fears of people.



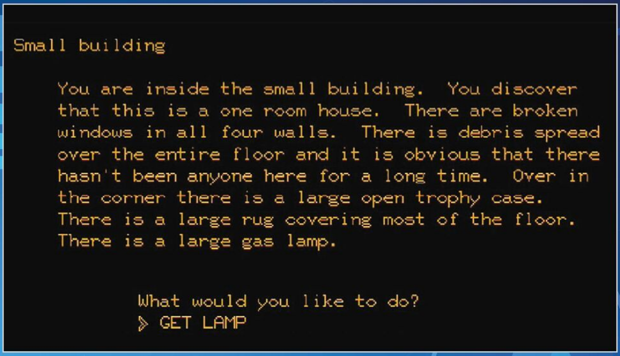
```python
loop = 4
print("------------------------------------------------------------")
print("Welcome to Zork - The Unofficial Python Version.")

while True:
    # First Input Loop
    while loop == 4:
        if loop == 4:
            print("------------------------------------------------------------")
            print("You are standing in an open field west of a white house, with a boarded front door.")
            print("(A secret path leads southwest into the forest.)")
            print("There is a Small Mailbox.")
            second = input("What do you do? ")

        if second.lower() == ("take mailbox"):
            print("------------------------------------------------------------")
            print("It is securely anchored.")
        elif second.lower() == ("open mailbox"):
            print("------------------------------------------------------------")
            print("Opening the small mailbox reveals a leaflet.")
        elif second.lower() == ("go east"):
            print("------------------------------------------------------------")
            print("The door is boarded and you cannot remove the boards.")
        elif second.lower() == ("open door"):
            print("------------------------------------------------------------")
            print("The door cannot be opened.")
        elif second.lower() == ("take boards"):
            print("------------------------------------------------------------")
            print("The boards are securely fastened.")
        elif second.lower() == ("look at house"):
            print("------------------------------------------------------------")
            print("The house is a beautiful colonial house which is painted white. It is clear that the owners must hav
        elif second.lower() == ("go southwest"):
            loop = 8
        elif second.lower() == ("read leaflet"):
            print("------------------------------------------------------------")
            print("Welcome to the Unofficial Python Version of Zork. Your mission is to find a Jade Statue.")
        else:
            print("------------------------------------------------------------")


    # Southwest Loop
    while loop == 8:
        if loop == 8:
            print("------------------------------------------------------------")
```

The second crucial element is choice. Without that, the player may as well read the story from a book. It's the choice presented to the player that makes the game an adventure. If the player chooses something wrong, they could be punished, whereas a correct choice could gain them gold or some other form of reward. However, it doesn't always need to be a positive or negative choice; after all, most common real-life decisions we make aren't life or death, they merely result in one thing happening or another.

Locations are similarly important. Text Adventures rely on good the locations featured in the game can draw a picture in the mind of the player. Whether they're on a wooden ship in the middle of the sea or trying to escape a haunted mansion, each location has to stand out.

Items are also a vital requirement for a good adventure game. A player will need to be able to pick up, examine, take, and use, an item from its location in the game. For example, your character could find a key in the first location, pocket it, and then use it later on in the game to unlock a door. You'll need to factor in the list of possible items, and you'll also need to factor in the real-world physics of each item too. If you have your adventurer enter a room with an anvil, then allowing them to pick up the anvil and put it in their pocket isn't particularly realistic. One of the major drawbacks of some text adventures, and even modern graphical games, is the unrealistic carrying capability of the player.

Although not always necessary, puzzles do form an interesting aspect of a text adventure. These puzzles can test the player's abilities and expand the game beyond simply finding an object and using it. Puzzles can range from using these objects correctly to achieve something to finding the correct route through a maze, with some being vital to the plot while others are simply there to give the player something interesting to do. Ideally, puzzles should be easy near the beginning of the game and get progressively harder as your character advances through the game.



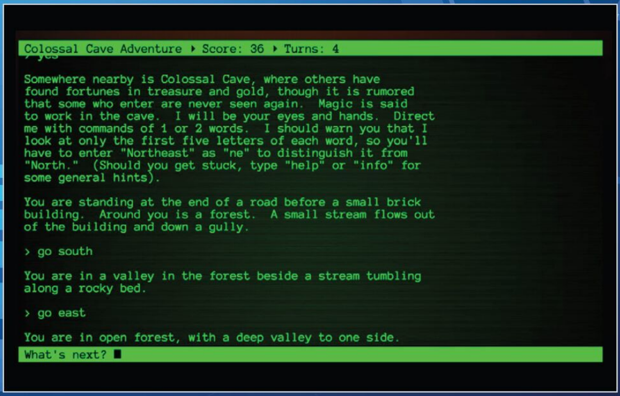
```
Small building

    You are inside the small building.  You discover
    that this is a one room house.  There are broken
    windows in all four walls.  There is debris spread
    over the entire floor and it is obvious that there
    hasn't been anyone here for a long time.  Over in
    the corner there is a large open trophy case.
    There is a large rug covering most of the floor.
    There is a large gas lamp.

           What would you like to do?
           > GET LAMP
```

Hints and help along the way can make the adventure seem less daunting. Of course, if you're particularly mean-spirited, then you can opt to exclude any form of help along the way altogether. However, most text adventures do include some form of a hint. If you've created a great set of puzzles for the player to beat, and they're stuck in their progression of the game, then allowing them to enter the word: help, or hint, into the system, will reveal a small clue as to what to do next. It's not cheating, just keeping the story moving along.

Pacing is the final aspect of the adventure game that needs to be planned correctly. Good pacing within the story is vital to keeping your player's attention, as well as keeping them entertained and engaged with what's going on. It's not an easy egg to crack, however. Too much pace and the player can get lost. Too slow, and they'll get bored and move on to something else.

THE CODE

It's impossible to list the code you'll need for your adventure game since it'll be different from each reader to the next, but here are some pointers to help you out.



```
Colossal Cave Adventure ▸ Score: 36 ▸ Turns: 4
> yes

Somewhere nearby is Colossal Cave, where others have
found fortunes in treasure and gold, though it is rumored
that some who enter are never seen again.  Magic is said
to work in the cave.  I will be your eyes and hands.  Direct
me with commands of 1 or 2 words.  I should warn you that I
look at only the first five letters of each word, so you'll
have to enter "Northeast" as "ne" to distinguish it from
"North."  (Should you get stuck, type "help" or "info" for
some general hints).

You are standing at the end of a road before a small brick
building.  Around you is a forest.  A small stream flows out
of the building and down a gully.

> go south

You are in a valley in the forest beside a stream tumbling
along a rocky bed.

> go east

You are in open forest, with a deep valley to one side.
What's next? █
```

Lists, tuples and dictionaries will be needed to store the various elements of the game. These can be descriptions of characters whom you'll meet, either friend or foe. Locations, with descriptions, throughout the game; items you'll come across that can be used, along with a player inventory, which will grow or lessen depending on what the player has used/dropped/picked up.

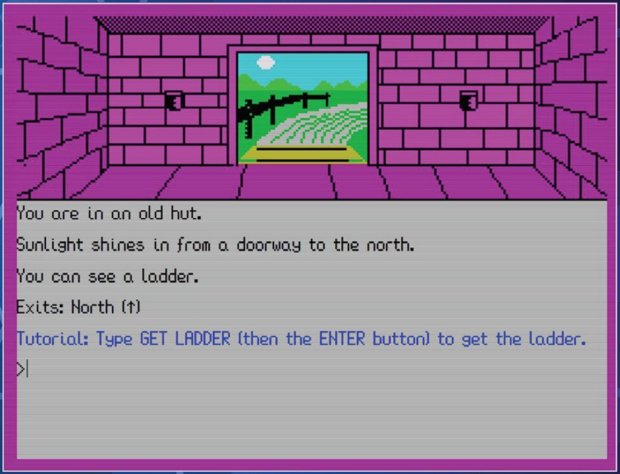
Variables will be needed to hold vital statistics during gameplay, the player's name, amount of strength, endurance, skill and health, for example. You'll also need variables to hold the current score or number of moves used so far, plus any statistics regarding the other characters you'll meet.

Randomly generated numbers are vital for those who want to include an element of combat. Within such examples, a virtual dice roll will determine the winner or loser, either in a fight or when using luck to win through something.

Strings can be used to keep non-moving elements of the game, in other words the names of the rooms you may enter. These components are static throughout the game, each and every time you play, so they can be called upon when the player interacts with them.

Functions are the lifeblood of a text adventure, as they will form the backbone of the locations, interactions, events and anything else that happens within the game.

Loops will keep a process going, such as a fight with an orc, until an event occurs – either the death of the orc or your player. In addition, you'll need to include a loop, if and else components to help out with choices made: do you take the green door or the red door?



```
You are in an old hut.
Sunlight shines in from a doorway to the north.
You can see a ladder.
Exits: North (↑)
Tutorial: Type GET LADDER (then the ENTER button) to get the ladder.
>|
```

GOOD LUCK, ADVENTURER

There's a lot to take in and plan, so our advice is to start small and plan out your story first. Once you've got the basic story, build on the settings and locations, then puzzles and other elements. From there, you can begin to expand the game world and add things like combat.

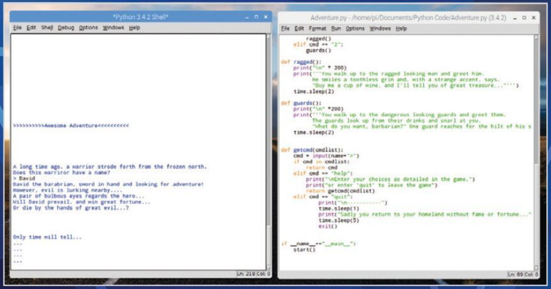
It won't be easy, but it'll be great fun and will help you become a better Python coder.

Text Adventure Script

Text adventures are an excellent way to build your Python coding skills and have some fun at the same time. This example that we created will start you on the path to making a classic text adventure; where it will end is up to you.

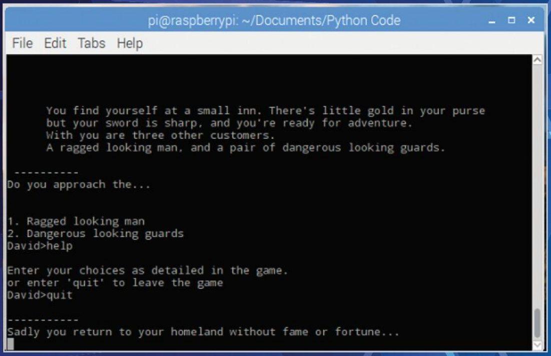
ADVENTURE.PY

The Adventure game uses just the Time module to begin with, creating pauses between print functions. There's a help system in place to expand upon, as well as the story itself.

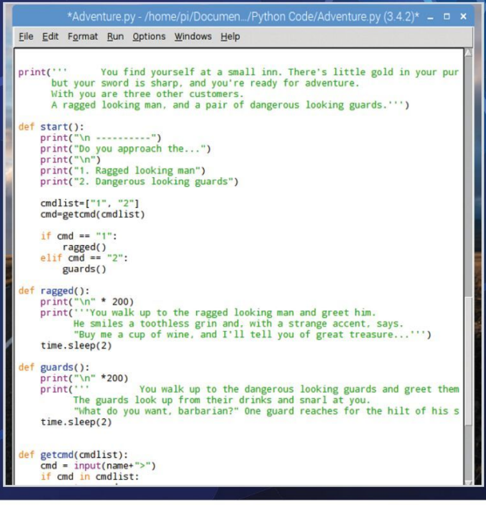


```
>>>>>>>>>>Awesome Adventure<<<<<<<<<<

A long time ago, a warrior strode forth from the frozen north.
Does this warriror have a name?
> David
David the barabrian, sword in hand and looking for adventure!
However, evil is lurking nearby....
A pair of bulbous eyes regards the hero...
Will David prevail, and win great fortune...
Or die by the hands of great evil...?

Only time will tell...
...
...
...
...
```

```
        ragged()
    elif cmd == "2":
        guards()

def ragged():
    print("\n" * 200)
    print('''You walk up to the ragged looking man and greet him.
          He smiles a toothless grin and, with a strange accent, says.
          "Buy me a cup of wine, and I'll tell you of great treasure..."''')
    time.sleep(2)

def guards():
    print("\n" *200)
    print('''You walk up to the dangerous looking guards and greet them.
          The guards look up from their drinks and snarl at you.
          "What do you want, barbarian?" One guard reaches for the hilt of his s
    time.sleep(2)


def getcmd(cmdlist):
    cmd = input(name+">")
    if cmd in cmdlist:
        return cmd
    elif cmd == "help":
        print("\nEnter your choices as detailed in the game.")
        print("or enter 'quit' to leave the game")
        return getcmd(cmdlist)
    elif cmd == "quit":
            print("\n-----------")
            time.sleep(1)
            print("Sadly you return to your homeland without fame or fortune..."
            time.sleep(5)
            exit()


if __name__=="__main__":
    start()
```



```
        You find yourself at a small inn. There's little gold in your purse
        but your sword is sharp, and you're ready for adventure.
        With you are three other customers.
        A ragged looking man, and a pair of dangerous looking guards.

 ----------
Do you approach the...


1. Ragged looking man
2. Dangerous looking guards
David>help

Enter your choices as detailed in the game.
or enter 'quit' to leave the game
David>quit

-----------
Sadly you return to your homeland without fame or fortune...
```



```
print('''      You find yourself at a small inn. There's little gold in your pur
      but your sword is sharp, and you're ready for adventure.
      With you are three other customers.
      A ragged looking man, and a pair of dangerous looking guards.''')

def start():
    print("\n ----------")
    print("Do you approach the...")
    print("\n")
    print("1. Ragged looking man")
    print("2. Dangerous looking guards")

    cmdlist=["1", "2"]
    cmd=getcmd(cmdlist)

    if cmd == "1":
        ragged()
    elif cmd == "2":
        guards()

def ragged():
    print("\n" * 200)
    print('''You walk up to the ragged looking man and greet him.
          He smiles a toothless grin and, with a strange accent, says.
          "Buy me a cup of wine, and I'll tell you of great treasure...''')
    time.sleep(2)

def guards():
    print("\n" *200)
    print('''          You walk up to the dangerous looking guards and greet them
          The guards look up from their drinks and snarl at you.
          "What do you want, barbarian?" One guard reaches for the hilt of his s
    time.sleep(2)


def getcmd(cmdlist):
    cmd = input(name+">")
    if cmd in cmdlist:
```

```
import time

print("\n" * 200)
print(">>>>>>>>>>Awesome Adventure<<<<<<<<<<\n")
print("\n" * 3)
time.sleep(3)
print("\nA long time ago, a warrior strode forth from
the frozen north.")
time.sleep(1)
print("Does this warrior have a name?")
name=input("> ")
print(name, "the barbarian, sword in hand and looking
for adventure!")
time.sleep(1)
print("However, evil is lurking nearby....")
time.sleep(1)
print("A pair of bulbous eyes regards the hero...")
time.sleep(1)
print("Will", name, "prevail, and win great fortune...")
time.sleep(1)
print("Or die by the hands of great evil...?")
time.sleep(1)
print("\n" *3)
print("Only time will tell...")
time.sleep(1)
print('...')
time.sleep(1)
print('...')
time.sleep(1)
print('...')
time.sleep(1)
print('...')
time.sleep(5)
print("\n" *200)

print('''       You find yourself at a small inn. There's
   little gold in your purse but your sword is sharp,
   and you're ready for adventure.
   With you are three other customers.
   A ragged looking man, and a pair of dangerous
   looking guards.''')

def start():
   print("\n ----------")
   print("Do you approach the...")
   print("\n")
   print("1. Ragged looking man")
   print("2. Dangerous looking guards")

   cmdlist=["1", "2"]
   cmd=getcmd(cmdlist)
```

```
    if cmd == "1":
        ragged()
    elif cmd == "2":
        guards()

def ragged():
    print("\n" * 200)
    print('''You walk up to the ragged looking man and
    greet him.
        He smiles a toothless grin and, with a strange
        accent, says.
        "Buy me a cup of wine, and I'll tell you of
        great treasure...''')
    time.sleep(2)

def guards():
    print("\n" *200)
    print('''You walk up to the dangerous looking guards
    and greet them.
        The guards look up from their drinks and
        snarl at you.
        "What do you want, barbarian?" One guard reaches
        for the hilt of his sword...''')
    time.sleep(2)
```

```
def getcmd(cmdlist):
    cmd = input(name+">")
    if cmd in cmdlist:
        return cmd
    elif cmd == "help":
        print("\nEnter your choices as detailed in
        the game.")
        print("or enter 'quit' to leave the game")
        return getcmd(cmdlist)
    elif cmd == "quit":
        print("\n-----------")
        time.sleep(1)
        print("Sadly you return to your homeland without
        fame or fortune...")
        time.sleep(5)
        exit()


if _ _ name _ _ ==" _ _ main _ _ ":
    start()
```

Adventure Time

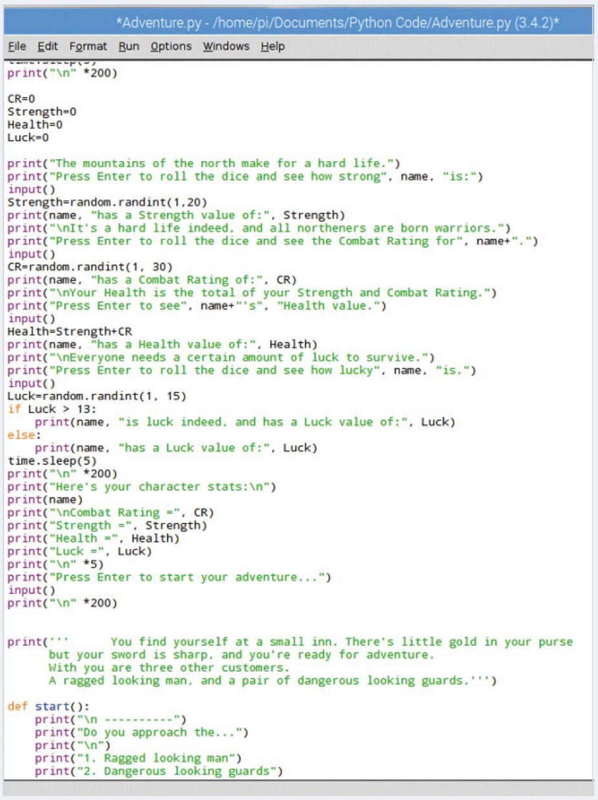
This, as you can see, is just the beginning of the adventure and takes up a fair few lines of code. When you expand it, and weave the story along, you'll find that you can repeat certain instances such as a chance meeting with an enemy or the like.

We've created each of the two encounters as a defined set of functions, along with a list of possible choices under the cmdlist list, and cmd variable, of which is also a defined function. Expanding on this is quite easy, just map out each encounter and choice and create a defined function around it. Providing the user doesn't enter quit into the adventure, they can keep playing.

There's also room in the adventure for a set of variables designed for combat, luck, health, endurance and even an inventory or amount of gold earned. Each successful combat situation can reduce the main character's health but increase their combat skills or endurance. Plus, they could loot the body and gain gold, or earn gold through quests.

Finally, how about introducing the Random module. This will enable you to include an element of chance in the game. For example, in combat, when you strike an enemy you will do a random amount of damage as will they. You could even work out the maths behind improving the chance of a better hit based on your or your opponent's combat skills, current health, strength and endurance. You could create a game of dice in the inn, to see if you win or lose gold (again, improve the chances of winning by working out your luck factor into the equation).

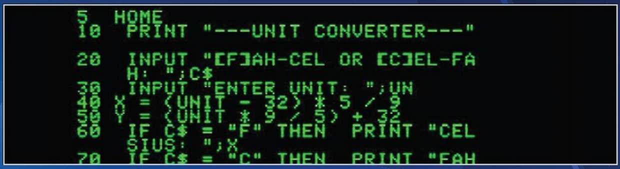
Needless to say, your text adventure can grow exponentially and prove to be a work of wonder. Good luck, and have fun with your adventure.


Adventure.py - /home/pi/Documents/Python Code/Adventure.py (3.4.2)
File Edit Format Run Options Windows Help

```
print("\n" *200)

CR=0
Strength=0
Health=0
Luck=0

print("The mountains of the north make for a hard life.")
print("Press Enter to roll the dice and see how strong", name, "is:")
input()
Strength=random.randint(1,20)
print(name, "has a Strength value of:", Strength)
print("\nIt's a hard life indeed, and all northeners are born warriors.")
print("Press Enter to roll the dice and see the Combat Rating for", name+".")
input()
CR=random.randint(1, 30)
print(name, "has a Combat Rating of:", CR)
print("\nYour Health is the total of your Strength and Combat Rating.")
print("Press Enter to see", name+"'s", "Health value.")
input()
Health=Strength+CR
print(name, "has a Health value of:", Health)
print("\nEveryone needs a certain amount of luck to survive.")
print("Press Enter to roll the dice and see how lucky", name, "is.")
input()
Luck=random.randint(1, 15)
if Luck > 13:
    print(name, "is luck indeed, and has a Luck value of:", Luck)
else:
    print(name, "has a Luck value of:", Luck)
time.sleep(5)
print("\n" *200)
print("Here's your character stats:\n")
print(name)
print("\nCombat Rating =", CR)
print("Strength =", Strength)
print("Health =", Health)
print("Luck =", Luck)
print("\n" *5)
print("Press Enter to start your adventure...")
input()
print("\n" *200)


print('''      You find yourself at a small inn. There's little gold in your purse
      but your sword is sharp, and you're ready for adventure.
      With you are three other customers.
      A ragged looking man, and a pair of dangerous looking guards.''')

def start():
    print("\n ----------")
    print("Do you approach the...")
    print("\n")
    print("1. Ragged looking man")
    print("2. Dangerous looking guards")
```

Retro Coding

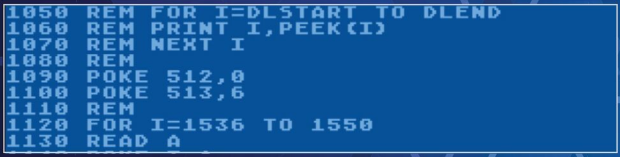
There's a school of thought, that to master the foundations of good coding skills you need to have some experience of how code was written in the past. In the past is a bit of a loose term, but mostly, it means coding from the 80s.



```
5  HOME
10  PRINT "---UNIT CONVERTER---"

20  INPUT "[F]AH-CEL OR [C]EL-FA
    H: ";C$
30  INPUT "ENTER UNIT: ";UN
40 X = (UNIT - 32) * 5 / 9
50 Y = (UNIT * 9 / 5) + 32
60  IF C$ = "F" THEN  PRINT "CEL
    SIUS: ";X
70  IF C$ = "C" THEN  PRINT "FAH
```

THE GOLDEN ERA OF CODE

While it may seem a little counterproductive to learn how to code in a language that's virtually obsolete, there are some surprising benefits to getting your hands dirty with a bit of retro coding. Firstly, learning old code will help you build the structure of code as, regardless of whether it is a language that was developed yesterday, or forty years ago, code still demands strict discipline to work correctly. Secondly, everyday coding elements, such as loops, sub routines and so on, are a great visual aid to learn in older code, especially BASIC. Lastly, it's simply good fun.



```
1050 REM FOR I=DLSTART TO DLEND
1060 REM PRINT I,PEEK(I)
1070 REM NEXT I
1080 REM
1090 POKE 512,0
1100 POKE 513,6
1110 REM
1120 FOR I=1536 TO 1550
1130 READ A
```

GOING BASIC

The easiest retro language to play around with is, without doubt, BASIC. Developed back in the mid-sixties, BASIC (Beginner's All-Purpose Symbolic Instruction Code) is a high-level programming language whose design was geared toward ease of use. In a time when computers were beginning to become more accessible, designers John Kemeny and Thomas Kurtz needed a language that students could get to grips with, quickly and easily. Think of BASIC as a distant relation to Python.

THE BEEB

The problem with BASIC is that there were so many different versions available, across multiple 8-bit platforms, with each having its own unique elements on top of the core BASIC functions. The BASIC that was packaged with the Commodore 64 was different to that on the ZX Spectrum, or the Atari home computers, due to the differing hardware of each system. However, it's widely recognised that one of the 'best', and possibly most utilised, form of BASIC from the 80s was that of BBC BASIC.

BBC BASIC was used on the Acorn BBC Micro range of computers, utilising the MOS 6502-based processor technologies. It was one of the quickest examples of BASIC and, thanks to an inline assembler, it was also capable of allowing the developers of the time to write code for different processor types, such as the Zilog Z80 – a CPU present in the ZX Spectrum, as well as many arcade machines.

The BBC Micro was designed and built by Acorn Computers – a company that is historically responsible for the creation of the ARM CPU - the processor that's used in virtually every Android phone and tablet, smart TV, set top box and so on, as well as the Raspberry Pi – so essentially, the BBC Micro is the grandfather of the Raspberry Pi.

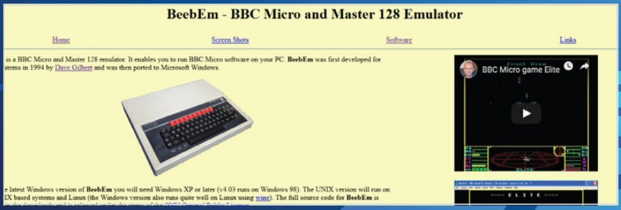
The BBC Micro was born in a time when the UK government was looking for a countrywide computer platform to be used throughout education. Different companies bid, but it was the BBC's Computer Literacy Project (the BBC Micro) that was chosen, due to its ruggedness, upgradability, and potential for education. As a result, the BBC Micro, or the Beeb as it's affectionately known, became the dominant educational computer throughout the 80s.

BEEBEM

Naturally, you could scour eBay and look for a working BBC Micro to play around on, and it'll be a lot of fun. However, for the sake of just getting hands-on with some retro code, we'll use one of the best BBC Micro emulators available: BeebEm.

BeebEm was originally developed for UNIX in 1994 by Dave Gilbert and later ported to Windows. It is now developed by Mike Wyatt and Jon Welch, who maintain the Mac port of the emulator, and is therefore available for Windows 10, Linux and macOS, as well as other platforms.

If you're using Windows 10, simply navigate to **http://www.mkw.me.uk/beebem/index.html**, and download the BeebEM414.exe that's displayed in the main screen.



BeebEm - BBC Micro and Master 128 Emulator

Home | Screen Shots | Software | Links

is a BBC Micro and Master 128 emulator. It enables you to run BBC Micro software on your PC. **BeebEm** was first developed for stems in 1994 by Dave Gilbert and was then ported to Microsoft Windows.

e latest Windows version of **BeebEm** you will need Windows XP or later (v4.03 runs on Windows 98). The UNIX version will run on IX based systems and Linux (the Windows version also runs quite well on Linux using wine). The full source code for **BeebEm** is

Once downloaded, launch the executable and follow the on-screen instructions to install it. MacOS users can get everything they need from: **http://www.g7jjf.com/**. However, Raspberry Pi and Linux users will have to do a little nifty keyboard work before they can enjoy the benefits of the Beeb on their screens. Here's how to get it working under Linux:

First, drop to a Terminal and enter:

```
sudo apt-get update && upgrade
wget http://beebem-unix.bbcmicro.com/download/
beebem-0.0.13.tar.gz
```

Then, extract the compressed files with:

```
tar zxvf beebem-0.0.13.tar.gz, then enter the newly
created Beebem folder with: cd beebem-0.0.13/.
```

Now enter the following lines, hitting Enter and answering Y to accept any changes:

```
sudo apt-get install libgtk2.0-dev libsdl1.2-dev
./configure
make
sudo make install
```

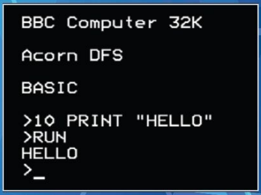
This may take some time, but when it's all done, simply enter: `beebem`, to start the BBC Micro emulator.

BBC BASIC

Once installed and powered up, BeebEm will display the default BBC system start-up, along with a couple of beeps. Those of you old enough to have been in a UK school in the 80s will certainly recall this setup.

In BASIC, we use line numbers to determine which lines of code run in sequence. For example, to print something to screen we'd enter:

```
10 print "hello"
```



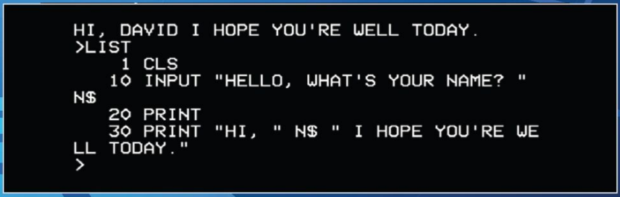
```
BBC Computer 32K

Acorn DFS

BASIC

>10 PRINT "HELLO"
>RUN
HELLO
>_
```

Once you've typed the above in, press Enter and then type:

```
run
```

We can of course expand the code to include variables, multi-line print statements, and so on:

```
1 CLS
10 Input "Hello, what's your name? " n$
20 print
30 print "Hi, " n$ " I hope you're well today."
```



```
HI, DAVID I HOPE YOU'RE WELL TODAY.
>LIST
    1 CLS
   10 INPUT "HELLO, WHAT'S YOUR NAME? "
N$
   20 PRINT
   30 PRINT "HI, " N$ " I HOPE YOU'RE WE
LL TODAY."
>
```

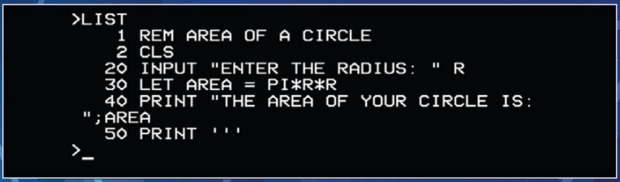
Type RUN to execute the code, you can also type LIST to view the code you've entered.

As you can see, variables are handled in much the same way as Python, a print statement on its own displays a blank line, and CLS clears the screen; although the Pi uses Clear instead of CLS. We're also able to do some maths work, and play around with variables too:

```
1 CLS
10 input "how old are you? " a
20 print
30 if a > 40 print "You're over 40 years old."
40 if a < 40 print "You're under 40 years old."
50 print
```

BBC BASIC also has some interesting built-in features, such as the value of PI:

```
1 REM Area of a circle
2 CLS
20 Input "Enter the radius: " r
30 let area = PI*r*r
40 print "The area of your circle is: "; area
50 print '''
```



```
>LIST
    1 REM AREA OF A CIRCLE
    2 CLS
   20 INPUT "ENTER THE RADIUS: " R
   30 LET AREA = PI*R*R
   40 PRINT "THE AREA OF YOUR CIRCLE IS:
 ";AREA
   50 PRINT '''
>_
```

As you'll notice, variables with a dollar ($) represent strings, nothing after the variable, or a hash (#) represent floating point decimals; a whole integer has a % character, and a byte has an ampersand (&). The single quotes after the Print on line 50 indicate a blank line, one for each tick, while REM on line 1 is a comment, and thus ignored by the BASIC compiler.

Needless to say, there's a lot you can learn, as well as having fun, with BBC BASIC. It's a rainy day project and something that's interesting to show the kids – this is how we rolled back in the 80s, kids!

There are a number of sites you can visit to learn BBC BASIC, such as **http://archive.retro-kit.co.uk/bbc.nvg.org/docs.php3.html**. See what you can come up with using BBC BASIC, or other BASIC types for different systems, and let us know what you've created.

OTHER SYSTEMS

Naturally, you don't have to look to the BBC Micro to play around with some retro code. If you grew up with a Commodore 64, then you can always try VICE, the C64 emulator. Likewise, the ZX Spectrum has a slew of great emulators available for every modern system to play around on. In fact, you can probably find an emulator for virtually every 8-bit or 16-bit machine that was produced over the years. Each has their own unique perspective and coding nuances, so find a few and see what you can create.

128
Tape Loader
128 BASIC
Calculator
48 BASIC
Tape Tester
© 1986 Sinclair Resear
Amstrad 64K Microcomputer (v1)
©1984 Amstrad Consumer Electronics plc and Locomotive Software Ltd.
BASIC 1.0
Ready
**** COMMODORE 64 BASIC V2 ****
64K RAM SYSTEM 38911 BASIC BYTES FREE
READY.
GEM, Graphics Environment Manager
TOS
Copyright (c) 1985
ATARI CORP.
Digital Research, Inc.
All Rights Reserved.

Text Animations

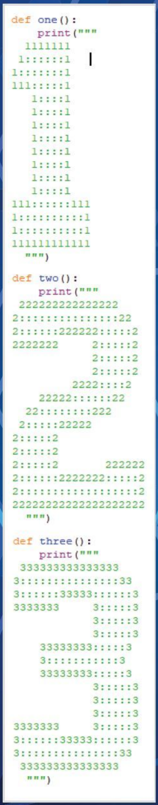
There's a remarkable amount you can do with some simple text and a little Python know-how. Combining what you've already learned, we can create some interesting animation effects from the command line.

THE FINAL COUNTDOWN

Let's begin with some example code that will display a large countdown from ten, then clear the screen and display a message. The code itself is quite simple, but lengthy. You will need to start by importing the OS and Time modules, then start creating functions that display the numbers (see image below) and so on to 10. It'll take some time, but it's worth it in the end. Of course, you can always take a different approach and design the numbers yourself.

The next step of the process is to initialise the code settings and start the countdown:

```
#Initialise settings
start = 10
message = ">        BLAST OFF!!
<"

#Start the countdown
for counter in range(start, 0, -1):
    if counter == 10:
        ten()
    elif counter == 9:
        nine()
    elif counter == 8:
        eight()
    elif counter == 7:
        seven()
    elif counter == 6:
        six()
    elif counter == 5:
        five()
    elif counter == 4:
        four()
    elif counter == 3:
        three()
    elif counter == 2:
        two()
    elif counter == 1:
        one()
    time.sleep(1)
    os.system('cls' if os.name ==
'nt' else 'clear')
```

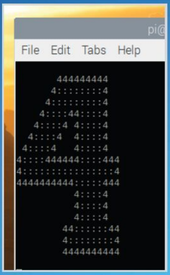


```
def one():
    print("""
  1111111
 1::::::1
1:::::::1
111:::::1
   1::::1
   1::::1
   1::::1
   1::::1
   1::::1
   1::::1
   1::::1
   1::::1
111::::::111
1::::::::::1
1::::::::::1
111111111111
  """)

def two():
    print("""
 222222222222222
2:::::::::::::::22
2::::::222222:::::2
2222222     2:::::2
            2:::::2
            2:::::2
         2222::::2
    22222::::::22
  22::::::::222
 2:::::22222
2:::::2
2:::::2
2:::::2       222222
2::::::2222222:::::2
2::::::::::::::::::2
22222222222222222222
  """)

def three():
    print("""
 333333333333333
3:::::::::::::::33
3::::::33333::::::3
3333333     3:::::3
            3:::::3
            3:::::3
    33333333:::::3
    3:::::::::::3
    33333333:::::3
            3:::::3
            3:::::3
            3:::::3
3333333     3:::::3
3::::::33333::::::3
3:::::::::::::::33
 333333333333333
  """)
```

And finally, we can add a display for the message:

```
#Display the message
print("v^v^v^v^v^v^v^v^v^v^v^v^v^v^v")
print("<                           >")
print(message)
print("<                           >")
print("v^v^v^v^v^v^v^v^v^v^v^v^v^v^v")
print("\n\n\n")
```

The code in its entirety can be viewed from within our Code Repository: **https://bdmpublications.com/code-portal**, where you're free to copy it to your own Python IDLE and use it as you see fit. The end effect is quite good and it'll be worth adding to your own games, or presentations, in Python.



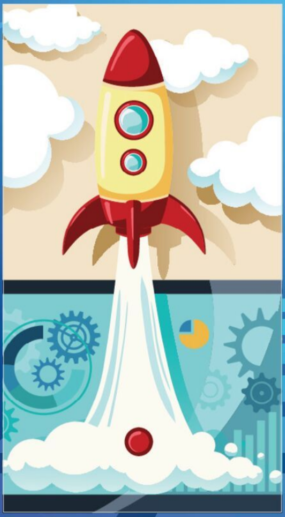
```
       444444444
      4::::::::4
     4:::::::::4
    4::::44::::4
   4::::4 4::::4
  4::::4  4::::4
 4::::4   4::::4
4::::444444::::444
4::::::::::::::::4
4444444444:::::444
          4::::4
          4::::4
          4::::4
        44::::::44
        4::::::::4
        4444444444
```

To extend the code, or make it easier to use, you can always create the number functions in their own Python code, call it Count.py for example, then import Count at the beginning of a new Python file called **Countdown.py**, along with the OS and Time modules:

```
import os
import time
import count
```

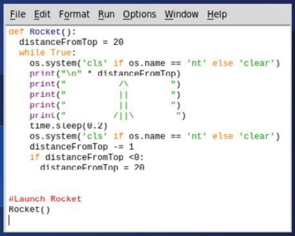
From there, you will need to specify the imported code in the Countdown section:

```
#Start the countdown
for counter in range(start, 0, -1):
    if counter == 10:
        count.ten()
    elif counter == 9:
        count.nine()
    elif counter == 8:
        count.eight()
    elif counter == 7:
        count.seven()
    elif counter == 6:
        count.six()
    elif counter == 5:
        count.five()
    elif counter == 4:
        count.four()
    elif counter == 3:
        count.three()
    elif counter == 2:
        count.two()
    elif counter == 1:
        count.one()
```



ROCKET LAUNCH

Building on the previous countdown example, we can create an animated rocket that'll launch after the Blast Off!! message has been printed. The code would look something like:


File Edit Format Run Options Window Help

```
def Rocket():
  distanceFromTop = 20
  while True:
    os.system('cls' if os.name == 'nt' else 'clear')
    print("\n" * distanceFromTop)
    print("          /\        ")
    print("          ||        ")
    print("          ||        ")
    print("         /||\        ")
    time.sleep(0.2)
    os.system('cls' if os.name == 'nt' else 'clear')
    distanceFromTop -= 1
    if distanceFromTop <0:
      distanceFromTop = 20


#Launch Rocket
Rocket()
```

```
def Rocket():
  distanceFromTop = 20
  while True:
    os.system('cls' if os.name == 'nt' else 'clear')
    print("\n" * distanceFromTop)
    print("          /\        ")
    print("          ||        ")
    print("          ||        ")
    print("         /||\       ")
    time.sleep(0.2)
    os.system('cls' if os.name == 'nt' else 'clear')
    distanceFromTop -= 1
    if distanceFromTop <0:
      distanceFromTop = 20

#Launch Rocket
Rocket()
```

Here, we've created a new function called Rocket, which produces the effect of an ASCII-like rocket taking off and scrolling upwards; using the distanceFromTop variable.

To use this, add it to the end of the previous countdown code and, at the end of the Blast Off!! message, add the following lines:

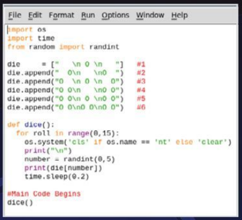
```
print("\n\n\n\n")
input("Press Enter to launch rocket...")
```

This will allow your message to be displayed and then, when the user has hit the Enter button, the rocket will launch.

Again, the code in its entirety can be found in the Code Repository at: **https://bdmpublications.com/ code-portal**.

ROLLING DIE

Aside from the rocket animation, together with its countdown, another fun bit of text-based animation is that of a rolling dice.


File Edit Format Run Options Window Help

```
import os
import time
from random import randint

die      = ["   \n O \n   "]   #1
die.append("  O\n   \nO  ")   #2
die.append("O  \n O \n  O")   #3
die.append("O O\n   \nO O")   #4
die.append("O O\n O \nO O")   #5
die.append("O O\nO O\nO O")   #6

def dice():
  for roll in range(0,15):
    os.system('cls' if os.name == 'nt' else 'clear')
    print("\n")
    number = randint(0,5)
    print(die[number])
    time.sleep(0.2)

#Main Code Begins
dice()
```

A rolling dice can be a great animation to include in an adventure game, where the player rolls to see what their score is compared to that of an enemy. The highest roller wins the round and the losers' health drops as a result. It's an age-old combat sequence, used mainly in the Dungeon and Dragons board games and Fighting Fantasy novels, but it works well.

The code you'll need to animate a dice roll is:

```
import os
import time
from random import randint

die      = ["   \n O \n   "]   #1
die.append("  O\n   \nO  ")   #2
die.append("O  \n O \n  O")   #3
die.append("O O\n   \nO O")   #4
die.append("O O\n O \nO O")   #5
die.append("O O\nO O\nO O")   #6

def dice():
  for roll in range(0,15):
    os.system('cls' if os.name == 'nt' else 'clear')
    print("\n")
    number = randint(0,5)
    print(die[number])
    time.sleep(0.2)

#Main Code Begins
dice()
```

You may need to tweak the O entries, to line up the dots on the virtual dice. Once it's done, though, you'll be able to add this function to your adventure game code and call it up whenever your character, or the situation, requires some element of luck, combat, or chance roll of the dice.

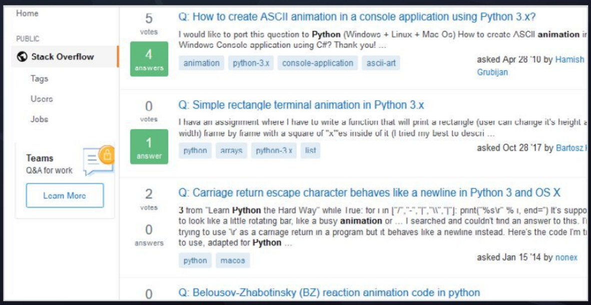
DISCOVER ANIMATIONS

The great thing about Python code is that it's so accessible. These few examples will help you add some fun, or something different, to your programs, but they're just the tip of the proverbial iceberg. If there's something you want to include in your code, and you're at a sticking point, then consider heading over to Stack Overflow and search for Python 3 content.

Stack Overflow is a great online help and resource portal, where you can ask questions and experts will try and help you. It doesn't always work out, but most of the time, you'll find what you're looking for within this great resource.

Also, if you're after a simple animation then take to Google and spend some time searching for it. While you may not find exactly what it is you're after, you're bound to come across something very like the desired effect; all you need to do is modify it slightly to accomplish your own goals.

You'll find that many professionals will often take to the Internet to find content they're after. True, they're able to code it themselves, but even experts get stuck sometimes, so don't worry about hunting code snippets down; you're in good company.



5 votes 4 answers
Q: How to create ASCII animation in a console application using Python 3.x?
I would like to port this question to **Python** (Windows + Linux + Mac Os) How to create ASCII **animation** in Windows Console application using C#? Thank you! ...
animation python-3.x console-application ascii-art
asked Apr 28 '10 by Hamish Grubijan

0 votes 1 answer
Q: Simple rectangle terminal animation in Python 3.x
I have an assignment where I have to write a function that will print a rectangle (user can change it's height a width) frame by frame with a square of "x"'es inside of it (I tried my best to descri ...
python arrays python-3.x list
asked Oct 28 '17 by Bartosz I

2 votes 0 answers
Q: Carriage return escape character behaves like a newline in Python 3 and OS X
3 from "Learn **Python** the Hard Way" while True: for i in ["/","-","|","\\","|"]: print("%s\r" % i, end='') It's suppo to look like a little rotating bar, like a busy **animation** or ... I searched and couldn't find an answer to this. I'i trying to use '\r' as a carriage return in a program but it behaves like a newline instead. Here's the code I'm t to use, adapted for **Python** ...
python macos
asked Jan 15 '14 by nonex

0
Q: Belousov-Zhabotinsky (BZ) reaction animation code in python

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

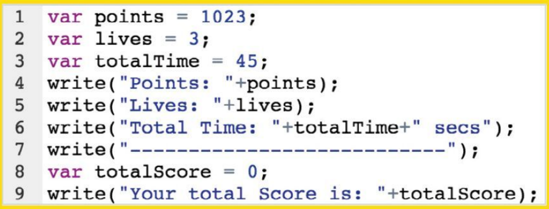
There are many pitfalls for the programmer to be aware of, far too many to be listed here. Being able to recognise a mistake and fix it is when you start to move into more advanced territory.

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

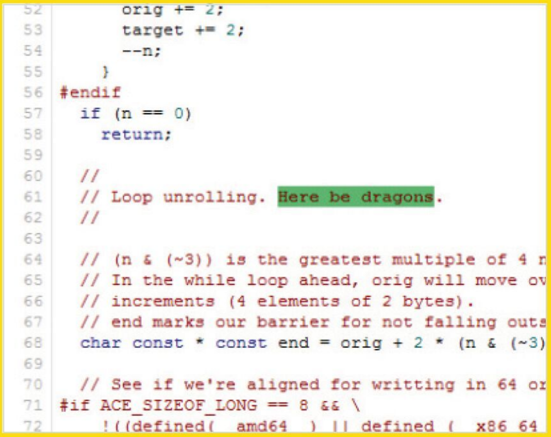
Meaningful naming for variables is a must to eliminate common coding mistakes. Having letters of the alphabet is fine but what happens when the code states there's a problem with x variable. It's not too difficult to name variables lives, money, player1 and so on.



```
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

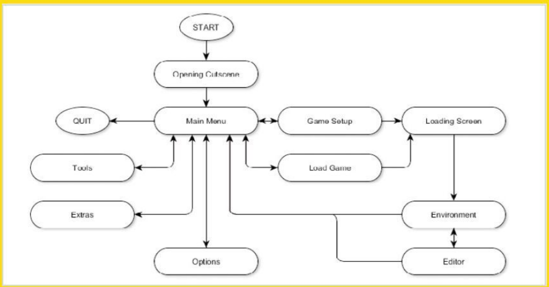
Use comments. It's a simple concept but commenting on your code saves so many problems when you next come to look over it. Inserting comment lines helps you quickly sift through the sections of code that are causing problems; also useful if you need to review an older piece of code.



```
            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
```

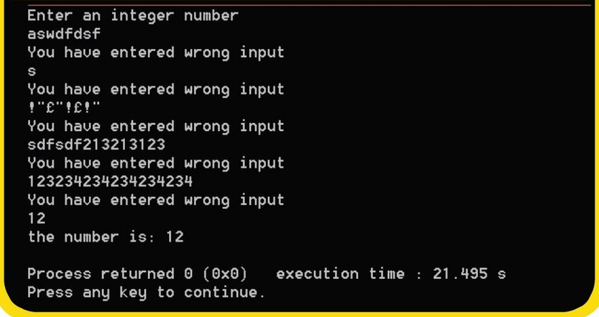
PLAN AHEAD

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

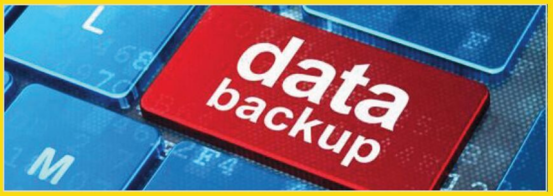
User input is often a paralysing mistake in code. For example, when the user is supposed to enter a number for their age and instead they enter it in letters. Often a user can enter so much into an input that it overflows some internal buffer, thus sending the code crashing. Watch those user inputs and 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.
```

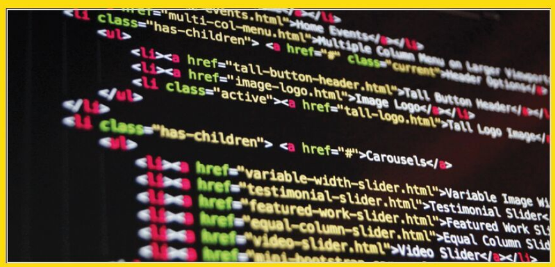
BACKUPS

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

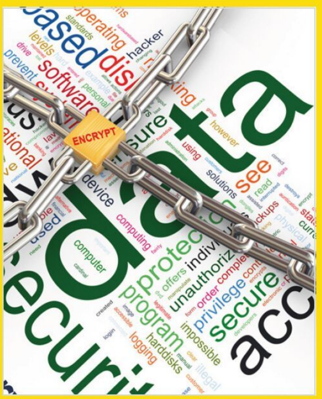
You can easily spend days trying to fathom out a section of code to achieve a given result and it's frustrating and often time-wasting. While it's equally rewarding to solve the problem yourself, often the same code is out there on the Internet somewhere. Don't try and re-invent the wheel, look to see if some else has done it first.



```
href="events.html">Events</a></li>
<li class="has-children"> <a href="multi-col-menu.html">Multiple Column Menu on Larger Viewport
<ul>
<li><a href="#" class="current">Header Options</a>
<li><a href="tall-button-header.html">Tall Button Header</a></li>
<li><a href="image-logo.html">Image Logo</a></li>
<li class="active"><a href="tall-logo.html">Tall Logo Image</a></li>
</ul>
</li>
<li class="has-children"> <a href="#">Carousels</a>
<ul>
<li><a href="variable-width-slider.html">Variable Image Wi
<li><a href="testimonial-slider.html">Testimonial Slider<
<li><a href="featured-work-slider.html">Featured Work Sl
<li><a href="equal-column-slider.html">Equal Column Slid
<li><a href="video-slider.html">Video Slider</a></li>
```

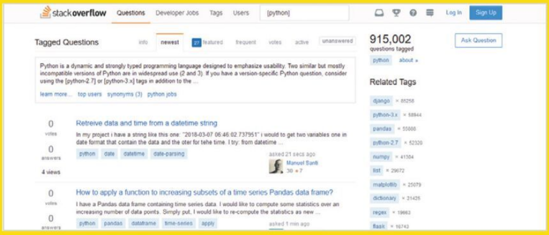
SECURE DATA

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 query in the correct manner, obey any forum rules and be polite, then your question isn't silly.



915,002 questions tagged python about »

Python is a dynamic and strongly typed programming language designed to emphasize usability. Two similar but mostly incompatible versions of Python are in widespread use (2 and 3). If you have a version-specific Python question, consider using the [python-2.7] or [python-3.x] tags in addition to the ...

learn more... top users synonyms (3) python jobs

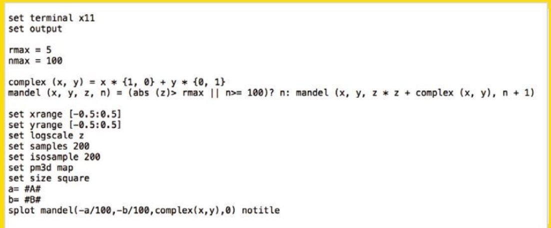
0 votes 0 answers 4 views — Retreive data and time from a datetime string
In my project i have a string like this one: "2018-03-07 06:46:02.737951" i would to get two variables one in date format that contain the data and the oter for tehe time. I try: from datetime ...
python date datetime date-parsing — asked 21 secs ago Manuel Santi 30 ●7

0 votes 0 answers — How to apply a function to increasing subsets of a time series Pandas data frame?
I have a Pandas data frame containing time series data. I would like to compute some statistics over an increasing number of data points. Simply put, I would like to re-compute the statistics as new ...
python pandas dataframe time-series apply — asked 1 min ago

Related Tags
django × 85258
python-3.x × 58944
pandas × 55688
python-2.7 × 52320
numpy × 41304
list × 29672
matplotlib × 25079
dictionary × 21435
regex × 19663
flask × 16743

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
```

Python Beginner's Mistakes

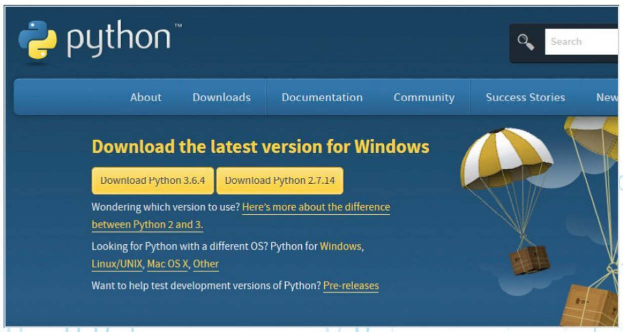
Python is a relatively easy language to get started in where there's plenty of room for the beginner to find their programming feet. However, as with any other programming language, it can be easy to make common mistakes that'll stop your code from running.

DEF BEGINNER(MISTAKES=10)

Here are ten common Python programming mistakes most beginners find themselves making. Being able to identify these mistakes will save you headaches in the future.

VERSIONS

To add to the confusion that most beginners already face when coming into programming, Python has two live versions of its language available to download and use. There is Python version 2.7.x and Python 3.6.x. The 3.6.x version is the most recent, and the one we'd recommend starting. But, version 2.7.x code doesn't always work with 3.6.x code and vice versa.



Download the latest version for Windows

Download Python 3.6.4 Download Python 2.7.14

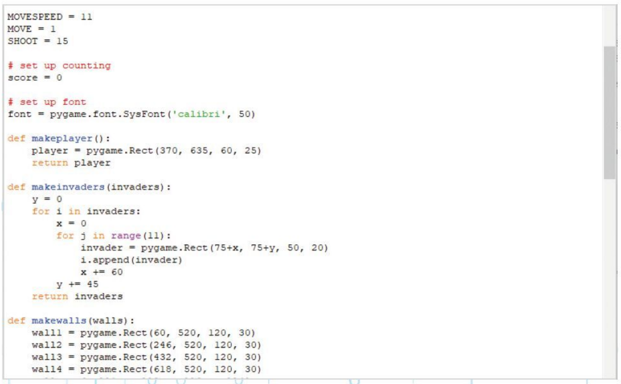
Wondering which version to use? Here's more about the difference between Python 2 and 3.

Looking for Python with a different OS? Python for Windows, Linux/UNIX, Mac OS X, Other

Want to help test development versions of Python? Pre-releases

INDENTS, TABS AND SPACES

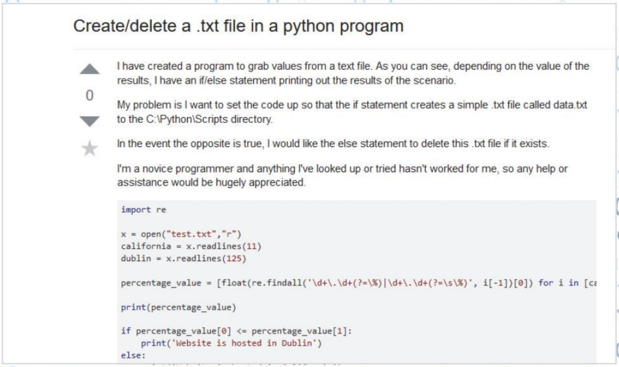
Python uses precise indentations when displaying its code. The indents mean that the code in that section is a part of the previous statement, and not something linked with another part of the code. Use four spaces to create an indent, not the Tab key.



```
MOVESPEED = 11
MOVE = 1
SHOOT = 15

# set up counting
score = 0

# set up font
font = pygame.font.SysFont('calibri', 50)

def makeplayer():
    player = pygame.Rect(370, 635, 60, 25)
    return player

def makeinvaders(invaders):
    y = 0
    for i in invaders:
        x = 0
        for j in range(11):
            invader = pygame.Rect(75+x, 75+y, 50, 20)
            i.append(invader)
            x += 60
        y += 45
    return invaders

def makewalls(walls):
    wall1 = pygame.Rect(60, 520, 120, 30)
    wall2 = pygame.Rect(246, 520, 120, 30)
    wall3 = pygame.Rect(432, 520, 120, 30)
    wall4 = pygame.Rect(618, 520, 120, 30)
```

THE INTERNET

Every programmer has and does at some point go on the Internet and copy some code to insert into their own routines. There's nothing wrong with using others' code, but you need to know how the code works and what it does before you go blindly running it on your own computer.



Create/delete a .txt file in a python program

0

I have created a program to grab values from a text file. As you can see, depending on the value of the results, I have an if/else statement printing out the results of the scenario.

My problem is I want to set the code up so that the if statement creates a simple .txt file called data.txt to the C:\Python\Scripts directory.

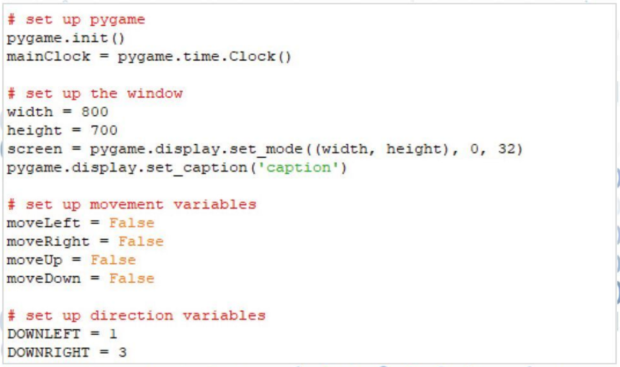
In the event the opposite is true, I would like the else statement to delete this .txt file if it exists.

I'm a novice programmer and anything I've looked up or tried hasn't worked for me, so any help or assistance would be hugely appreciated.

```
import re

x = open("test.txt","r")
california = x.readlines(11)
dublin = x.readlines(125)

percentage_value = [float(re.findall('\d+\.\d+(?=\%)|\d+\.\d+(?=\s\%)', i[-1])[0]) for i in [ca

print(percentage_value)

if percentage_value[0] <= percentage_value[1]:
    print('Website is hosted in Dublin')
else:
```

COMMENTING

Again we mention commenting. It's a hugely important factor in programming, even if you're the only one who is ever going to view the code, you need to add comments as to what's going on. Is this function where you lose a life? Write a comment and help you, or anyone else, see what's going on.



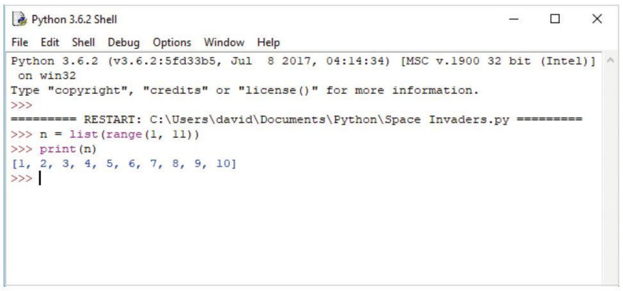
```
# set up pygame
pygame.init()
mainClock = pygame.time.Clock()

# set up the window
width = 800
height = 700
screen = pygame.display.set_mode((width, height), 0, 32)
pygame.display.set_caption('caption')

# set up movement variables
moveLeft = False
moveRight = False
moveUp = False
moveDown = False

# set up direction variables
DOWNLEFT = 1
DOWNRIGHT = 3
```

COUNTING LOOPS

Remember that in Python a loop doesn't count the last number you specify in a range. So if you wanted the loop to count from 1 to 10, then you will need to use:

```
n = list(range(1, 11))
```

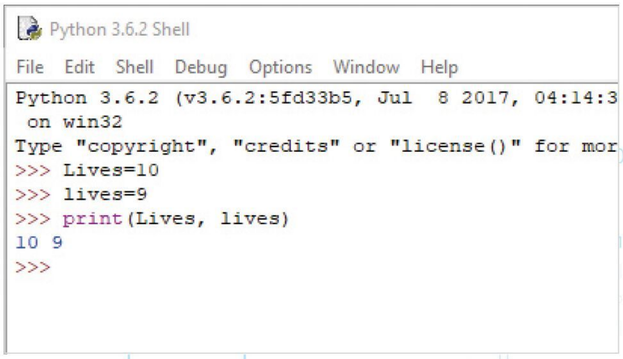
Which will return 1 to 10.



```
Python 3.6.2 (v3.6.2:5fd33b5, Jul  8 2017, 04:14:34) [MSC v.1900 32 bit (Intel)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>>
========= RESTART: C:\Users\david\Documents\Python\Space Invaders.py =========
>>> n = list(range(1, 11))
>>> print(n)
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> |
```

CASE SENSITIVE

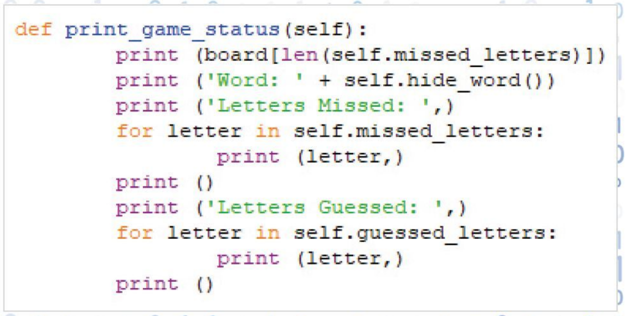
Python is a case sensitive programming language, so you will need to check any variables you assign. For example, `Lives=10` is a different variable to lives=10, calling the wrong variable in your code can have unexpected results.



```
Python 3.6.2 (v3.6.2:5fd33b5, Jul  8 2017, 04:14:3
 on win32
Type "copyright", "credits" or "license()" for mor
>>> Lives=10
>>> lives=9
>>> print(Lives, lives)
10 9
>>>
```

BRACKETS

Everyone forgets to include that extra bracket they should have added to the end of the statement. Python relies on the routine having an equal amount of closed brackets to open brackets, so any errors in your code could be due to you forgetting to count your brackets; including square brackets.



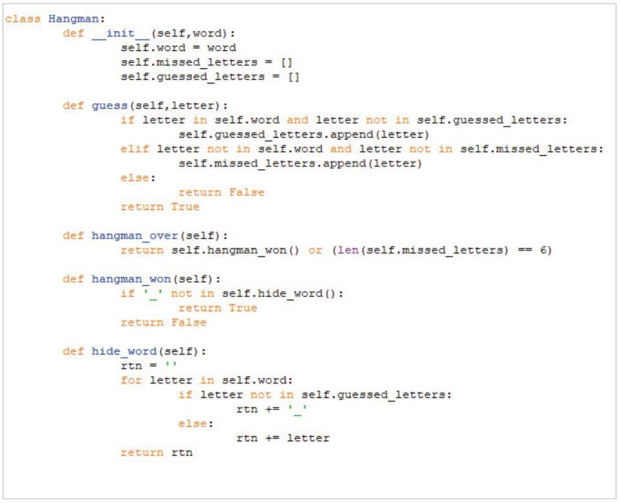
```
def print_game_status(self):
        print (board[len(self.missed_letters)])
        print ('Word: ' + self.hide_word())
        print ('Letters Missed: ',)
        for letter in self.missed_letters:
                print (letter,)
        print ()
        print ('Letters Guessed: ',)
        for letter in self.guessed_letters:
                print (letter,)
        print ()
```

COLONS

It's common for beginners to forget to add a colon to the end of a structural statement, such as:

```
class Hangman:
def guess(self, letter):
```

And so on. The colon is what separates the code, and creates the indents to which the following code belongs to.



```
class Hangman:
        def __init__(self,word):
                self.word = word
                self.missed_letters = []
                self.guessed_letters = []

        def guess(self,letter):
                if letter in self.word and letter not in self.guessed_letters:
                        self.guessed_letters.append(letter)
                elif letter not in self.word and letter not in self.missed_letters:
                        self.missed_letters.append(letter)
                else:
                        return False
                return True

        def hangman_over(self):
                return self.hangman_won() or (len(self.missed_letters) == 6)

        def hangman_won(self):
                if '_' not in self.hide_word():
                        return True
                return False

        def hide_word(self):
                rtn = ''
                for letter in self.word:
                        if letter not in self.guessed_letters:
                                rtn += '_'
                        else:
                                rtn += letter
                return rtn
```

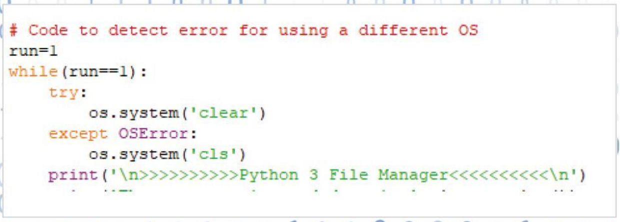
OPERATORS

Using the wrong operator is also a common mistake to make. When you're performing a comparison between two values, for example, you need to use the equality operator (a double equals, ==). Using a single equal (=) is an assignment operator that places a value to a variable (such as, lives=10).

```
b = 5
c = 10
d = 10
b == c #false because 5 is not equal to 10
c == d #true because 10 is equal to 10
```

OPERATING SYSTEMS

Writing code for multiple platforms is difficult, especially when you start to utilise the external commands of the operating system. For example, if your code calls for the screen to be cleared, then for Windows you would use `cls`. Whereas, for Linux you need to use `clear`. You need to solve this by capturing the error and issuing it with an alternative command.



```
# Code to detect error for using a different OS
run=1
while(run==1):
    try:
        os.system('clear')
    except OSError:
        os.system('cls')
    print('\n>>>>>>>>>>Python 3 File Manager<<<<<<<<<<\n')
```

C++ Beginner's Mistakes

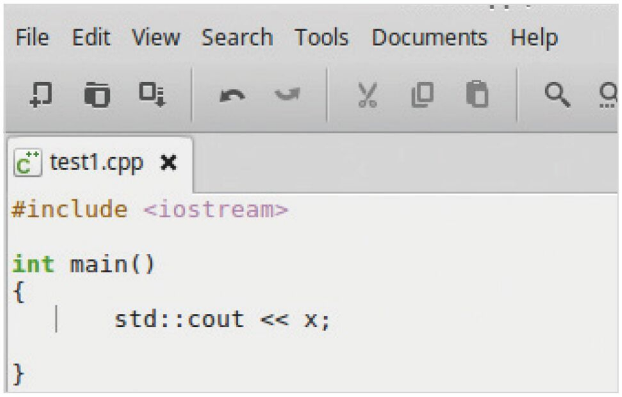
There are many pitfalls the C++ developer can encounter, especially as this is a more complex and often unforgiving language to master. Beginners need to take C++ a step at a time and digest what they've learned before moving on.

VOID(C++, MISTAKES)

Admittedly it's not just C++ beginners that make the kinds of errors we outline on these pages, even hardened coders are prone to the odd mishap here and there. Here are some common issues to try and avoid.

UNDECLARED IDENTIFIERS

A common C++ mistake, and to be honest a common mistake with most programming languages, is when you try and output a variable that doesn't exist. Displaying the value of x on-screen is fine but not if you haven't told the compiler what the value of x is to begin with.



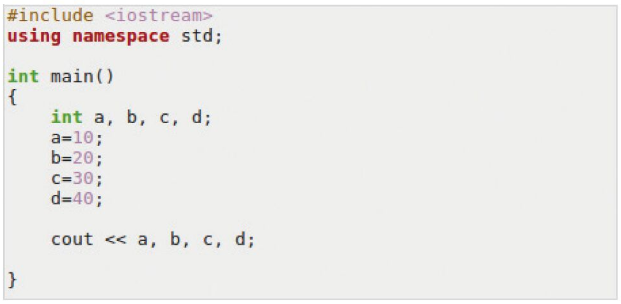
```
#include <iostream>

int main()
{
        std::cout << x;

}
```

STD NAMESPACE

Referencing the Standard Library is common for beginners throughout their code, but if you miss the std:: element of a statement, your code errors out when compiling. You can combat this by adding:

```
using namespace std;
```

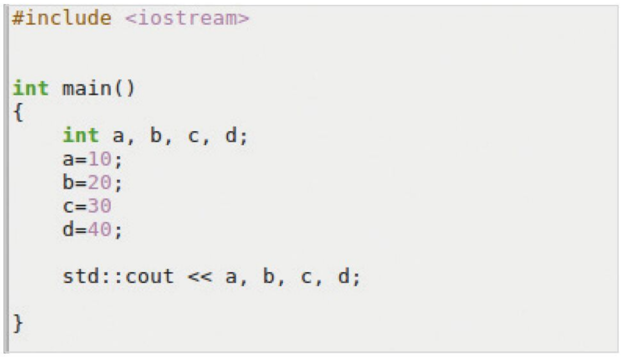
Under the #include part and simply using cout, cin and so on from then on.



```
#include <iostream>
using namespace std;

int main()
{
    int a, b, c, d;
    a=10;
    b=20;
    c=30;
    d=40;

    cout << a, b, c, d;

}
```

SEMICOLONS

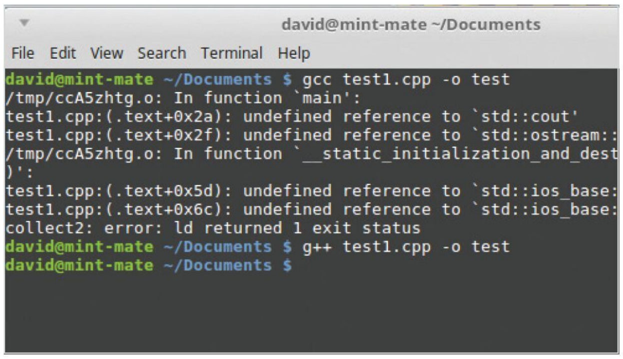
Remember that each line of a C++ program must end with a semicolon. If it doesn't then the compiler treats the line with the missing semicolon as the same line with the next semicolon on. This creates all manner of problems when trying to compile, so don't forget those semicolons.



```
#include <iostream>


int main()
{
    int a, b, c, d;
    a=10;
    b=20;
    c=30
    d=40;

    std::cout << a, b, c, d;

}
```

GCC OR G++

If you're compiling in Linux then you will no doubt come across gcc and g++. In short, gcc is the Gnu Compiler Collection (or Gnu C Compiler as it used to be called) and g++ is the Gnu ++ (the C++ version) of the compiler. If you're compiling C++ then you need to use g++, as the incorrect compiler drivers will be used.


david@mint-mate ~/Documents
File Edit View Search Terminal Help

```
david@mint-mate ~/Documents $ gcc test1.cpp -o test
/tmp/ccA5zhtg.o: In function `main':
test1.cpp:(.text+0x2a): undefined reference to `std::cout'
test1.cpp:(.text+0x2f): undefined reference to `std::ostream::
/tmp/ccA5zhtg.o: In function `__static_initialization_and_dest
)':
test1.cpp:(.text+0x5d): undefined reference to `std::ios_base:
test1.cpp:(.text+0x6c): undefined reference to `std::ios_base:
collect2: error: ld returned 1 exit status
david@mint-mate ~/Documents $ g++ test1.cpp -o test
david@mint-mate ~/Documents $
```

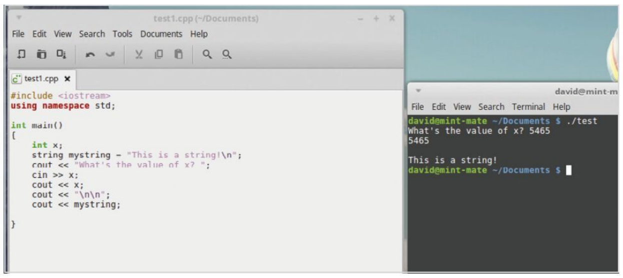
COMMENTS (AGAIN)

Indeed the mistake of never making any comments on code is back once more. As we've previously bemoaned, the lack of readable identifiers throughout the code makes it very difficult to look back at how it worked, for both you and someone else. Use more comments.

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int main()
{
    vector<double> v;

    double d;
    while(cin>>d) v.push_back(d);   // read elements
    if (!cin.eof()) {             // check if input failed
        cerr << "format error\n";
        return 1;                   // error return
    }

    cout << "read " << v.size() << " elements\n";

    reverse(v.begin(),v.end());
    cout << "elements in reverse order:\n";
    for (int i = 0; i<v.size(); ++i) cout << v[i] << '\n';

    return 0;                       // success return
}
```

QUOTES

Missing quotes is a common mistake to make, for every level of user. Remember that quotes need to encase strings and anything that's going to be outputted to the screen or into a file, for example. Most compilers errors are due to missing quotes in the code.


test1.cpp (~/Documents)
File Edit View Search Tools Documents Help
test1.cpp
david@mint-m
File Edit View Search Terminal Help

```
#include <iostream>
using namespace std;

int main()
{
    int x;
    string mystring = "This is a string!\n";
    cout << "What's the value of x? ";
    cin >> x;
    cout << x;
    cout << "\n\n";
    cout << mystring;

}
```

```
david@mint-mate ~/Documents $ ./test
What's the value of x? 5465
5465

This is a string!
david@mint-mate ~/Documents $ 
```

EXTRA SEMICOLONS

While it's necessary to have a semicolon at the end of every C++ line, there are some exceptions to the rule. Semicolons need to be at the end of every complete statement but some lines of code aren't complete statements. Such as:

`#include`
`if lines`
`switch lines`

If it sounds confusing don't worry, the compiler lets you know where you went wrong.

```
 // Program to print positive number entered by the user
 // If user enters negative number, it is skipped

 #include <iostream>
 using namespace std;

 int main()
 {
      int number;
      cout << "Enter an integer: ";
      cin >> number;

      // checks if the number is positive
      if ( number > 0)
      {
           cout << "You entered a positive integer: " << number << endl;
      }

      cout << "This statement is always executed.";
      return 0;

 }
```

TOO MANY BRACES

The braces, or curly brackets, are beginning and ending markers around blocks of code. So for every { you must have a }. Often it's easy to include or miss out one or the other facing brace when writing code; usually when writing in a text editor, as an IDE adds them for you.

```
#include <iostream>
using namespace std;

int main()
{
    int x;
    string mystring = "This is a string!\n";
    cout << "What's the value of x? ";
    cin >> x;
    cout << x;
    {
    cout << "\n\n";
    cout << mystring;

}
```

INITIALISE VARIABLES

In C++ variables aren't initialised to zero by default. This means if you create a variable called x then, potentially, it is given a random number from 0 to 18,446,744,073,709,551,616, which can be difficult to include in an equation. When creating a variable, give it the value of zero to begin with: **`x=0`**.

```
#include <iostream>
using namespace std;

int main()
{
    int x;
    x=0;

    cout << x;

}
```

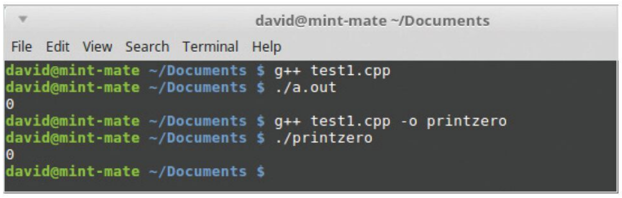
A.OUT

A common mistake when compiling in Linux is forgetting to name your C++ code post compiling. When you compile from the Terminal, you enter:

`g++ code.cpp`

This compiles the code in the file code.cpp and create an a.out file that can be executed with ./a.out. However, if you already have code in a.out then it's overwritten. Use:

`g++ code.cpp -o nameofprogram`


david@mint-mate ~/Documents
File Edit View Search Terminal Help

```
david@mint-mate ~/Documents $ g++ test1.cpp
david@mint-mate ~/Documents $ ./a.out
0
david@mint-mate ~/Documents $ g++ test1.cpp -o printzero
david@mint-mate ~/Documents $ ./printzero
0
david@mint-mate ~/Documents $
```

Coding and Programming Tricks & Tips
13th Edition | ISBN: 978-1-912847-53-2

Published by: Papercut Limited
Digital distribution by: Readly AB

Papercut Limited
Registered in England & Wales No: 04308513

ADVERTISING - For our latest media packs please contact:
Richard Rowe - richard@tandemmedia.co.uk
Will Smith - will@tandemmedia.co.uk

INTERNATIONAL LICENSING - Papercut Limited has many great publications and all are available for licensing worldwide.
For more information email: jgale@pclpublications.com