# 275.1

# Foundations - Computers, Technology, & Security Book 1

SANS | GIAC CERTIFICATIONS

THE MOST TRUSTED SOURCE FOR INFORMATION SECURITY TRAINING, CERTIFICATION, AND RESEARCH | sans.org

# SANS Foundations - Book 1

# Table of Contents

# Introduction

# Learning the Foundations

## Welcome to SANS Foundations

Welcome to the course! This syllabus was curated and the platform architected to provide you with a fast track to learning the foundations. Whether you plan to go on to study offensive security, forensics -- or perhaps a related profession such as development or engineering, SANS Foundations provides you with the background knowledge you need to enter the profession and accelerate your learning.

From the basics of networking, familiarisation with programming and scripting, through to basic security concepts, you will learn the language and meaning of terms that get you started within these roles.

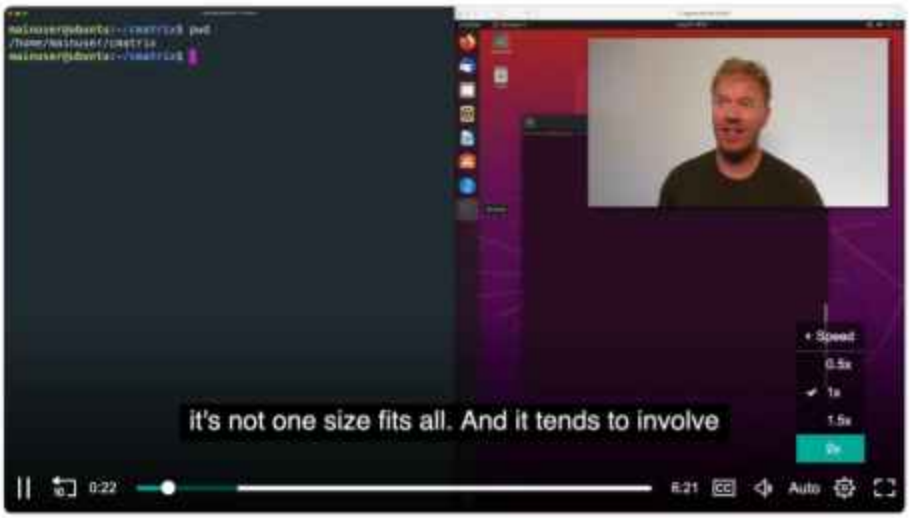
Every day, as a security practitioner and a developer, I make use of my Linux command line skills, and the ability to pop together a tool using Python has solved so many problems. In this platform, you will learn the theory, and get the chance to practise until you are a master of the foundations, and ready to springboard into your future career.

Good luck, and have a wonderful learning journey!

James Lyne, SANS CTO.

## Videos, Audio and eBook

The course platform provides you with a number of different formats to engage with the content. You will get videos, audio to listen to, and examples to read. Some videos are in lecture format where we share stories and examples, but there are also lots of demonstration modules where you can watch how it is done. Please note that subtitles and speed controls are available in the player for you to customise your experience:




As you step through the course you will be exposed to different formats of video, styles of study and material. Each of these will layer together to help you learn the foundations, and then supply you with lots of opportunities to practise!

## Lab System

The integrated lab system provides you with the opportunity to apply the theory and try for yourself. Using your browser, you will be provided with access to a personalised Linux system. Simply click into it and start typing! The user interface looks like this:

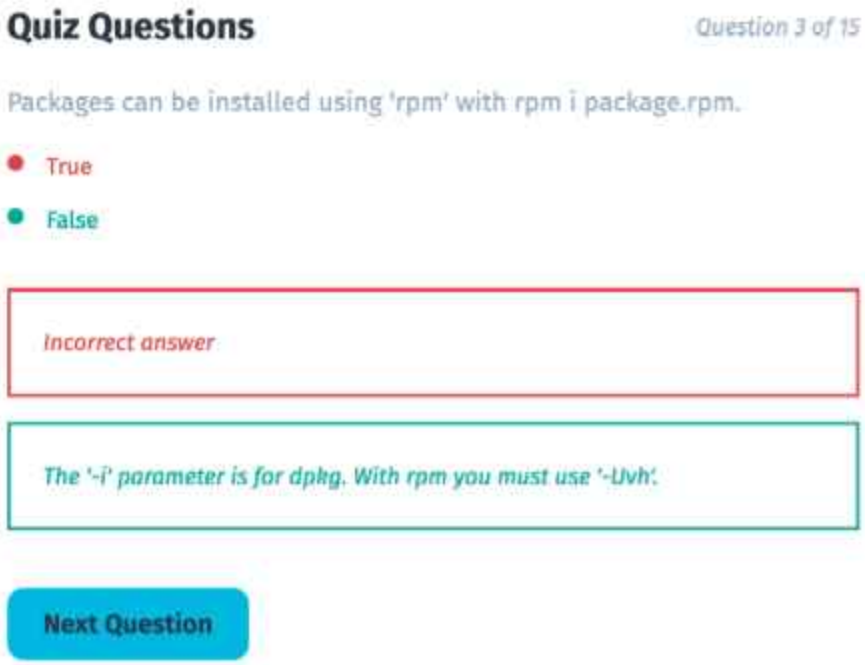

There are often lab steps provided, which can be navigated through step-by-step. Alternatively you can treat the lab as a challenge and try to make it on your own. You can always reset your lab environment if it goes wrong! Lab steps look like this:


In some modules, you will find you need to code solutions, not just use the command line. In this case, you will be provided with an editor and a terminal. You can switch between them to write, then run, your code. The interface looks like this:


Your connection will need to be reliable to use the lab system, but if anything goes wrong you can always re-load and get access to the system again. Also, please remember you are welcome to go off the rails in the labs, and try things you are interested in! Practice sharpens your skills!

## Quizzes

The quizzes are designed for you to challenge your knowledge, and check whether you understood the material from a module. They are not an examination, and the scores are not used as part of a final grading. You can take the quiz as many times as you would like, and you should treat it as a useful measure of whether you need to do more work. When you provide an answer, you are given feedback to help you learn. Here is an example:



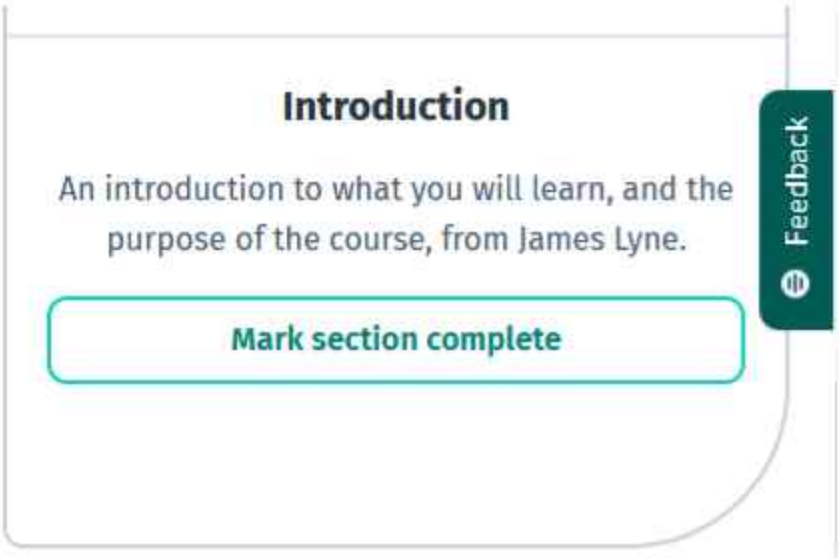

You can go back to a module, section, or a quiz at any time.

## Your Lab Environment

As you study the SANS Foundations course, you are going to get plenty of opportunities to practice hands on. This section introduces you to the lab environment, and how to write code, execute programs, and make the most of your learning systems. Worry not, you will get plenty of practice using these features, and you can always jump back here and watch the video again if you get stuck!

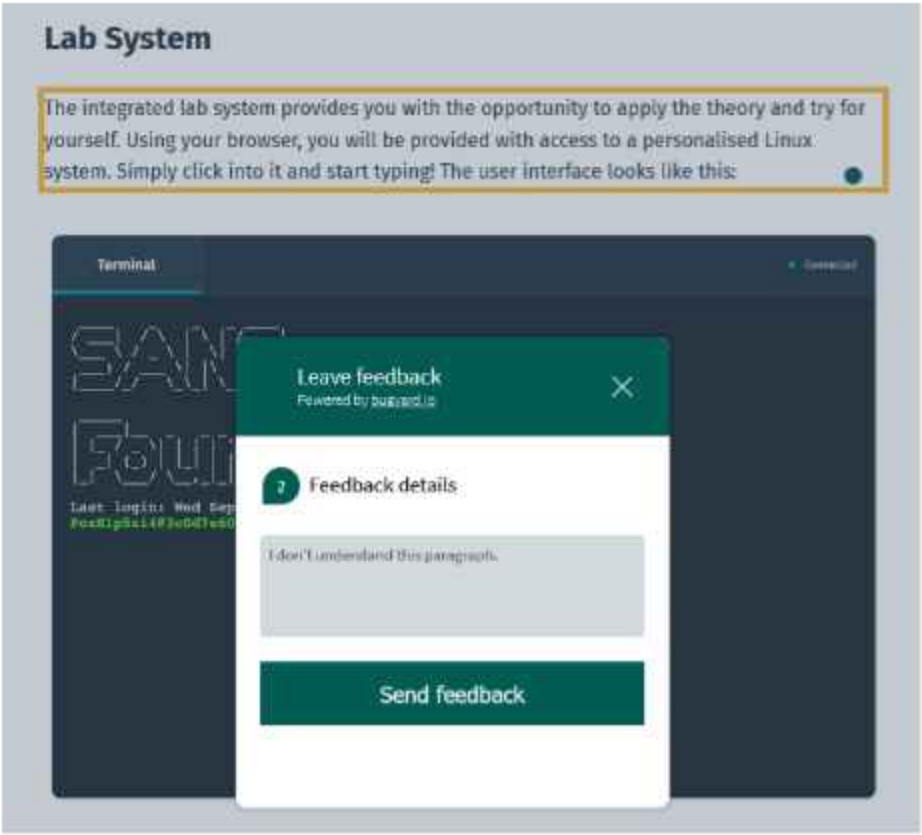
## How to get Help

If you get stuck, confused or find an error, then there are multiple ways you can let us know. Firstly, on every page in the platform, there is a feedback tab on the right hand side of your screen:




You can click this to let us know how you are doing. You can even use it to tell us that something just clicked and you feel fantastic!

You can write a comment about the page or the content. If you want, you can also click and select a particular element on the page, for example highlighting that a particular paragraph of text left you confused.




These comments will be sent to our team and used to address content improvements and roadmap. If you need more active help and support, for example, if your browser is not allowing you to complete the labs, or you are experiencing a technical error, you can contact our support team.

support@sans-foundations.com

You will receive a notification that your ticket has been logged, and the support team will review your query and respond to you. In the interim, we suggest you keep on reviewing the material and working with modules in the platform - there is a lot to learn!

# Intro to Computer Hardware

## Learning Objectives

After completing this module, you should be capable of:

- Identifying computer hardware components.
- Replacing damaged components in a desktop computer.
- Assembling a computer from individual parts.
- Understanding what each component does, and how they work together as a whole.

## Module Content

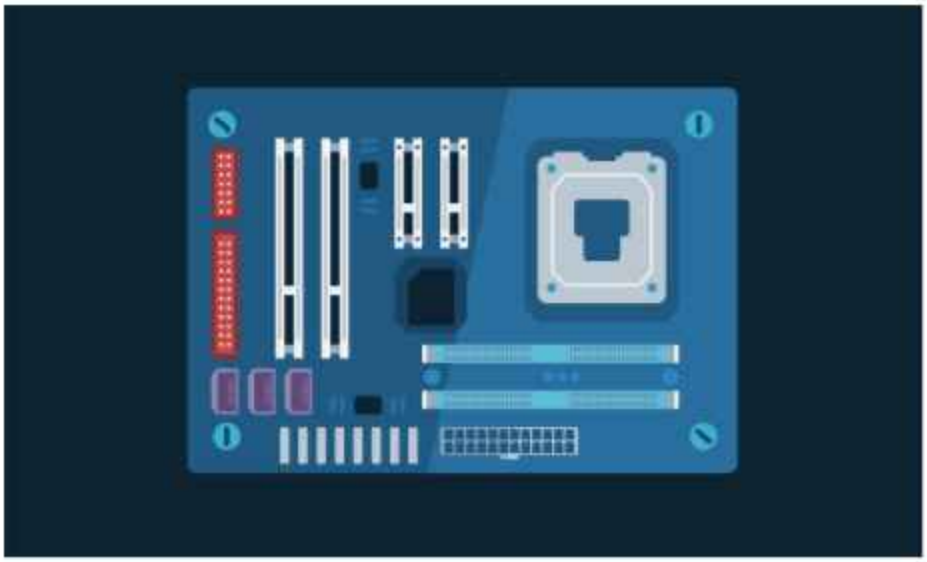
This module will provide a brief overview of each hardware component that goes into a functioning computer. The description of each element will include information on the role of the component in the operation of the computer, along with technical details that could help a student assemble a computer from parts.

We will be covering the following components:

- Motherboard
- Processor
- Random Access Memory (RAM)
- Storage
- Graphics Processing Unit (GPU)
- Input Devices
- Output Devices
- Power Supply Unit (PSU)
- Connectors
- Heat Sink
- Thermal Paste

## Motherboard

The motherboard is the central piece to which all other components connect.



This image shows the front of a typical motherboard. We can see there are clear spaces for other components to slot into the board.

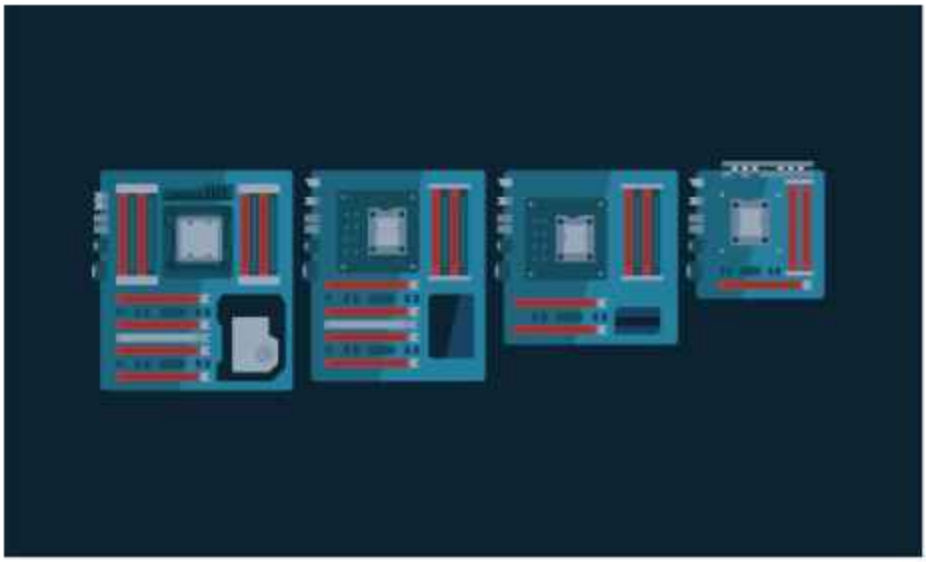
### Sizes

Motherboards typically come in different, but standard, sizes. PC cases usually come with holes pre-drilled for motherboards to attach to with screws.

The various sizes of a motherboard are:

- E-ATX: The largest size, known as 'extended ATX'.
- ATX: The 'standard' size.
- micro-ATX: Smaller than ATX.
- mini-ITX: Even smaller still.

Most desktop computers use an ATX or E-ATX motherboard, with micro-ATX and below usually used in laptops or other small form-factor applications.



Here we have an image showing the differences between motherboard sizes. If you are building your first computer, make sure you select a motherboard size that is compatible with your case!

**Sockets**

Motherboards are often classified based on the 'socket' type with which they are compatible. The socket is just a name given to the kind of connector that the CPU processor has. The CPU processor must have a socket that is compatible with the motherboard, otherwise it simply will not fit!

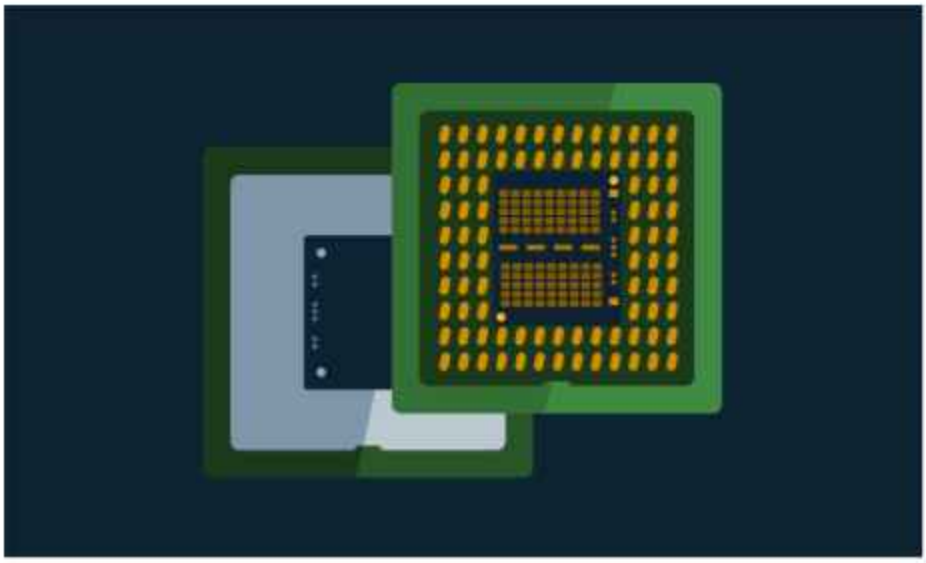
For example, say you have a processor with a socket type of LGA 2011. If you want to connect it to a motherboard, that motherboard should also have a socket type of LGA 2011.

**Buses**

In the images of motherboards above, you can see the locations in which hardware will eventually slot. These components naturally need to be able to communicate, and the motherboard is the component which enables this. The motherboard does this by using buses. Think of them as physical connections between the components that connect to the motherboard. The buses are responsible for moving data between components, similar to how buses are used to move people between locations in a city.

## Processor

The processor, or 'Central Processing Unit (CPU)', is the 'brain' of the computer. The processor is responsible for executing the instructions contained in computer programs.



This image shows a modern processor's front and back view. The front is the piece with the silver 'lid' on it, while the back view is the section with the gold connectors. The connectors are laid out according to the 'socket' specification. The socket configuration of a processor allows the processor to be used in motherboards that have a compatible socket type.

### Cores

Each processor contains at least one processing 'core'. The core is responsible for executing instructions. Modern processors often contain more than one core, allowing them to perform multiple tasks simultaneously. Older processors only had a single core, meaning they could only execute a single instruction at a time.

Even in an older processor with only a single core, computers seem to give us the ability to perform many tasks simultaneously. For example, you might be writing a document in Microsoft Word, and have your email open in the background. While you are typing away, you might see a new email come into your inbox. If a computer can only perform one task at a time, how did you receive the email while you were typing? To solve this mystery, we will first need to learn about clock speed.

### Clock Speed

A computer program is broken up into a series of small instructions, which a processor can understand and execute very quickly. In fact, we often measure a processor's speed by how many instructions it can execute in one second. A processor that can execute one

instruction per second has a clock speed of 1 Hz (hertz). It is not unusual to find modern processors which run at a clock speed of 4 GHz (Gigahertz). A computer with a clock speed of 4 GHz can execute 4 billion instructions per second.

**Context Switching**

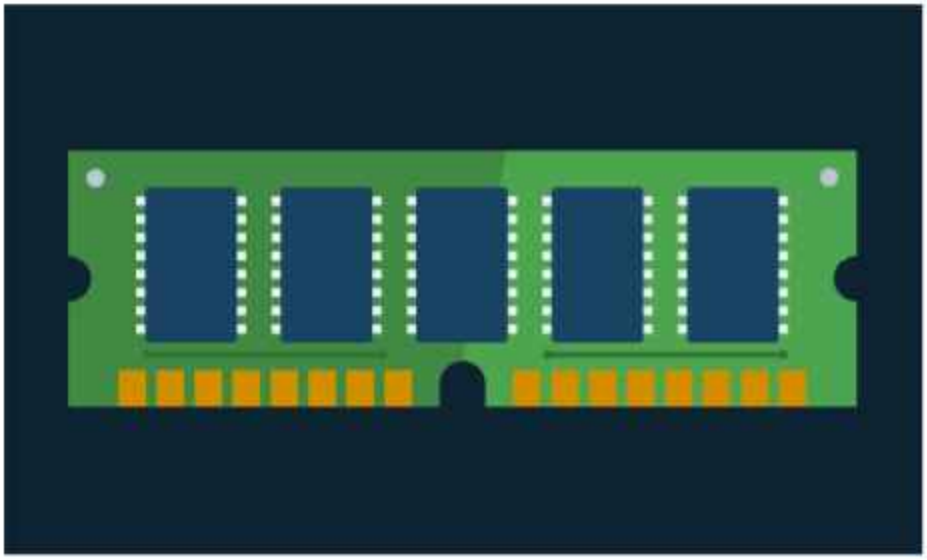
A single core processor does not perform multiple tasks simultaneously; it merely feels that way to the user. This capability is achieved using a process called 'context switching', whereby the processor shares processor time between multiple applications, swapping back and forth between them at a speed that the human mind cannot discern.

## RAM

RAM is short for Random Access Memory. RAM can also be referred to as simply 'memory'. RAM is often confused with 'storage', which is something else entirely.

Erm, not that ram...

This is RAM:



Here we have a 'stick' of RAM. The gold connector at the bottom of the RAM 'stick' slots into the RAM slots on the motherboard. The 'stick' in the image above is 8GB of RAM. Think of RAM as the number of things you can have open on your computer at once. So with 8GB of RAM, you can have 8GB of software running all at once. Part of that will be taken up with the operating system (e.g. Windows), but the rest will be for whichever programs, images etc. you have open at once.

That isn't to say there is a direct correlation between storage and RAM. If you purchase and download a video game that is 40GB in size, you don't need 40GB of RAM to run the game. A lot of that storage will be resources such as models and textures which are only loaded and unloaded as they are needed. They won't all be loaded into RAM at once.

One of the most common ways to speed up a slow computer is to upgrade the RAM, although you should make sure that the lack of RAM is the bottleneck first. If you already have plenty of RAM, upgrading it won't make much of a difference. Typically, performing a RAM upgrade is easy. You just need to pluck out the existing sticks of RAM and replace them with new ones with a higher capacity. Of course, you must make sure that the RAM you buy as a replacement is compatible with your motherboard.

### Speed

The benefit of RAM is its speed. It is much faster to read data stored in RAM than it is to read data stored on a hard disk, for example. This is why data is often read from storage and then kept in RAM while the processor is actively using it. The processor will then refer to that data from its position in memory, rather than on disk.

### Volatility

Since RAM is so fast, why don't we use it for storing everything? The simple answer is that RAM is**volatile**. This means that the data stored in RAM exists only temporarily. As soon as the computer is switched off, any data stored in RAM will begin to degrade. Can you imagine storing your documents on your computer, but having all those files deleted after the computer restarts?

### Compatibility

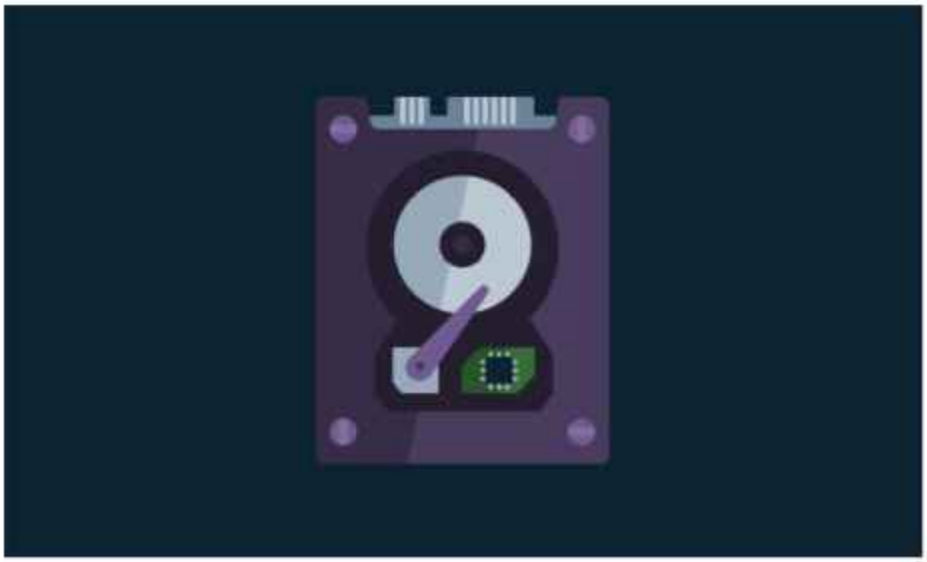
There are two factors to pay attention to with RAM. The first is the type of RAM. For example, DDR4 (double data rate fourth generation) RAM is relatively new, and only some motherboards support it. DDR3 RAM was first introduced in 2007 and most motherboards will happily support it.

The other factor to pay attention to is the speed. Obviously, faster RAM is better. The speed of RAM is measured in MHz, so, for example, DDR3-2133 RAM is DDR3 RAM with a maximum speed of 2133 MHz.

Many motherboards will explicitly state what type of RAM they accept. For instance, a motherboard manufacturer may say that the motherboard accepts RAM up to DDR3-2133, meaning any DDR3 RAM module with a frequency of 2133 MHz or less.

## Types of Storage

Storage is a component that provides data storage. Typically this is a hard drive or SSD (solid state drive). Again, this is often confused with RAM or memory and the terms are used interchangeably in many technical contexts - but the difference in speed and persistence of data are vast!



Here we have a 4 TeraByte 3.5 inch hard drive. At this size, this is a hard drive for use in desktop computers.

### Size

The physical size of the hard drive is important. 3.5 inch hard drives, as in the picture above, are frequently used in desktop computers, however they are rarely seen in laptops due to the amount of space they take up.

Laptops more commonly use the smaller form-factor 2.5 inch hard drives for storage instead. These tend to have a lower maximum capacity than their desktop counterparts. Make sure you pick the right size for your use case if you are purchasing parts!

### Capacity

Aside from the physical size of the drive, each drive has a maximum capacity, the largest amount of data that can be stored on the drive itself. Drives usually have a maximum capacity that is slightly lower than their advertised value. For example, a 4 TB drive cannot actually store 4 TB of data. Usually, the capacity will be about 3.8 TB, so keep this in mind.

### Mechanical vs Solid State

For a long time, the only type of hard drives we had was mechanical drives. These drives contain moving parts and therefore they can be quite unreliable over longer periods of time, or if they are jolted while in use. They are also comparatively slower than their solid state counterparts.

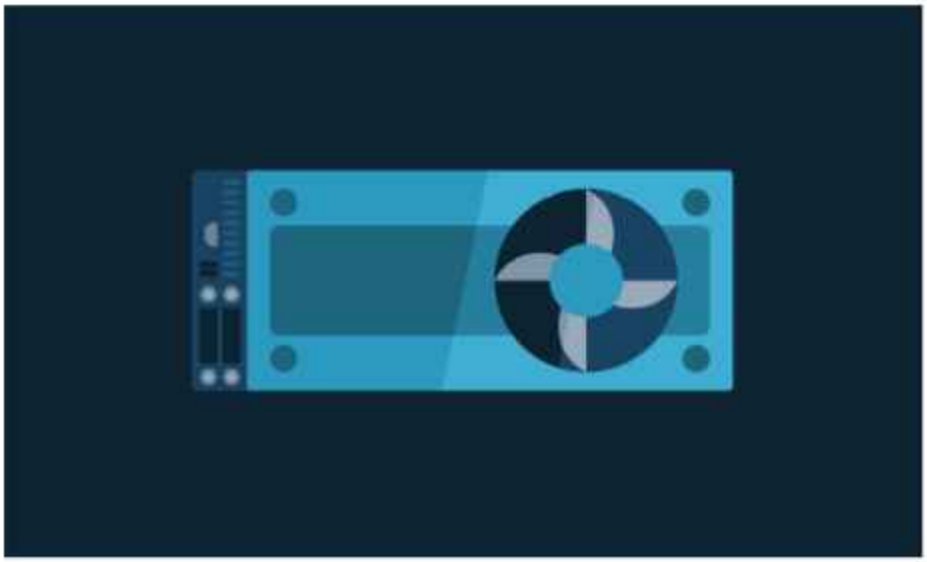
Solid state drives contain no moving parts and therefore they are both more reliable and faster than mechanical drives. However, they are still significantly more expensive than their mechanical counterparts.

Solid state drives include firmware and services to enable their fast operation with a modern computer, but as we will talk about later this can present interesting opportunities and challenges with forensic recovery of data, or secure destruction of data.

## GPU

The GPU, or Graphics Processing Unit, is an optional component. Not every computer has one, because a GPU is just another kind of processor that excels at number crunching. In computers without a dedicated GPU, the CPU performs the same function, albeit less efficiently than a computer with a dedicated GPU.

Where a computer has a GPU, the processor offioads the calculations necessary for displaying graphics on a monitor to the GPU, providing a performance benefit to the processor. The GPU usually resides on a card called a 'graphics card'. This card contains not just the GPU, but also its own dedicated RAM where computer graphics are stored for use by the GPU.



Here we have a relatively modern graphics card. The bulk of the card is taken up by a heat sink and fan, used for redirecting heat away from the sensitive components.

### Number Crunching

Don't be fooled by the name 'Graphics Processing Unit'; the GPU excels in all tasks that involve mathematical calculation. Often, GPUs are used in other mathematically demanding tasks, such as 3D rendering, or encryption. In cyber security, we often use graphics cards such as these to crack passwords.

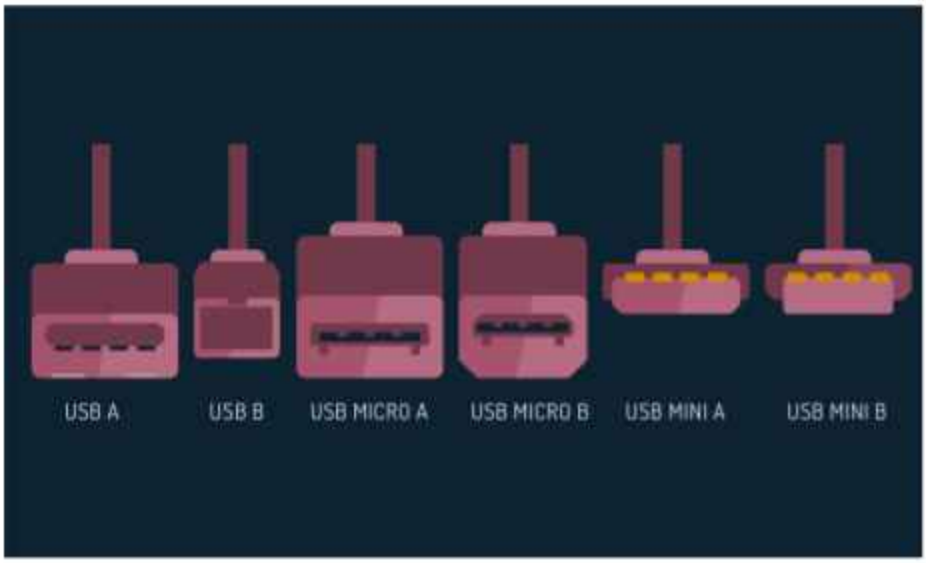
## Input Devices

Input devices are devices which, when connected to computers, can send data to them. These are used to control the computer. For example, a mouse is an input device, as is a keyboard. These devices are also known as 'Human Interface Devices', or HIDs, because they provide humans with a way to interface with the computer.

### USB

These days, the most common way of connecting an input device is over USB (Universal Serial Bus). Here is an image of a USB port:

Of course, not all USB connectors are the same. There are different types. You may be familiar with these other connection types:



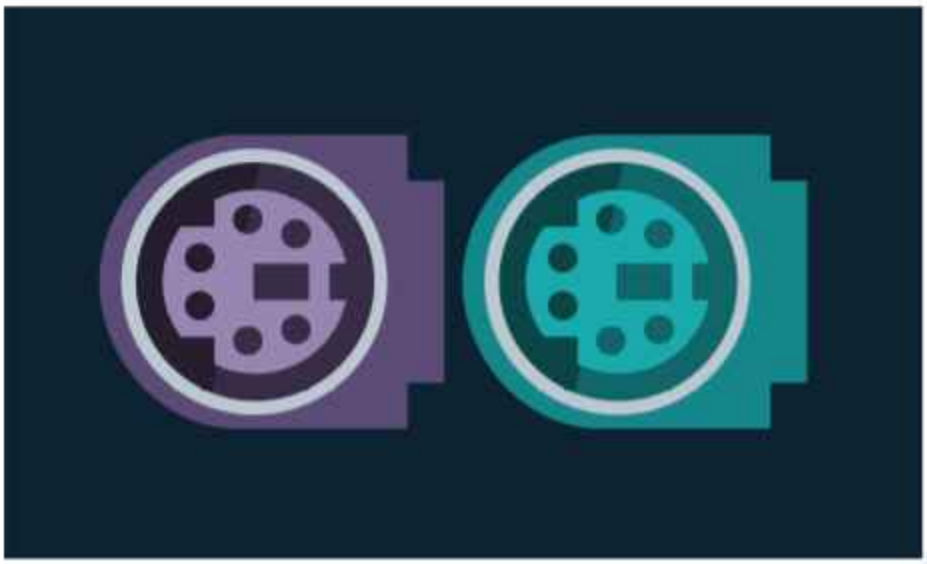
There is also a newer connection type known as 'Type-C', which looks like this:

As well as the differences in connectors, not all USB ports are the same either. There have been several upgrades to the USB standard in the past.

- **USB 1 - Transfer Speed:** 1.5 Mbit/s
- **USB 2 - Transfer Speed:** 480 Mbit/s
- **USB 3 - Transfer Speed:** 4.8 Gbit/s (You can usually tell if a port is USB 3 compatible because the port will be coloured blue)
- **USB 3.1 - Transfer Speed:** 10 Gbit/s (You can usually tell if a port is USB 3.1 compatible because the port will be coloured teal)

**PS/2**

Of course, before USB, there was the PS/2 port, which many older peripherals support.



## Output Devices

Output devices are devices that accept data from the computer. An output device could be a monitor, printer, or any such device. A monitor accepts data from the computer and displays an image. A printer accepts data from the computer and prints a document. We classify any other device that functions similarly as an output device.

Output devices such as printers are often connected over USB, whereas monitors might use VGA, or HDMI. More and more systems are tending towards using USB-C for 'everything' given the powerful range of capabilities of this port - from power to display or user input.

### VGA

VGA is the oldest standard for A/V output that we are going to cover. The port looks like this:

On either side of the port, there are two slots into which thumb screws on the connector connect. These need to be tightened to provide stability to the connection.

### DVI

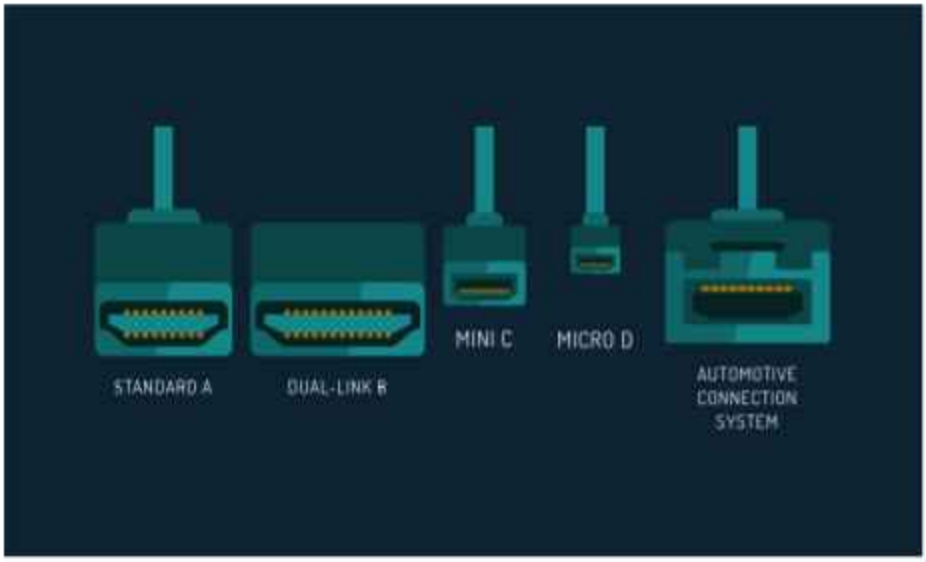
DVI is more modern than VGA. However, it is still old by modern standards. Here is what a DVI port looks like:

Once again, note the area for two thumb screws which need to be tightened after the connection is plugged in.

**HDMI**

The HDMI connection is more modern and more widely used than DVI or VGA these days. It can transmit not only video but audio also.

Here are some examples of HDMI connectors:




**Display Port**

Even more modern than HDMI is a display port.

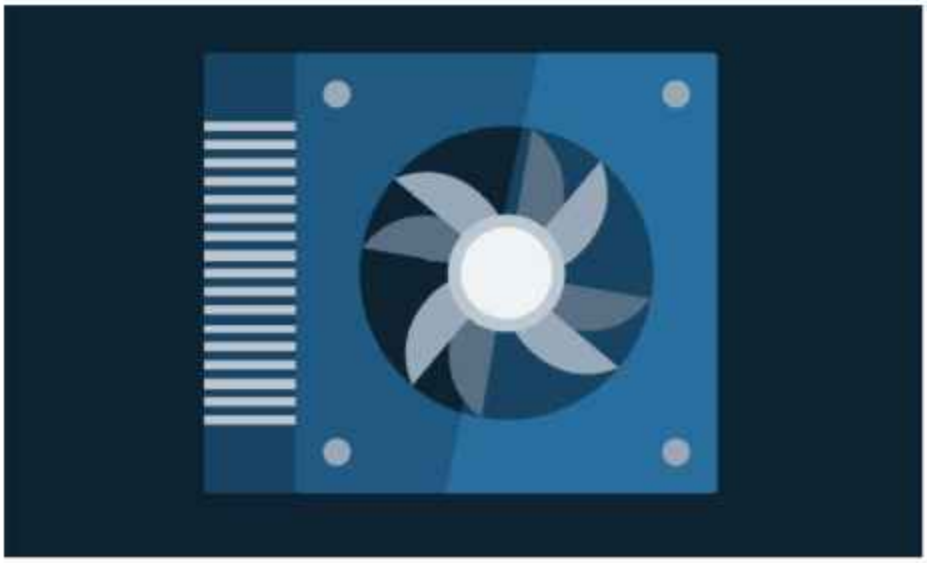
**USB-C**

A powerful connector with a wide range of capabilities. Some modern monitors support video output over USB-C, whilst also enabling hub capabilities for you to chain together large numbers of devices. Power can also be transferred, enabling 'one port' to rule them all! This is undeniably an attractive proposition to avoid the complexities of different ports, but this is far from universally used as yet.

## Heat Sinks

One of the primary considerations when building a computer has to be heat build-up. Computers can generate a lot of heat, and the components are often placed into a small space such as a case. They need to have a way to deal with the build-up of heat efficiently, otherwise the components may overheat and be damaged.

Many cases deal with this problem by having room for a lot of case fans, but fans alone are not enough. Heat sinks are used to move heat away from sensitive components.



Here we have an image of a heat sink attached to a processor. The heat sink is the metal block with many fins. A heat sink is made of a thermally conductive material. The purpose of the heat sink is to move heat away from the critical component, in this case the processor. The fins are designed to create more surface area for the air to cool it more efficiently.

To attach the heat sink to a component, we use thermal paste. Thermal paste is spread onto one of the components before the heat sink is attached. Thermal paste helps to conduct heat from one surface to another by filling in minute imperfections in the surfaces, which can trap air (which is a good insulator).

The fan attached to the top of the heat sink blows air away from the heat sink, therefore moving the heat from the heat sink out through the vents on the case.

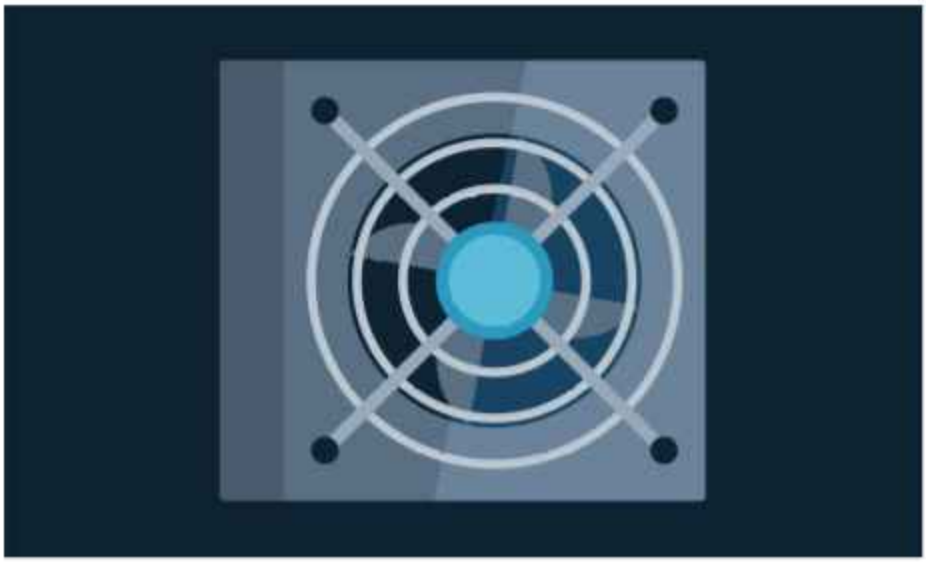
The processor isn't always the only item cooled with a heat sink. The graphics card, if the computer has one, often has a heat sink and fan built into it as standard. Many motherboards incorporate small heat sinks into the motherboard design, although these are usually too small to be cooled by dedicated fans. The case fans usually take care of cooling the heat sinks on the motherboard.

## Power Supply

The final component we will cover is the power supply or PSU (Power Supply Unit). The power supply is responsible for taking power from either the mains power supply or a battery (in the case of laptops) and converting then delivering it to the computer components.

The power supply usually connects to the motherboard, the graphics card(s), the hard drive(s) and the fans. All other components, such as the processor, are usually powered by the motherboard.

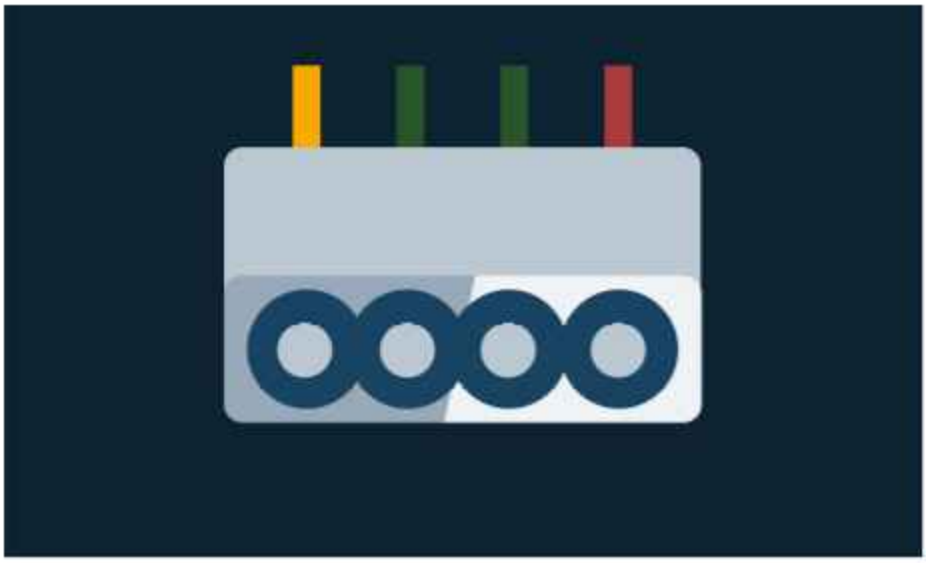
Each power supply will be rated to provide a certain amount of power. You must make sure the power supply you are using will be enough to power all of the components in your computer, otherwise the computer may not turn on or may turn off randomly when the components require more power.



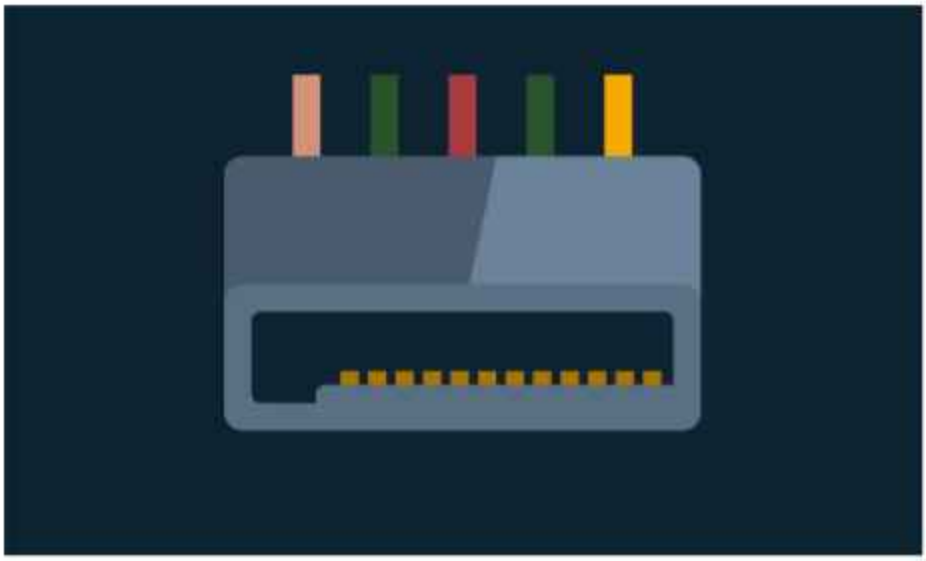
Here we have a power supply. Notice it has a built-in fan to cool it down during operation. Cables are running out of the unit, which will connect to the computer components.

### Connectors

We will briefly cover some of the connectors that power supplies use.



The above image shows a four pin peripheral connector. This type of connection is usually used for powering fans.



The above image shows a SATA (Serial ATA) power connector, which is used for powering hard drives.

The above image shows the main ATX connection that runs to the motherboard. Some motherboards require 20 pins, some require 24, and therefore most modern connectors provide a 20 pin connector with an optional extra four pins.

The above image shows a 12V power connector which runs to the motherboard. Again, some motherboards require four pins, and some require eight, so commonly you will find connections with four pins and an optional extra four.

The above image shows a PCI-E power connector, which usually runs to a graphics card. Again, some cards need six pins and some need eight, and therefore you will commonly find cables with an optional extra two pins.

## Walkthrough: Let's Build a Computer!

James Lyne walks through the full build process of a desktop PC, showcasing all the components we have looked at so far in this module.

# Data Storage and Representation

## Contents

This module will cover how data is stored on a computer. The topics covered are:

- Bits
- Bytes
- Alternate number bases
- Binary
- Hexadecimal
- ASCII
- Encoding
- Automatically Decoding Encoding
- File Headers

Note: As you can see, there will be maths in this module. DON'T PANIC! As long as you can add, subtract, multiply and divide, this will all be very basic.

## Bits

A bit is the smallest unit of data that can be stored on a computer system. There can only be two possible values stored in a single bit: a '1' or a '0'. This is known as a 'boolean' value. A 'boolean', or 'bool' for short, can only have two values. Think of it as a switch: a switch can have one of two positions: 'on' or 'off'.

All data in a computer system is made up of bits:

- 1 byte is 8 bits.
- 1 kilobyte (kB) is 1000 bytes.
- 1 megabyte (MB) is 1000 kilobytes.
- 1 gigabyte (GB) is 1000 megabytes.
- 1 terabyte (TB) is 1000 gigabytes.

Oh, and 4 bits (half a byte) is called a nibble...

Get it?

**1024?!**

Some of you may have read all this and gone, hey wait! Isn't 1 kilobyte, 1024 bytes?! Isn't 1 MB, 1024 kB?!

The answer is yes, and also no...

I hope that cleared things up for you!

Seriously though, the truth is, we started off using kB to denote 1024 bytes (The next few chapters will explain why it's 1024). The problem is that the prefix 'kilo' already means something. Scientifically, the prefix kilo means 1000 (10^3). This caused a lot of confusion

all around, so the IEC (International Electro-technical Commission) introduced KiB instead.

That's a kibibyte. There are also mebibytes, gibibytes and tebibytes and so on...

- 1 byte is 8 bits.
- 1 kibibyte (KiB) is 1024 bytes.
- 1 mebibyte (MiB) is 1024 kibibytes.
- 1 gibibyte (GiB) is 1024 mebibytes.
- 1 tebibyte (TiB) is 1024 gibibytes.

And so on...

However, this didn't really clear things up for most people. A lot of technical people went on thinking that 1 kB is 1024 bytes, without ever knowing that kibibytes existed.

To this day, it isn't always clear when someone writes kB if they mean 1000 bytes or 1024 bytes.

## Alternate Number Bases

To understand the format that computers use to store data, we first have to re-learn how to count. Counting is such a simple task that we learn from a very young age. It comes as naturally to most of us as breathing, and just like breathing we don't have to think about it very hard.

We usually count in 'base 10', also known as 'decimal' or 'denary'. The denary numbering system starts from 0 and runs through to 9. Those are all the digits that are available.

### Counting in Denary

If we want to count higher than 9, we need to start using more digits. The number '12' consists of two digits: a 1 and a 2. Written together, it's a twelve. Why is that?

Here we have a base 10 number, so think about it like this:

| *10000* | *1000* | *100* | *10* | *1* |
|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 2 |

There is one 10.

There are two 1s.

10 + 2 = 12.

Let's look at a larger number: 1337

| *10000* | *1000* | *100* | *10* | *1* |
|---|---|---|---|---|
| 0 | 1 | 3 | 3 | 7 |

There is one 1000.

There are three 100s.

There are three 10s.

There are seven 1s.

1000 + 300 + 30 + 7 = 1337

Makes sense, right? But you might be wondering about the table headings...

### Table Headings

So why are the table headings 10000, 1000, 100, 10, and 1? The easy answer is that they are powers of 10. See here:

(^ is the 'power of' symbol. For example, 4^3 is '4 to the power of 3' which is 4 x 4 x 4 or 64)

- 10^0 = 1 (Anything to the power of 0 is always 1, them's the rules!)
- 10^1 = 10 (Anything to the power of 1 is itself, them's also the rules!)
- 10^2 = 100 (10 x 10)
- 10^3 = 1000 (10 x 10 x 10)
- 10^4 = 10000 (10 x 10 x 10 x 10)

Now check out the table headings above again. See the pattern here? Remember, we call denary "base 10" because the base number is 10. The 'power of' is called the exponent. So 10^3 is base 10, with an exponent of 3.

**Alternate Number Bases**

You've guessed it; there are other number bases out there than just base 10. Base 10 is the one we all know and love (despite some people arguing we should switch to a base 12 numbering system because 12 has more factors).

Computers make use of base 2, which is also known as binary. Working with computers, we often convert binary numbers to base 16 (known as hexadecimal) because, frankly, binary numbers get long. Although they aren't much shorter in base 10, base 16 just makes them much more readable.

**Notation**

Now, this bit is really important. Since we're using all sorts of different number bases, we need a way to tell which number is written on what base. The number 10 is the perfect example.

The binary number 10 in denary is 2. The hexadecimal number 10 in denary is 16.

So if we just write 10, you'll probably assume I mean 10 in denary. But what if we actually meant 10 in binary?

This problem can be solved using notation. Specifically, you put the notation in front of the number to specify the number base it uses.

0d is for denary. 0b is for binary. 0x is for hexadecimal.

So 0d10 is 10 in denary. 0b10 is 10 in binary and 0x10 is 10 in hexadecimal.

## Binary

Counting in binary is very similar to counting in denary. First, let's calculate the headings:

- 2^0 = 1 (Remember, it's the rules: anything to the power of 0 is 1.)
- 2^1 = 2 (Again, anything to the power of 1 is itself!)
- 2^2 = 4 (2 x 2 = 4, no this isn't primary school...)
- 2^3 = 8 (2 x 2 x 2 = 8)
- 2^4 = 16 (2 x 2 x 2 x 2 = 16)
- 2^5 = 32 (I'm not writing it out anymore, it's starting to get very long!)
- 2^6 = 64
- 2^7 = 128
- 2^8 = 256

Remember a few sections ago, we mentioned that 1 KiB being 1024 bytes? Well, the why of it will probably make sense when I write it out like this:

- 2^10 = 1024

### Binary to Denary

Now that we've calculated the headings, let's write out some binary numbers in long form and turn them back into denary.

0b11001

| *512* | *256* | *128* | *64* | *32* | *16* | *8* | *4* | *2* | *1* |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |

So here we have:

1 x 16

1 x 8

1 x 1

So 16 + 8 + 1 = 25

That means 0b11001 = 0d25

Not too hard, right?

---

Here's another example:

0b101010101

| *512* | *256* | *128* | *64* | *32* | *16* | *8* | *4* | *2* | *1* |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

1x 256

1x 64

1x 16

1x 4

1x 1

So 256 + 64 + 16 + 4 + 1 = 341

That means 0b101010101 = 0d341

**Denary to Binary**

Okay, now to reverse the process, we'll take some numbers in denary and turn them into binary. This process is a little bit more complicated, but not by much.

0d666

| *512* | *256* | *128* | *64* | *32* | *16* | *8* | *4* | *2* | *1* |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

We'll start with a table above like so, with 0's in every column.

First of all, what is the largest table heading that goes into 666? It's 512, because larger than 512 is 1024, which is too big. So put a 1 in the 512 column:

| *512* | *256* | *128* | *64* | *32* | *16* | *8* | *4* | *2* | *1* |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Okay so we have 1 512, which means we need to subtract 512 from 666: 0d154

So what is the largest heading that goes into 154? It's 128, so put a 1 in the 128 column, and subtract 128 from 154:

| 512 | 256 | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

154 - 128 = 26

What is the largest heading that goes into 26? It's 16, so put a 1 in the 16 column and subtract 16 from 26:

| 512 | 256 | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |

26 - 16 = 10

What is the largest heading that goes into 10? It's 8, so put a 1 in the 8 column, and subtract 8 from 10.

| 512 | 256 | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |

10 - 8 = 2

What is the largest heading that goes into 2? It's 2, so put a 1 in the 2 column and then subtract 2 from 2:

| 512 | 256 | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 |

2 - 2 = 0, that means we're done with the conversion!

0d666 = 0b1010011010

---

Here is one more example, faster this time:

0d1023

| 512 | 256 | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

512 + 256 + 128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = 1023.

Notice how we can't make 1024 without adding an extra digit. With just 10 digits, we can only make 1023 or less.

## Hexadecimal

Hexadecimal is base 16; it's the same process as counting in denary and binary. But there is a unique problem to solve first. In denary, we have ten possible digits, 0 through to 9. In hexadecimal, we have 16 possible digits, but how do we represent them? Beyond 9, we use letters.

So counting in Hexadecimal goes like so:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

That's 16 possible digits, so there is nothing beyond F.

Let's calculate the table headings first:

- 16^0 = 1
- 16^1 = 16
- 16^2 = 256
- 16^3 = 4096
- 16^4 = 65536

Wow, that produced some massive numbers pretty quickly! That's the exact reason we use hexadecimal: it can represent large numbers in a smaller space.

### Hexadecimal to Denary

Let's convert hexadecimal to denary using the same process as we did for Binary.

0x2A

| ***65536*** | ***4096*** | ***256*** | ***16*** | ***1*** |
|---|---|---|---|---|
| 0 | 0 | 0 | 2 | A |

There are:

2 x 16s

10 x 1s (remember A is 10 in denary)

32 + 10 = 42

0x2A = 0d42

---

Here's another:

0xFADE

| *65536* | *4096* | *256* | *16* | *1* |
|---|---|---|---|---|
| 0 | F | A | D | E |

There are:

15 x 4096s (remember F is 15)

10 x 256s (remember A is 10)

13 x 16s (remember D is 13)

14 x 1s (remember E is 14)

61440 + 2560 + 208 + 14 = 64222

**Denary to Hexadecimal**

Now let's do some conversion between denary and hexadecimal. The process is the same as for binary:

0d1337

| *65536* | *4096* | *256* | *16* | *1* |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |

Okay, so what is the largest heading that goes into 1337? It's 256. How many times does 256 go into 1337? 5 times. So put a 5 in the 256 column. Then subtract (5 * 256) from 1337:

| *65536* | *4096* | *256* | *16* | *1* |
|---|---|---|---|---|
| 0 | 0 | 5 | 0 | 0 |

1337 - (5 * 256) = 57

What is the largest heading that goes into 57? It's 16. How many times does it go into 57? 3 times. So put a 3 in the 16 column and then subtract (3 * 16) from 57:

| *65536* | *4096* | *256* | *16* | *1* |
|---|---|---|---|---|
| 0 | 0 | 5 | 3 | 0 |

57 - (3 * 16) = 9

What is the largest heading that goes into 9? It's 1. How many times does 1 go into 9? It's 9 times so put a 9 in the 1 column, then subtract (9 * 1) from 9:

| *65536* | *4096* | *256* | *16* | *1* |
|---|---|---|---|---|
| 0 | 0 | 5 | 3 | 9 |

9 - (9 * 1) = 0, so we're done with the conversion!

0d1337 = 0x539

---

Let's try another:

0d6878

| *65536* | *4096* | *256* | *16* | *1* |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |

What is the largest heading that goes into 6878? It's 4096. How many times does it go into it? 1:

| *65536* | *4096* | *256* | *16* | *1* |
|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 0 |

6878 - (1 * 4096) = 2782

What is the largest heading that goes into 2782? It's 256. How many times does it go into it? 10:

| *65536* | *4096* | *256* | *16* | *1* |
|---|---|---|---|---|
| 0 | 1 | A | 0 | 0 |

2782 - (10 * 256) = 222

What is the largest heading that goes into 222? It's 16. How many times does it go into it? 13:

| *65536* | *4096* | *256* | *16* | *1* |
|---|---|---|---|---|

| 0 | 1 | A | D | 0 |
|---|---|---|---|---|

222 - (13 * 16) = 14

What is the largest heading that goes into 14? It's 1. How many times does it go into it? 14:

| ***65536*** | ***4096*** | ***256*** | ***16*** | ***1*** |
|---|---|---|---|---|
| 0 | 1 | A | D | E |

14 - (14 * 1) = 0, so we're done with the conversion:

0d6878 = 0x1ADE

## ASCII

If you've been following along, you probably have a very good question right now. If all data on a computer is represented as binary, then isn't everything a number? But what about text? We all read text on computers every day, don't we?

The text that is drawn on the computer screen by a computer is read from storage as a binary number. The binary number correlates to a character, which is drawn on the screen. The exact correlations depend on the type of encoding that is used by the computer.

One of the most common types of character encoding is ASCII, which stands for 'American Standard Code for Information Interchange'. It is simply a table that maps binary to characters. So, for example, the capital letter 'A' is 0x41, or 0b01000001. When the computer wants to display some text to you, if it sees a 0x41, it will draw a capital A on the screen.

The original ASCII was 7 bits wide, so it supported only 127 characters ((2^7) - 1 = 127). These days we actually use extended ASCII which is 8 bits and therefore supports 255 characters.

### ASCII Table

Here is the regular ASCII table:

| DECIMAL | HEX | Char |
|---|---|---|
| 0 | 0 | [NULL] |
| 1 | 1 | [START OF HEADING ] |
| 2 | 2 | [START OF TCXT] |
| 3 | 3 | [END OF TCXT] |
| 4 | 4 | [END OF TRANSMISSION] |
| 5 | 5 | [ENQUIRY] |
| 6 | 6 | [ACKNOWLEDGE] |
| 7 | 7 | [BELL] |
| 8 | 8 | [BACK SPACE] |
| 9 | 9 | [HORIZONTAL TAB] |
| 10 | A | [LINE FEED] |
| 11 | B | [VERTICAL TAB] |
| 12 | C | [FORM FEED] |
| 13 | D | [CARRIAGE RETURN] |
| 14 | E | [SHIFT OUT] |
| 15 | F | [SHIFT IN] |
| 16 | 10 | [DATA LINK ESCAPE] |
| 17 | 11 | [DEVICE CONTROL 1] |
| 18 | 12 | [DEVICE CONTROL 2] |
| 19 | 13 | [DEVICE CONTROL 3] |
| 20 | 14 | [DEVICE CONTROL 4] |
| 21 | 15 | [NEGATIVE ACKNOWLEDGE] |
| 22 | 16 | [SYNCHRONOUS IDLE] |
| 23 | 17 | [END OF TRANS. BLOCK] |
| 24 | 18 | [CANCEL] |
| 25 | 19 | [END OF MEDIUM] |
| 26 | 1A | [SUBSTITUTE] |
| 27 | 1B | [ESCAPE] |
| 28 | 1C | [FILE SEPARATOR] |
| 29 | 1D | [GROUP SEPARATOR] |
| 30 | 1E | [RECORD SEPARATOR] |
| 31 | 1F | [UNIT SEPARATOR] |
| 32 | 20 | [space] |
| 33 | 21 | ! |
| 34 | 22 | " |
| 35 | 23 | # |
| 36 | 24 | $ |
| 37 | 25 | % |
| 38 | 26 | & |
| 39 | 27 | ' |
| 40 | 28 | ( |
| 41 | 29 | ) |
| 42 | 2A | * |
| 43 | 2B | + |
| 44 | 2C | , |
| 45 | 2D | - |
| 46 | 2E | . |
| 47 | 2F | / |
| 48 | 30 | 0 |
| 49 | 31 | 1 |
| 50 | 32 | 2 |
| 51 | 33 | 3 |
| 52 | 34 | 4 |
| 53 | 35 | 5 |
| 54 | 36 | 6 |
| 55 | 37 | 7 |
| 56 | 38 | 8 |
| 57 | 39 | 9 |
| 58 | 3A | : |
| 59 | 3B | ; |
| 60 | 3C | < |
| 61 | 3D | = |
| 62 | 3E | > |
| 63 | 3F | ? |

| DECIMAL | HEX | Char |
|---|---|---|
| 64 | 40 | @ |
| 65 | 41 | A |
| 66 | 42 | B |
| 67 | 43 | C |
| 68 | 44 | D |
| 69 | 45 | E |
| 70 | 46 | F |
| 71 | 47 | G |
| 72 | 48 | H |
| 73 | 49 | I |
| 74 | 4A | J |
| 75 | 4B | K |
| 76 | 4C | L |
| 77 | 4D | M |
| 78 | 4E | N |
| 79 | 4F | O |
| 80 | 50 | P |
| 81 | 51 | Q |
| 82 | 52 | R |
| 83 | 53 | S |
| 84 | 54 | T |
| 85 | 55 | U |
| 86 | 56 | V |
| 87 | 57 | W |
| 88 | 58 | X |
| 89 | 59 | Y |
| 90 | 5A | Z |
| 91 | 5B | [ |
| 92 | 5C | \ |
| 93 | 5D | ] |
| 94 | 5E | ^ |
| 95 | 5F | _ |
| 96 | 60 | ` |
| 97 | 61 | a |
| 98 | 62 | b |
| 99 | 63 | c |
| 100 | 64 | d |
| 101 | 65 | e |
| 102 | 66 | f |
| 103 | 67 | g |
| 104 | 68 | h |
| 105 | 69 | i |
| 106 | 6A | j |
| 107 | 6B | k |
| 108 | 6C | l |
| 109 | 6D | m |
| 110 | 6E | n |
| 111 | 6F | o |
| 112 | 70 | p |
| 113 | 71 | q |
| 114 | 72 | r |
| 115 | 73 | s |
| 116 | 74 | t |
| 117 | 75 | u |
| 118 | 76 | v |
| 119 | 77 | w |
| 120 | 78 | x |
| 121 | 79 | y |
| 122 | 7A | z |
| 123 | 7B | { |
| 124 | 7C | \| |
| 125 | 7D | } |
| 126 | 7E | ~ |
| 127 | 7F | [DEL] |

## Other Standards

There are other standards than ASCII, such as Unicode; however, ASCII is the simplest. Unicode supports a much larger character set so that languages which don't use the Latin alphabet can also be supported. Of course, Unicode encoding uses up more data because you can't support the Chinese, Latin, Arabic, and Russian alphabets all in a mere 255 spaces.

**Why do we care?**

ASCII and other such representation methods are incredibly frequently used within cyber security. If you are doing penetration testing in the future, you will likely be manipulating values like this to try and solicit unexpected or undesirable responses from computers.

## Encoding

Encoding is a mechanism enabling us to take data in one format, e.g. binary and then 'package' it in another, e.g. ASCII. This is useful when you have a medium that only allows a specific type of data transfer. For example, you might have text data that needs to be sent as a series of electrical 'blips' or 1s and 0s. That would be converting your data into binary. Equally, you might need to take a wider character space and send it through something that is happy with ASCII only. Base64 is a common mechanism to do that.

A great example of this is with images, which contain an array of numerical data points and structure. Base64 can make this a string of ASCII characters only using an encoding scheme.

Let's do a basic example with some simple information. We are going to run a command on our system that prints data. It is:

```
echo -e '\x254 Hey folks how goes'
```

This prints out a non-ASCII byte of data, and then an easy-to-read string. We can't send this as we are only allowed to use ASCII values. We will now encode it with base64.

```
echo -e '\x254 Hey folks how goes' | base64
```

This produces the result JTQgSGV5IGZvbGtzIGhvdyBnb2VzCg== . This is the base64 encoded data. We can send it over the network and no sign of that pesky non-ASCII character.

When we want to get it back, we can use base64 decode:

```
echo -e "JTQgSGV5IGZvbGtzIGhvdyBnb2VzCg==" | base64 -d
```

This returns back our weird data from before, "%4 Hey folks how goes". Encoding! Think of it as packaging.

There are lots of schemes for encoding out there, base64 is just one example. There is also base32, for example, which uses fewer characters and has a smaller range to express data. If you are following along on your own system, you may have to install additional software to perform the encoding.

## Automatically Decoding Encoding

The purpose of encoding is to enable us to fit data that we need to store or transmit within a set of constraints that are suitable to the storage or transmission medium. For example, we might have text that we need to fit into a protocol that supports a subset of characters, like base64. We might alternatively need to encode it and present it as a binary sequence or similar. Encoding is widely used for data storage and transfer, but does not mean strong security. Encoding is unfortunately sometimes mistaken for encryption, which has very different goals. That being said, encoding is often used with encryption to make the results transmissible.

In this walkthrough, we use a tool to automatically detect and reverse encoding to get back to the original data. Don't worry about how the tool is run for now, and the use of the command line; we will cover this in much greater depth later in the course. Of course, this process can be done by hand, but it is interesting that computers can use a clever search process to revert the data even when they are not provided with any prior details of how the data was encoded.

## File Headers

When most people want to know what kind of file something is, they look at the file extension. Naturally, a .txt file is a text file, and a .zip file is a zip archive, right? Not necessarily. What about files that don't have an extension, or what if someone changes the extension for an existing file?

The truth is, most file types have a header at the beginning of the file that tells us what kind of file it is. Even if you change the file extension, the file header will stay the same, so it's often more accurate than just looking at the file extension. File headers are often also called 'magic numbers' or 'magic bytes' or 'file signatures'.

A plain text file doesn't have a file header, but let's look at a zip file. A zip file can have the file headers:

50 4B 03 04 (for a normal zip archive)

50 4B 05 06 (for an empty zip archive)

50 4B 07 08 (for a spanned zip archive)

jpg files have the file headers:

FF D8 FF DB

or

FF D8 FF E0

or

FF D8 FF E1

Notice that these are all hex values. On disk, they are of course binary, but we use hexadecimal when writing binary would be too long. We write them in pairs, because every two hex characters equals one byte of data. Don't believe me? Let's work it out...

- **0b11111111** - Here we have 8 bits (1 byte), which works out to a maximum value of 255 in denary.
- **0xFF** - Here we have the largest value two hex characters can make; this is also 255 in denary.

## Logic and Data Manipulation

## Contents

In this module, we will be covering:

- Logic
- Truth Tables
- Logical AND
- Logical OR
- Logical NOT
- Logical NAND
- Logical NOR
- Logical XOR

## Boolean Logic

When we talk about logic in computer science, we usually mean Boolean logic. Remember, a boolean value is a value that can be either true or false. This kind of value is perfect for computers, which talk in bits: either a 1 or a 0. In other words, bits are Boolean values, like a switch they can be either on or off.

Logic is calculated in the logic unit of the processor through a series of logic gates. These gates are circuits which usually take two inputs and produce one output. Every input to the circuit is in one of two states, depending on the voltage flowing along it at the time: either 0 (low voltage, approx 0 volts) or 1 (high voltage, approx 5 volts). The circuit then produces an output, again, either 0 or 1 depending on the voltage. There are different types of logic circuits, which behave in different ways, but together they allow the processor to function.

Boolean logic is important because programs use it all the time. You may need to use it in your own programs, and programmers are often tripped up by logic since it isn't always intuitive. It's common to see bugs in computer programs which stem from incorrect usage of Boolean logic.

### The language of logic

There are a number of terms surrounding logic which you should know. The concepts are of course more important but knowing the language will allow you to communicate with your peers about the topic.

A boolean statement - one which can only be true or false is known as a **proposition**. For example: Tomorrow is Friday. This can either be true or false and is therefore a **proposition**.

### Truth Tables

When we look at Boolean logic, we usually use logic tables or truth tables to show all possible outcomes from inputs to that gate. Here is an example:

| *A* | *B* | *A AND B* |
|---|---|---|
| 0 | 0 | |
| 0 | 1 | |
| 1 | 0 | |
| 1 | 1 | |

We haven't filled in the answers in this case because we haven't explained an AND gate yet, but the concept is the same. The two inputs are represented by A and B. Together this table shows every possible combination of inputs. You'll be seeing a lot of these truth tables in this module, so get comfortable with them!

## Logical AND

The AND logic gate checks if the two inputs are both true. If they are both true, the output is 'true' and if one or more is false, the output is 'false'. Have a look at the truth table:

| *A* | *B* | *A AND B* |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

The two inputs**must** be true for the output to be true. In programming, you might see something like:

```
string1 = "demo"
string2 = "demo"
int1 = 42
int2 = 33

if (string1 == string2 && int1 == int2):
  dostuff()
```

This is a logical AND.`string1 == string2` is a comparison, and the result will be true if they are both the same, and false if they are not. In this case, it is true.

`int1 == int2` is another comparison and the result will be true if they are both the same and false if they are not. In this case, it is false.

The result we have is:

```
True AND False = False
```

Therefore, our`dostuff()` function is never called, because the condition has not been met.

### The language of logic

An AND is considered to be a**connective**. A connective is used to connect two or more **propositions**. The AND**connective** specifically is called a **conjunction**.

There are other forms of connectives which we will be covering in this module such as:

- OR
- NOT
- NAND
- NOR
- XOR

## Logical OR

The OR logic gate checks if at least one of the inputs is true. If it is, then the output is also true. Here is the truth table:

| *A* | *B* | *A OR B* |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

As long as one of the two inputs is true, then the result is true. If neither of the inputs is true, then the result is false. In programming, you might see something like:

```
string1 = "demo"
string2 = "demo"

int1 = 42
int2 = 33

if (string1 == string2 || int1 == int2):
  dostuff()
```

Let's break this down:

`string1` does equal `string2`, so the result is true.`int1` does not equal`int2`, so the result is false, so that works out as:

```
True OR False = True
```

In this case, our`dostuff()` function will run, because the condition has been met.

### The language of logic

As you already know an OR is a type of**connective** and the OR**connective** itself is known as a **disjunction**.

## Logical NOT

The logical NOT gate is a bit of an oddity because it only takes one input. It has the effect of inverting whatever input it receives, so true becomes false and false becomes true. Here is the truth table:

| *A* | *NOT A* |
|---|---|
| 0 | 1 |
| 1 | 0 |

You might see it in programming, like so:

```
string1 = "demo"
string2 = "demo"

if !(string1 == string2):
  dostuff()
```

In the example above, NOT is represented by the exclamation mark ('!'). Let's calculate it:

Does `string1` equal `string2`? Yes, so the result is true. The NOT causes that to become false, so `dostuff()` will not run because the condition hasn't been met.

You could also see not written like so:

```
if (string1 != string2):
```

In English, this is: "if string1 does NOT equal string2".

### The language of logic

As you already know a NOT is a type of **connective** and the NOT **connective** itself is known as a **negation**.

## Logical NAND

The NAND gate is just an AND circuit followed by a NOT circuit, so this one should be very easy. Just invert the result of an AND gate, and you'll have your answer. Take a look at the truth table:

| *A* | *B* | *A NAND B* |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Compare that to the AND truth table:

| *A* | *B* | *A AND B* |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

It's just the AND truth table inverted.

In programming, you might see:

```
string1 = "demo"
string2 = "demo"

int1 = 42
int2 = 33

if !(string1 == string2 && int1 == int2):
  dostuff()
```

Let's break it down:

Does `string1` equal `string2`? Yes, so it's true. Does `int1` equal `int2`? No, so it's false. So far we have:

```
!(True AND False)
```

True AND false is false, so we now have:

```
!(False) = True
```

So our `dostuff()` function is going to run because the condition has been met.

**The language of logic**

As you already know a NAND is a type of **connective** and the NAND **connective** itself is known as an **alternative denial**.

## Logical NOR

The NOR logic gate is just the OR circuit followed by a NOT circuit, similar to how NAND works. You just take the results of an OR truth table and invert the output. Here is the truth table:

| *A* | *B* | *A NOR B* |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

Compare that with an OR truth table:

| *A* | *B* | *A OR B* |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

You can see this is just the same table with the output inverted. In programming you might see it like this:

```
string1 = "demo"
string2 = "demo"

int1 = 42
int2 = 33

if !(string1 == string2 || int1 == int2):
  dostuff()
```

Let's break it down:

Does `string1` equal `string2`? Yes, so it is true. Does `int1` equal `int2`? No, so it is false. We now have:

```
!(True OR False)
```

True OR false is true (remember, with OR just one of the inputs being true makes it true). We now have:

```
!(True) = False
```

So our `dostuff()` function is not going to run because the condition has not been met.

**The language of logic**

As you already know a NOR is a type o**connective** and the NOR**connective** itself is known as a **joint denial**.

## Logical XOR

The XOR gate stands for Exclusive-OR. This gate checks if only one side or the other is true but not both. If only one input is true, then the output is true. Here is the truth table:

| *A* | *B* | *A XOR B* |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Notice when A and B are both true, then the output is false? This is the difference between XOR and OR. Remember, XOR is exclusive, it doesn't like both inputs being true. Unlike the others, you won't see this one a lot in conditionals, but you will see it a LOT in cryptography.

XOR is really cool, in that:

A XOR B = C
A XOR C = B
B XOR C = A

Take a look:

| **1** | **1** | **0** | **0** |
|---|---|---|---|
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |

So 1100 XOR 0101 = 1001

If you treat 1100 as your data, and 0101 as your encryption key, then 1001 is your encrypted data.

If you have your encrypted data, you can XOR it with your key to get your decrypted data back:

| **1** | **0** | **0** | **1** |
|---|---|---|---|
| 0 | 1 | 0 | 1 |

| | | | |
|---|---|---|---|
| 1 | 1 | 0 | 0 |

1001 XOR 0101 = 1100

1100 was our initial piece of data. This is a very basic form of encryption on its own. It isn't a very strong form of encryption, but it is used in a lot of cryptographic algorithms as part of the process, including AES which is the current industry standard.

**The language of logic**

As you already know a XOR is a type of **connective** and the XOR **connective** itself is known as an **exclusive disjunction**.

## Encryption with XOR

XOR is a crucial concept in encrypting data and one of the most simple forms of encryption/decryption. It is used in many real world implementations, though often as a stage of the process. With XOR you iterate through text using XOR bit by bit to produce the new result, this is the encrypted text. For example in python3:

First we define a function to XOR two strings. We will cover the code much later in the class, just trust it works for now!

```
def sxor(s1,s2):
  return ''.join(chr(ord(a) ^ ord(b)) for a,b in zip(s1,s2))
```

Now we will use this to populate plaintext, key and then encrypt.

```
plaintext="Hey how goes?"
key="adhnawdagjswb" # Note the key is equal in length to the plaintext.
ciphertext=sxor(plaintext,key)
```

If we print each of these as binary values:

```
bin(int.from_bytes(plaintext.encode(),'big'))
bin(int.from_bytes(key.encode(),'big'))
bin(int.from_bytes(ciphertext.encode(),'big'))
```

Producing (in order):

```
0b1001000011001010111100100100000011010000011011111
1100100000011001110110111101100101011100110011111
```

```
0b1100001011001000110100001101110011000010111011101
1100100011000010110011101101010011100110111011101100010
```

```
0b0101001000000010001000101001110000010010001
1000000100110100000100000000000001010001011000000100011101
```

If you take the very first values after the 0b (denoting binary) they are 0 from plaintext and 1 from the key. XOR 0 and 1 produces a 1 as the values are not identical, so as expected the third sequence shows a 1 straight after the 0b. Continue down the sequence and you will find they are simply XOR transformations.

**Note:** If you try running the Python above the ciphertext may give you `0b10100100000001000100010100111000001001000110000010011010000010000000000000010100010110000001000101110 1` instead. This is actually one bit shorter than it should be because a preceding 0 is optional in Python and not displayed. If you didn't try it yourself don't worry about it for now.

The resulting data?

```
>>> ciphertext
'\x01\x11N\t\x18\x13A\x00\x05\x16\x04]'
```

That data looks pretty darn unreadable, and it is!

You can XOR plaintext with the key and you get the ciphertext. This is encryption. You can XOR the ciphertext with the key and get the plaintext. This is decryption. You can XOR the plaintext with the ciphertext and get the key, which might be re-used elsewhere and enable further decryption. This is key recovery.

## The Language of Logic

There are still a few important terms that you don't know so in this section we will go over them.

### Tautology

A tautology is an assertion or formula which is always true no matter the inputs. If you craft a logical statement that no matter the inputs is always true then this is a tautology.

Another way of saying this would be if you have a logical statement where the truth table for it always produces only true results then this is a tautology.

### Contradiction

A contradiction is the opposite of a tautology. This is where all the results of a logical statement are always false no matter the inputs. If your truth table only produces false results then you have crafted a contradiction.

### Contingency

A contingency is essentially anything that isn't a tautology or contradiction. In other words a logical statement where the outcomes could be true or fals**eontingent** on the inputs.

# Storing Data and Files

## Contents

In this module you will learn:

- File Systems
- How Files are Stored
- FAT32 & ExFAT
- NTFS
- EXT3 & EXT4
- HFS & AFS

## File Systems

Let's now consider how files are stored on storage media, such as a hard drive or SSD (solid state drive). Each drive must be formatted with a file system. The file system determines how files are stored on the device, and what features the file system offers. The operating system usually has a list of file systems which are supported; not all file systems will be supported by every operating system.

Think of the file system as something like a protocol for accessing files on and saving files to the physical storage media. The operating system needs to understand the protocol to use a particular file system, therefore if the operating system doesn't understand the protocol, it cannot use drives that are formatted using that file system.

Every storage device is broken down into a series of clusters. The size of each cluster is determined by the file system. A cluster is the smallest section of the disk that can be used to store a file. So if you have a file system with a cluster size of 32kb, and save a file that is 64kb in size, then that file will be spread across two clusters.

Likewise, if you have a file of size 1kb and you save it to the disk, it will take up one whole cluster and actually use up 32kb of space on the disk. That's because two files can't use the same cluster, so the minimum file size on a file system with a cluster size of 32kb is 32kb. If you have a smaller file than 32kb, the remaining space in the cluster will be wasted. This wasted space is known as 'slack space'.

Similar to how memory is addressed in RAM, each cluster on a disk has an address.

Every file system stores at least two pieces of information per file. The first is the data in the file, in other words, the contents of the file. The next is metadata (data which describes other data). This information is usually at least the name of the file and the address where the contents of the file can be found. Some file systems will store more metadata like the user who created the file, and the last modified time and so on.

The metadata is stored in an index which provides a list of files and the locations where they can be found on the disk. The exact method for doing this varies from file system to file system, but the concept remains the same.

If a file is deleted, then the index entry is removed, but the content of the file isn't removed from the disk. Instead, that cluster is marked as overwritable, meaning the contents of a new file could overwrite the data there. The reason for this is simply efficiency. It is pointless to overwrite the data of a deleted file with 0s and then allow the contents of a new file to overwrite those 0s. Instead, we just mark that cluster as overwritable and allow a new file to overwrite the contents of the previous file. That's one overwrite procedure instead of two, and the result is the same.

This is the reason why you can sometimes recover deleted files from a hard drive; the contents of the file remains even if the metadata has been removed. Of course, there is a chance that deleted file's contents had been overwritten by a new file and the old

deleted file will not be recoverable. When we talk about securely deleting files, the data is written over the contents of the file when the file is deleted, instead of just marking the cluster as overwritable.

**Multiple Clusters**

So the real question is, how are files tracked over multiple clusters? If you have a file of size 64kb on a file system with a 32kb cluster size, then that file will take up two clusters on the disk.

- If the first cluster the file is stored in has a cluster immediately after it that is free, then the rest of the file will be placed there.
- If there is no cluster free immediately after the first cluster, then the rest of the file will be put into a different cluster, and the address of the next cluster will be added to the end of the first cluster.
- Some file systems will use a file allocation table to map each cluster, so the first cluster will point to the table entry, which contains the addresses of the next cluster. And that cluster will also have an entry in the table, which points to the next cluster and so on until the file has been read.

## FAT32 & exFAT

### FAT32

The FAT32 file system was introduced with Windows 95. It uses a File Allocation Table to map each cluster, which is where the name FAT comes from. The FAT32 file system doesn't support files larger than 4 GB, which seemed a huge amount back in 1995, but these days is hardly anything. It doesn't support file permissions because it doesn't store metadata such as who created a file; therefore it was primarily used in USB drives, which could be connected to any computer. Think about it, if you use a file system which supports permissions, you can't guarantee the computer you connect the device to will have the same user account on it, so a lack of permissions support is actually a bonus for a file system designed for USB drives and other removable media.

The FAT32 file system is not as common in USB drives as it used to be just a few years ago because of the new kid on the block, exFAT.

### exFAT

The exFAT file system is a file system designed for USB drives and other removable media, so it doesn't support permissions. It was introduced in 2006, but it took a few years to gain enough traction for USB drive manufacturers to start loading it by default. It is based on FAT32, however it has been completely modernised. The file size limit is so large that it effectively has no maximum file size. It supports Windows, Mac and Linux, however be careful using it on very old operating systems such as Windows 95. (If you still have a Windows 95 computer then you have bigger issues.)

## NTFS

The NTFS (New Technology File System) is the file system used by modern versions of Windows. It is an advanced file system with many features, including permissions support (what usernames can access a file), encryption support and shadow copies (effectively backups of files). There is also a file size limit that is so large, it is effectively meaningless. The NTFS file system is also more reliable than older file systems; to a limited extent, it is capable of healing from data corruption.

The downside is that there is limited support for the NTFS file system amongst non-Windows operating systems. For example, if you connect an NTFS formatted drive to a Mac computer, you'll find that you can read files on the drive but not write to the drive.

## EXT3 & EXT4

### EXT3

The EXT3 (extended file system 3) is an older file system often used in Linux. It was introduced in 2001 and supports permissions and encryption, although no shadow copy (which is strictly a Microsoft thing). The EXT3 file system features a maximum file size of 2TB. It is a 'journaling' file system, which means that changes to the disk are tracked in a separate part of the file system known as the 'journal'. This can help to recover the drive in the event of a disk corruption that might result from a sudden shutdown or jolt.

### EXT4

The EXT4 (extended file system 4) is the modern file system that is used in Linux. It was introduced in 2008 and supports permissions and encryption, although, again, no shadow copy. The EXT4 file system has a maximum file size so large that, in practical terms, there is no maximum file size. Other than that, the EXT4 file system allows you to optionally turn off the journal, and features a faster disk check process. Generally speaking, this is the file system you want to be using on Linux.

## HFS+ & APFS

### HFS+

The HFS+ (Hierarchical File System Plus) was, until very recently (as of the time of writing), the file system that Apple used in Mac OS X. It is a proprietary file system that Apple produced, only compatible with the Mac OS operating system. It supports files so large that there are effectively no file size limitations, and it also has a journal similar to EXT3 and EXT4. As with most modern file systems, it supports permissions and encryption, amongst other features.

### APFS

Recently, Apple has introduced APFS: the Apple File System, which is another proprietary file system. The APFS debuts on Mac OS High Sierra. It supports permissions and encryption, and duplicate files can be stored without using additional space, with changes to one copy of a file being saved as a delta (the difference between the old file and the new file) to lower space requirements. This is a modern and robust file system designed with resilience and security in mind. It is now the default on most Mac systems.

# Cloud Computing

## Cloud Computing

In this module we will review the concept of cloud computing and the various models that are growing increasingly popular for hosting applications and services. This is very much a default for most businesses going forwards, so we want to make sure we are familiar with it.

All your skills in the Linux section of the course will be invaluable when it comes to driving cloud platforms.

## SaaS, IaaS and PaaS

In this module we will review the models of service that cloud providers make available to use today. We will review them one by one, and the major definitions and benefits of each of them. Some providers are even in the business of offering multiples of these models!

### SaaS - Software as a Service

SaaS is fundamentally software that is available via a third party over the internet. Take, for example, a business reporting application that you use to report on the metrics of your business moment to moment, day to day. This application could be installed locally at your office, or it can be hosted by the provider of the software so that you do not need to maintain the server, OS and application - you just configure and use it. The major benefit of this is that you can focus on using the application and benefiting from the 'service' without getting into having to operationalise it. This usually comes at a cost though. Whilst you get less hassle and ownership, two things are typically true: * SaaS is often more expensive in the licence/service sense, as you are handing off responsibility for lots of tasks. That doesn't mean the 'true cost' is not better, but this should be evaluated. * You need to fit the shape and size of the service the provider gives you - you can't go and modify as much as if you owned the installation, so need to be confident you fit what they provide.

Ultimately, think of SaaS as renting a finished house. You move in and use it, but you are welcome to move some of the furniture around or customise it within reason. You don't own it though, so you can't do whatever you want!

A few examples of SaaS are Slack, ZenDesk or SalesForce.

### IaaS - Infrastructure as a Service

IaaS gives you cloud-based services, typically delivered with a pay-as-you-go and based-on-what-you-use model. You might pay for storage, networking, compute, containers and other such components, which you can use to deliver your own applications or services. It is, in many ways, like buying parts of a data centre, except someone else deals with the power, connectivity and availability. The major business benefit of IaaS is that you get to give the task of managing physical hardware and data centre space to someone else, and instead benefit from professional and scaled infrastructure, likely more credible than your own. The other major benefit is that you get what you need, and can buy more as your business grows. IaaS is: * Flexible, scalable and adaptable to your needs. * Cost-effective when used right - but pay-as-you-go models with unlimited scalability can quickly explode if flippancy is exercised with use. You want 50 computers to crack passwords for 48 hours? No problem! It will cost you though... * Like renting the foundations of a house. You know the foundations are firm, the power and utilities are installed, but you need to do everything else to your specification.

A good example of IaaS is Amazon Web Services EC2. EC2 provides you with an instance that you can connect to and do (largely) whatever you want with. Unlike SaaS, you build your OS, configuration and software as you need on top of it. You never own the server, you are renting the capacity, but it 'feels' like yours. You can deliver traditional applications and on-premise services via IaaS by lifting and shifting them to the cloud.

**PaaS - Platform as a Service**

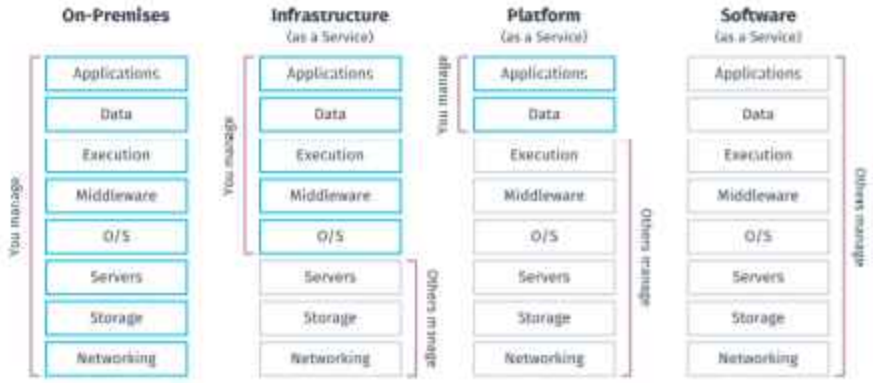
PaaS is where a vendor provides hardware and software designed to enable you to deliver applications. The typical users of these services are integrators and developers. IaaS gives you very basic components, kind of like renting a server, but often when you build applications you want much more than a server. You want a way to manage your code, test your application, build staging environments, manage production and focus on building better applications - not having to manage systems. If you are in the business of building a really great accounting app, you ideally do not want to have to be a true expert in database operations, or load balancing web servers. PaaS takes IaaS components and packages up capabilities for application and service delivery, so you can focus on the creative side of development rather than managing things.

A really good example of PaaS is AWS Elastic Beanstalk. This service from AWS PaaS offering enables you to supply code, and it does the work of stitching together infrastructure components with sensible defaults to deliver it. For many use cases, you can provide your application and 'forget' the infrastructure. There is, however, configuration to enable you to modify the behaviours and settings. There are lots of PaaS providers now like Symfony Cloud, which is designed to enable framework applications to be delivered where you can 'ask' for a database with code rather than having to know how to set one up. These offerings come in various shapes, sizes and costs, depending on your level of expertise and requirements.

Think of PaaS as like renting a house that has solid foundations and utilities. Much of the function of the house has been designed and some rooms are setup for you, but you can decorate as you need using what is there. It's quicker than building your own house from the foundations up!

## What You Get to Manage

Different cloud providers use different models, and sometimes offer more than one. At different ends of the spectrum you get more or less control, and with that can come savings in operations as you outsource management to a third party. This stack is not entirely complete, but may help you identify the portions of control you typically get with each provider. Note, there are always variations, and for example, some PaaS services are built to be very turnkey where others provide a degree of control across more layers. The definitions still largely hold true however.




You should carefully evaluate as a business what is most suitable for your project. You might want to leverage all of these models for different parts of your infrastructure - it is rarely one size fits all.

# Operating Systems 1

## Operating Systems

In this module we are going to dive into the basic terminology and functions of an operating system. These huge pieces of software are what enable your computer and your applications to do all the things you are used to. Understanding the lower levels of your 'OS' will help you immensely when it comes to troubleshooting, advanced configuration or cyber security fundamentals.

## What is an Operating System?

An operating system is software that runs on the computer, which manages how the computer operates. The operating system provides useful functionality such as window management, which allows us to drag programs around the screen, as well as copy and paste, the ability to plug in peripherals such as mice and keyboards, and many other useful things.

### Different Operating Systems

There are dozens of different operating systems available for installation today, the most common of which is Windows.

Other operating systems include:

- Ubuntu Linux
- Mac OS
- Android

Which operating system one chooses to use depends almost entirely on personal preference, however, practically speaking, each Operating System has its own strengths and weaknesses, and the user must evaluate the practicality of the operating system for themselves.

The most notable differences lie in the way the operating system itself looks.

Here are some examples of the desktops of different operating system:

Windows Desktop:

Ubuntu Linux Desktop:

Mac OS X Desktop:

## What is the Kernel?

The kernel is the first part of the operating system code to be loaded, and has complete control over the computer. It is responsible for controlling access to the computer's most sensitive information and functionality, and is loaded into a protected region of memory to prevent accidental or malicious corruption by other programs.

The kernel is responsible for the loading of new programs, handling input and output between peripherals, and managing access to the hardware's shared resources, such as the RAM and hard drive.

## What is a Process?

The definition of a process is: "A series of actions or steps taken in order to achieve a particular end". In the context of computers, this is much the same. A process is created on a computer when a program is requested to be loaded. The computer will load the code in the program, such as Google's Chrome browser, and will create a process that contains all of the necessary information for execution by the processor.

Each time you run a program on your computer, a new process will be created in order for the program to serve the purpose you intend, which, in this case, would be surfing the internet. Some programs even create many processes in order to get more work done.

## What is an Interrupt?

An interrupt is a signal that is sent to the CPU, which alerts the CPU to a task requiring its immediate attention. An interrupt will halt the CPU and cause it to begin executing the corresponding interrupt handler.

Interrupts are used in situations where immediate processing is required, such as when a peripheral is plugged in, like a mouse. A user would not be happy if it took 10 seconds for the mouse to be recognised.

## Hardware Interrupts

A hardware interrupt is generated by some sort of hardware either inside or outside of the computer. This includes a keyboard or mouse, which will send interrupts when new input is available. The interrupt handler will then be executed, and the window which is currently in focus will receive the keyboard input via the operating system.

## Software Interrupts

A software interrupt is an interrupt which is generated by software. A program may want to generate an interrupt when it wants to open a file for example. The program has to do this because the act of opening a file is a privileged operation, and as such the kernel has to be consulted before the operation takes place. Software interrupts act as an interface between the kernel and the program. In short, the program can use a specific interrupt to ask the kernel to perform an operation on its behalf.

## What is the Bootloader

The bootloader is a program that is loaded by the BIOS when a computer is first turned on, and is responsible for loading the operating system. A bootloader is necessary due to the complexity that would be involved in creating a BIOS capable of loading hundreds of different operating systems. Due to this, the bootloader is usually installed at the same time as the operating system, and is loaded from a known place on the hard drive by the BIOS.

## The BIOS

A BIOS, or "Basic Input Output System", is a program stored on the motherboard of your computer. When the computer is first turned on, it is the first program that is loaded, and prepares, or initialises, the hardware ready to load the bootloader.

Nowadays, modern computers almost exclusively use UEFI (Unified Extensible Firmware Interface). UEFI is the successor to BIOS; however, the term BIOS is commonly used as an umbrella term for both, due to the similarities between them. UEFI achieves the same functionality as BIOS, in a more versatile and secure manner.

# Virtualization

## Virtualization

Virtualization is arguably one of the most instrumental technology architecture changes to occur in a long time. It has changed the way we deploy systems, test systems, and the cost of ownership of services and applications.

It comes in many shapes and sizes, and herein we will learn more about how it works and what it can be used for. You will be getting lots of practice with virtualization, and it powers this very course's lab system!

## What is Virtualization?

Virtualization is where we create a virtual computer or 'virtual machine' out of software that behaves like a separate computer. All the hardware components of that virtual machine are actually software. The software that is the virtual machine uses the hardware resources of the computer it is running on. This is achieved using a hypervisor, which creates a thin layer that breaks the traditional 1:1 relationship between an operating system and the hardware. There are various types of hypervisors we will cover in more depth later.

Put simply, we can build a virtual computer out of software, and use that to run a different operating system on top of our main operating system. Imagine running Windows at the same time as running Mac OS X for example. Or even running Windows while running Linux. All this and more is possible with virtualization.

Here we have a Linux system running VMware (a popular virtualization software); the operating system inside the virtual machine is Windows 10.

### Host Operating System

We call the operating system that runs the virtualization software the host operating system. It is the master, the operating system that loads when you first turn on your physical computer.

### Guest Operating System

We call the operating system that runs inside the virtual machine the 'guest operating system'. The guest operating system is separate from the host operating system, and cannot directly access the resources on the host. For example, if you were running

Windows as the host operating system and Linux as the guest, the Linux guest is not able to access files on your Windows desktop directly. This separation is a key use case for virtualization. If you are analysing a computer virus, you will want to be doing it in a virtual machine to prevent it from spreading to your host operating system and potentially escaping out onto the internet from there.

## The Hypervisor

Before we talk about the different types of virtualization, we have to talk about the hypervisor. The hypervisor is the layer of code that allows multiple operating systems to share the same hardware resources. Essentially, it's the bit of code that directs traffic, deciding which bits of memory are used for which virtual machines, where the hard disk for that virtual machine is kept in storage, and so on.

### Type 2 Hypervisor

The first type of virtualization we'll talk about is the type 2 hypervisor. This type of virtualization is where the virtualization is done by a software program that runs on an operating system. This is the kind of virtualization we'll be using throughout this course. Several software programs can be used to perform this; the most popular ones are VMware Workstation and Virtualbox.

### Type 1 Hypervisor

The second type of virtualization is the type 1 hypervisor. This type of virtualization is where the virtualization occurs at the firmware level. This is still software, but there is no host operating system. The virtualization software in effect is the host operating system. This is commonly used in server environments such as data centres, particularly ones that form 'the cloud'. It is a more efficient setup than a type 2 hypervisor, but it isn't convenient for people to use on their personal computers.

Some examples are VMware vSphere and Proxmox.

## Uses of Virtualization

Virtualization sounds pretty niche, so why is it so important? Well, you might be surprised to learn that a large proportion of the internet runs on virtualised servers. Ever heard of the cloud? Well, most of that is run off of Type 1 hypervisors. The fact of the matter is virtualization offers huge efficiency improvements for large-scale applications.

Imagine if you have a website which is usually quiet, but often it will suddenly get a huge deluge of users accessing it all at once. You need a server that can handle the maximum number of users, but that will leave your server doing nothing most of the time when it's quiet. With virtualization, you can run the site off of one small virtual machine, and have new virtual machines come online during peak times to load balance your traffic across the multiple servers. When it becomes quiet again, you can delete some of the virtual machines and go back to having only one serving the website. Cloud hosting providers such as Amazon AWS offer this functionality using virtualization.

### Uses in Security

In cyber security, we make extensive use of virtualization. Usually, we run a type 2 hypervisor (remember this one is the software that runs as an application on your host operating system). We do this because we often have to use multiple operating systems all at once; because some tools will only run on Windows, and some will only run on Linux. Before virtualization, we would have to install two operating systems on one hard drive and reboot to swap between them. With virtualization, we can run Windows as the host operating system and run Linux as the guest operating system, or vice versa. Working this way is a lot more efficient.

Beyond simply having access to all our tools, virtualization offers separation. The applications running in the guest operating system cannot interfere with the host operating system. That means we can work on dangerous tasks such as analysing the latest malware without risking infecting our host operating system.

**Disclaimer:** If you do any malware analysis, make sure you disable all virtual machine communication methods, such as the virtual network adapter.

### Uses in Development

Many programmers make use of virtual machines to test their programs in different environments. If you were making a program in Windows and you wanted to see if it would run on Linux, you could just spin up a new virtual machine quickly and test it out.

## Setting up a VM

In this section we will review the procedure to set up a virtual machine and some of the key configuration parameters. Different virtualization solutions use different terminology, but essentially have very similar features. Whether you use VirtualBox, VMware, Hyper-V or another product you will find a great deal of these concepts transferable.

Being able to build a virtual machine is an incredible useful skill when it comes to practice and sharpening your skills. It is also invaluable when you want to replicate real networks and scenarios - for example building your own mini network of a web server, database server and log server. Using virtualization we can test and replicate complicated setups on just one device!

It is strongly recommended that you take the time to setup your own virtual system using VMware, VirtualBox or Hyper-V take the challenge to install a copy of Ubuntu. This is a widely documented procedure, so seek help online if you need it and treat it as a challenge!

# Introduction to Linux

## Linux

There are Linux distributions for almost every specialist task in cyber security, but Linux is also incredibly modular and a powerful platform from which to chain together tools for complex tasks. In this module we will introduce you to Linux and how to get your own system up and running for test purposes.

## Learning Objectives

After completing this module, you should be able to:

- Know the difference between server and desktop versions of Linux.
- Know the difference between different fiavours (distributions) of Linux.
- Install Linux.
- Set up networking on Linux.
- Navigate the GUI on Ubuntu Budgie. Many Linux distributions have common features.
- Access the terminal.

## Module Content

This module will provide a basic introduction to Linux, focusing on familiarising you with the desktop interface.

We will be covering the following components:

- What is Linux?
- How to install Linux.
- Navigating the Linux desktop GUI.
- How to set up networking on Linux from the GUI.
- What is the Terminal?
- Accessing the Terminal.

## What is Linux?

When we talk about Linux, we are really talking about the Linux kernel. The kernel is the core of an operating system, the part that interfaces directly with the hardware components. Linux is more of a class of operating systems than one single operating system. The common aspect that all of these operating systems share is they all use the Linux kernel. We call all the operating systems that use the Linux kernel, 'distributions'.

## Linux Distributions

Linux distributions come in all shapes and sizes. Linux is ultimately an incredibly powerful and customisable operating system, and we can see this reflected in the variety of distributions available.

### Desktop Distributions

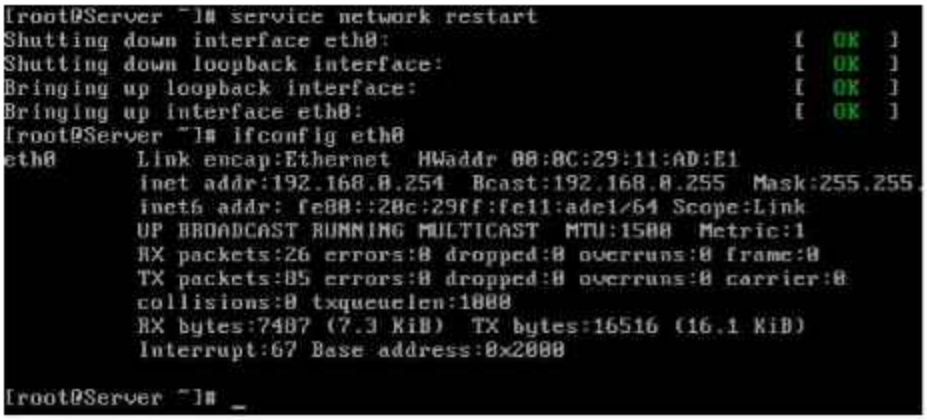
Linux desktop distributions come with a GUI (Graphical User Interface). Historically the GUI on Linux has been terrible; more of a bolted on afterthought than a real usable product. More recently, that has been changing, and there has been a real push to make Linux accessible to more people.

Here we have a screenshot from Elementary OS, a Linux distribution which focuses on providing an operating system that is as easy as possible for non-technical people to use as a replacement for Windows or OS X. Compare that with this older Linux GUI, and you can see the difference right away:


There are distributions with other focuses too, such as Kali Linux which is a distribution built for cyber security professionals. It comes customised with most of the cyber security tools you might need pre-installed:

**Server Distributions**

Just as with Windows, Linux distributions that are specialised for use on servers exist also. These don't come with any GUI installed at all; everything must be done using text-based commands in the 'terminal'. In fact, Linux was originally command line only and didn't have a GUI at all. To this day, the most powerful way to get something done on Linux is to use the terminal, even on desktop distributions.




Although this may look intimidating, by the end of this course you will be comfortable with the Linux terminal.

## Installing Linux

In this walkthrough we will install a new Linux virtual machine from scratch. We will be focusing on the desktop installation process here and some sensible defaults, as well as looking at how to enable rudimentary encryption for the system. Options will vary depending on the system you are installing on and the distribution, but this will act as a robust framework for most installation processes.

There are huge numbers of resources available online to help you setup Linux in all shapes and sizes, and it is well worth the time setting up a few different systems and getting used to what often goes right, and wrong.

## Installing a Linux server

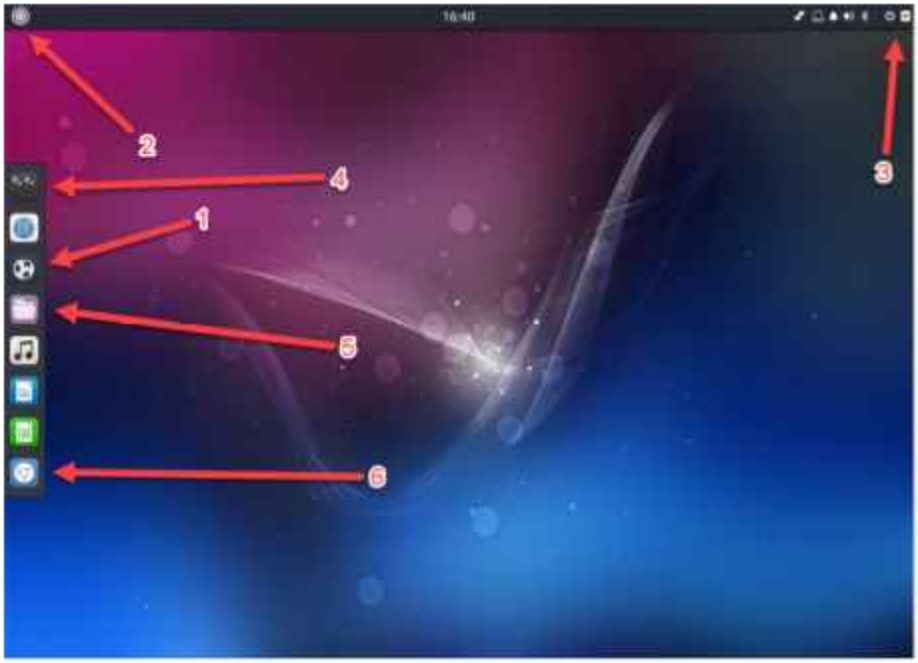
In this walkthrough we take a look at the installation process for a Linux server and how it differs from a typical Desktop. The user interface is typically built for compatibility with remote access systems and technologies as it is likely you may be doing this installation remotely, or you may be installing to a server that has limited graphics capabilities - focusing its resources instead on the computational tasks it is likely to perform. The process is a little different but once you know how to navigate the options using the keyboard, such as toggling with the space bar, it is ostensibly the same.

Downloading a Linux server and trying this in your own virtualisation software is a really good way to practice and build familiarity, that will aid you in later portions of the course.

## Navigating the Linux GUI

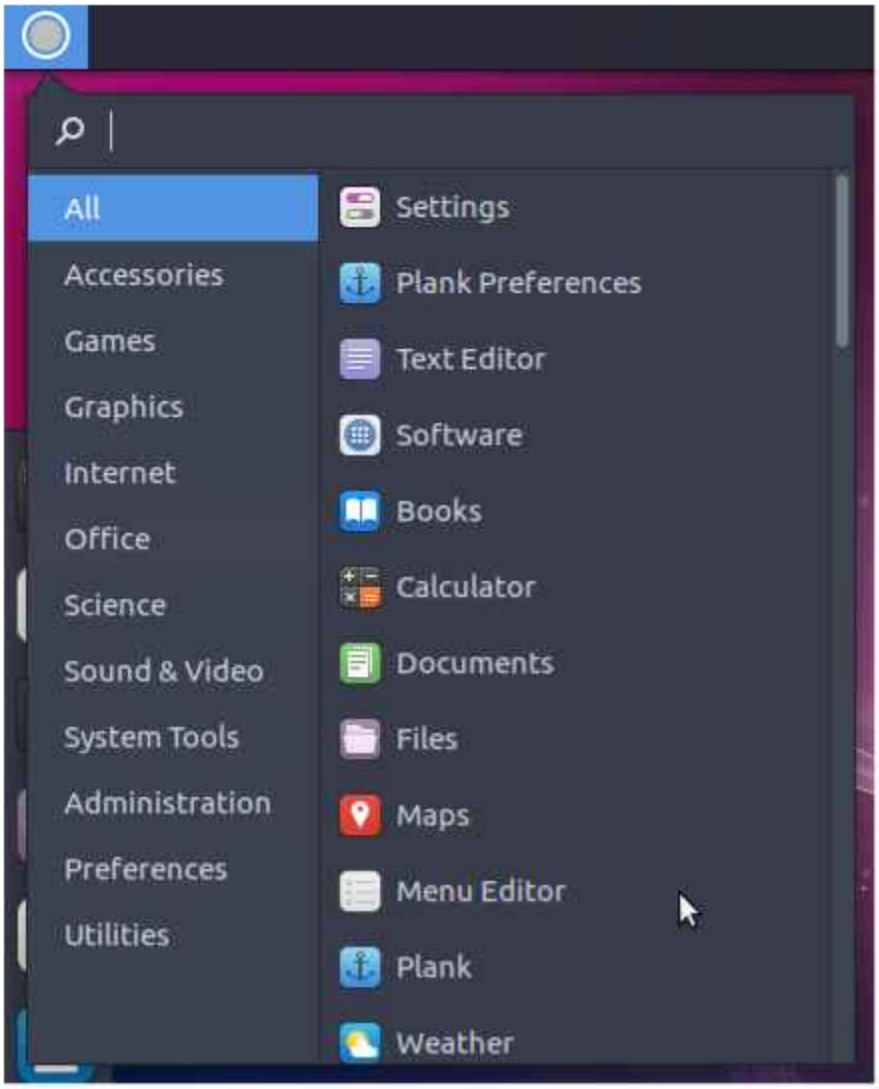
There is no single GUI or layout for Linux. Every distribution will have its own layout, and even the software used to display the GUI can be different from distribution to distribution. A lot of the time, you will be left to explore for yourself. However, there are plenty of common features that are available across distributions, even if the locations may change.

Here is an example of an Ubuntu installation. After installing, the desktop will look like this:




At position 1 we have the 'dock'. The dock is where you can save your frequently used applications, and any running applications can also be accessed from here. It's like the Dock on OS X or the taskbar on Windows.

At position 2 we have the start menu, by clicking it we can bring up a Windows-like start menu, which lists all applications. It is also searchable if you type after clicking on it.




At position 3 we have the system tray, which is similar to the system tray on Windows. From here you can shut down, reboot, update or access other settings such as volume, Bluetooth, notifications and network settings.

At position 4 is the terminal application icon. This application allows you to use the Linux command line even from a desktop distribution. By default, it occupies the first position on the dock, which should give you an idea of its importance.

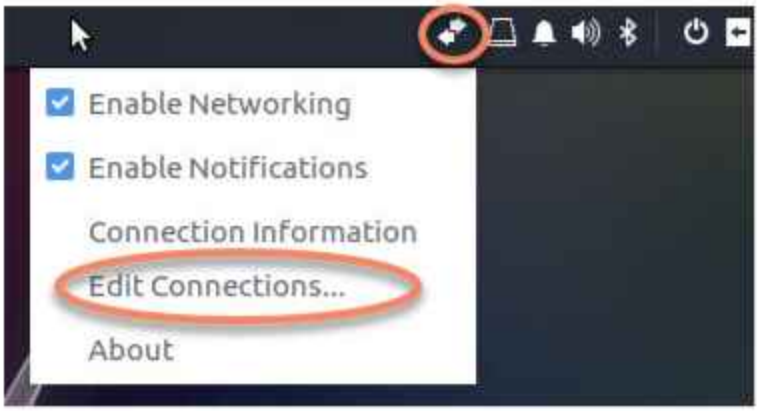
At position 5 is the file browser application, which works just like Explorer does in Windows. You can navigate through the folder structure of your installation.

At position 6 is the web browser which is installed by default; in this case, it's Chromium, a version of Google Chrome. Just above it, you will see a blue and green icon. These are icons for LibreOffice Writer and LibreOffice Calc, just as they sound they are basically copies of Microsoft Office, except Open Source and free. They work with normal Microsoft Office files also.

## Configuring Networking with the GUI

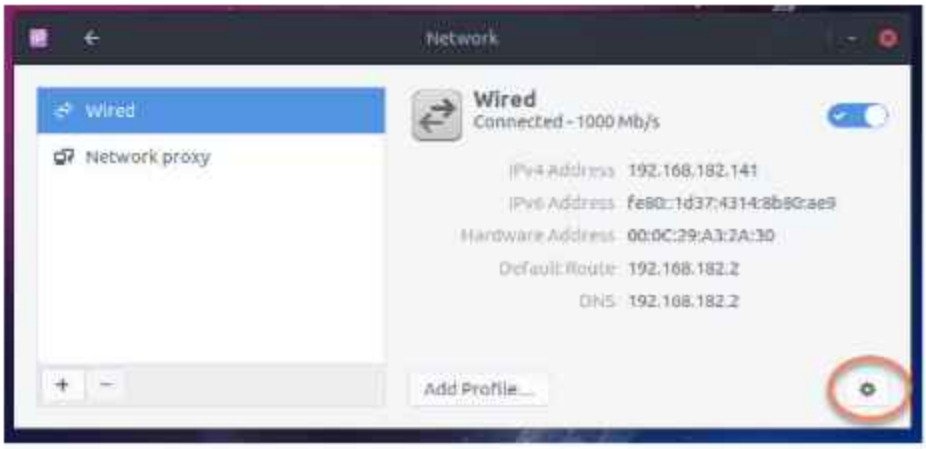
The best way to configure networking on a Linux system is actually to do it from the command line. You'll learn to do that in a later module, but for now, we'll show you how it's done through the GUI.

The first step is to find the Network Settings Preferences page. There are two obvious ways to get to it. The first is to right-click on the network icon in the system tray and go to 'Edit Connections':

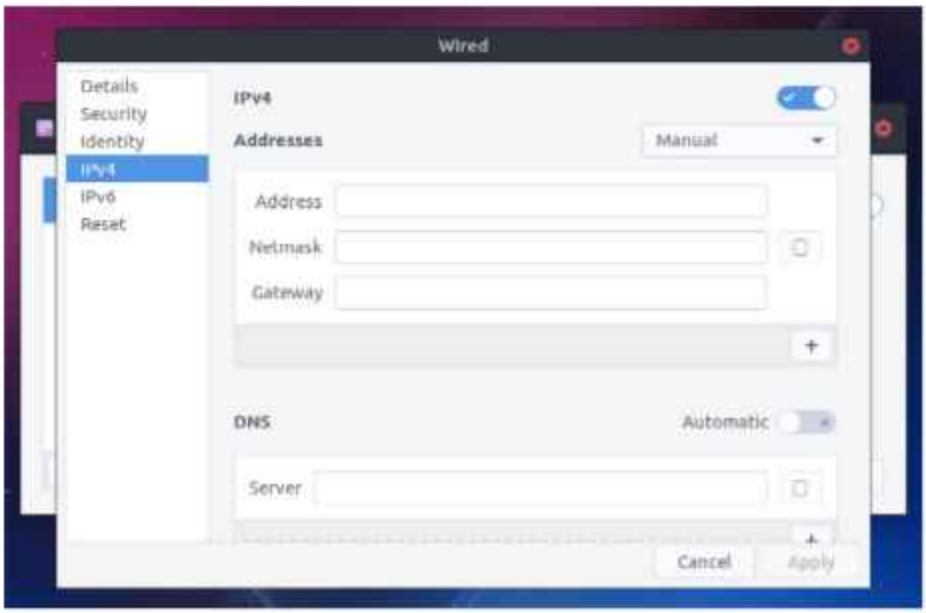



The second way to get to the Network Preferences page is to go via the 'Start' menu and select 'Settings', before selecting 'Network':

Once you have the network settings open, you can select the connection you wish to edit and click on the 'gear' icon in the bottom right:




From here you can select IPv4 (or IPv6 if that is what you use) and change it from DHCP (which is automatic) to 'manual' and assign custom settings if you wish.




If you do make some changes, don't forget to hit 'Apply', of course.

## Linux Terminal

We already covered the Linux terminal briefiy. Originally, Linux did not have a GUI, and everything was done from the command line. Eventually, a GUI was written and incorporated into many desktop distributions, but ultimately the GUI is still just a thin facade over the command line. This structure is the opposite to Windows where the GUI is the core of the operating system, and some things can't be done from the Windows command prompt at all. Ultimately, this means that on Linux the terminal is incredibly powerful.

When you open the Terminal you are first met with something like the following:

```
user@SANS:~$
```

This is called the 'prompt'. In other words, you type your commands in there, as it's 'prompting' you for input.

On this installation, the user account is called 'user', since we pride ourselves on our creativity here. The name (hostname) of the computer (that is the name that identifies the computer on the network), is 'SANS'. On Linux, the tilde character (~) is shorthand for the home folder of the current user. So this prompt gives us quite a lot of information. We know our username, we know the name of the computer, and we also know where we are in the file system. As the root user (the highest privilege level) this prompt will typically change from a $ to a # to signify the shift.

The prompt can be customized, as almost everything can on Linux, so it won't necessarily always look like this, however this style of prompt is a very common default. We will customize it later in the course.

### Shell

When you open the terminal, it runs a program automatically. The program that it runs is called the 'shell'. The shell is responsible for displaying the prompt, interpreting the commands you type, running programs and displaying the output to you. There are many different shell programs, but by far the most common currently is the 'bash' shell.

Bash stands for Bourne Again Shell; it's a clever play on words. One of the earlier shell programs was the Bourne shell, and Bash was written as a modern shell program to replace it. The default path for bash is /bin/bash. The first '/' means the root of the file system, from the root of the file system, in the 'bin' folder is the 'bash' program.

You can select a different shell and customize this heavily, introducing powerful new functionality but this setup is the most common starting point you will find.

# The Linux Environment

## Contents

In this module, we will be covering:

- Superuser
- Navigation in the Terminal
- Folder Structure
- File Permissions
- Hidden Files
- Environment Variables
- PATH

This will start to build your knowledge of Linux and the concepts that help you secure it.

## Superuser

The superuser account on a Linux system is effectively the administrative account. It's the account with total control over the operating system and has permissions to do anything and everything. There is always at least one superuser account on every Linux system, and the account name is usually called 'root'. I've yet to see a Linux system where the superuser account was called anything other than 'root', although given how customisable Linux is, it's possible there could be one distribution of Linux that doesn't follow convention.

On a Windows system, the administrative account isn't actually capable of doing anything it wants on the system. There are some things the administrator isn't qualified to do, typically things that would break the operating system irrevocably. On Linux, the root account can do absolutely anything it likes, even up to and including breaking the operating system.

On Windows, anything you do is usually followed by, "Are you sure you want to do this?". After you answer yes, you get another prompt "Are you really really sure you want to do this?". Alright, that's something of an exaggeration, but the point remains. Linux likes to keep things interesting though; it doesn't like to prompt you about anything, particularly if you are root. Typically, you'll type in a command, and the operating system will just do it. Delete the whole file system? Sure, all gone. You're root, so you're allowed to do that.

Using the root account for daily activities is a bad practice. You should always log in to a system as a normal user account. From there, you can switch user to root if you need to. Here is the procedure in action:

```
user@sans:~/Desktop$ whoami
user
user@sans:~/Desktop$ su root
Password:
```

```
root@sans:~/home/user/Desktop# whoami
root
root@sans:~/home/user/Desktop# exit
exit
user@sans:~/Desktop$ whoami
user
user@sans:~/Desktop$
```

You can see we start as the 'user' account. We then used 'su' to switch user (that's what 'su' stands for) to root. After entering the password (you can't see the characters as they are typed, as it's secure input) we successfully become root. After we're finished doing root-y things, we can drop back to being a normal user by typing 'exit'.

Alternatively, you can use a program called 'sudo' if it is installed on your system. The 'sudo' program will allow you to temporarily take on the privileges of the root account to run a command and then it will drop your privilege level back down to your normal account levels after the command runs.

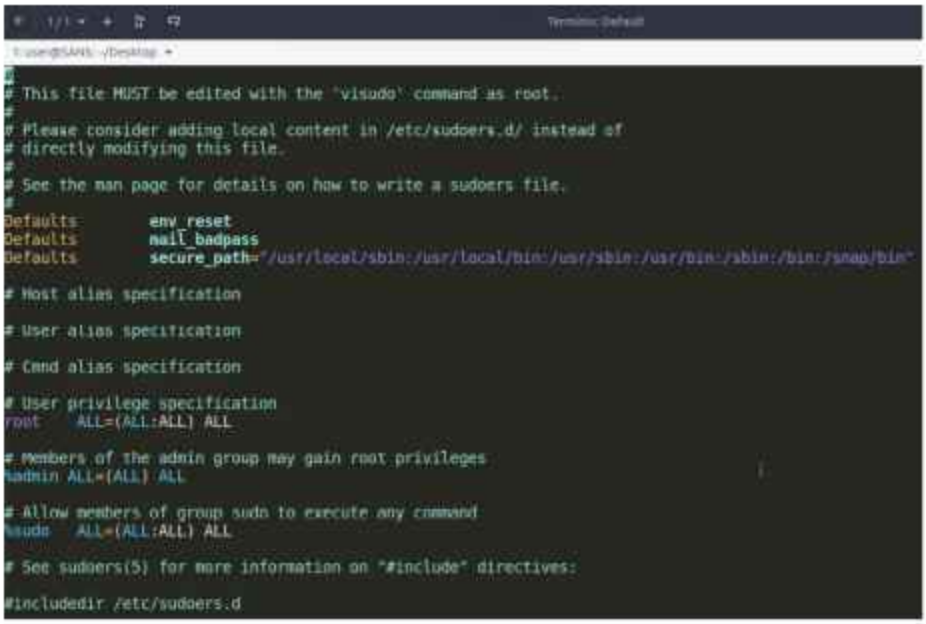
The way 'sudo' works is there is a configuration file called the 'sudoers' file, which is basically a list of which accounts are allowed to do what with superuser privileges (and only root can edit it). When you want to run a command with privileges, you append 'sudo' before the command. You will then be prompted for your normal account password (not the root password), and then the command will run with superuser privileges. Many distributions that come with 'sudo' installed will not tell you the root password (it will be randomly generated and thrown away). This is to encourage you always to use 'sudo', which is good practice.

Take a look:

```
user@sans:~/Desktop$ whoami
user
user@sans:~/Desktop$ sudo whoami
root
user@sans:~/Desktop$ whoami
user
user@sans:~/Desktop$
```

You may have noticed that we weren't prompted for a password this time. That's because we entered the password a few moments before taking the screenshot, and it caches for a short period of time after you enter it once.

And here is an example of the 'sudoers' file:




It's not as complicated as it looks, although we haven't covered groups yet. In Linux, a user account can belong to groups. For example, our user account is a member of the 'sudo' group. And we can see in the sudoers file here that the 'sudo' group is allowed to execute any command as root. The most important piece is a section labeled user privilege specification. It has lines that define user rights such as %sudo ALL=(ALL:ALL) ALL allowing sudo group users to run any command.

Just to prove it, here are the groups our user account is a member of:

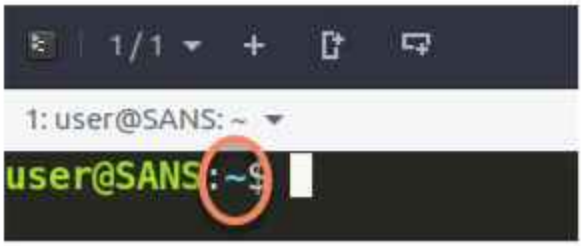
```
user@sans:~/Desktop$ groups
user adm cdrom sudo dip plugdev lpadmin sambashare
user@sans:~/Desktop$
```

**Note:** Beware editing the sudoers file yourself. You could lock yourself out of the superuser account entirely! If you must edit the sudoers file, use the 'visudo' command, which is a command line text editor that will check your sudoers file for any errors before it saves it. That doesn't make it impossible to lock yourself out, but it lowers the chances of a typo locking you out.

## Navigation in the Terminal

We're going to be using the terminal quite extensively from now on, because it's the only thing you are guaranteed to have access to, no matter which Linux system you are working on. The first step to using the terminal is figuring out how to navigate the file system.

When you first start the terminal, you will usually be in your user's home folder. The shorthand display for your home folder in Linux is the tilde character (~). You can see it here:



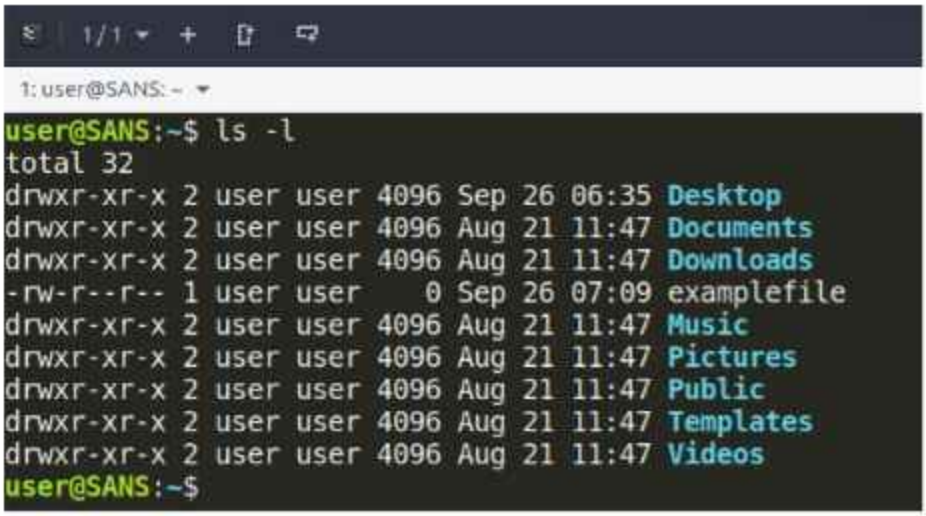

The circled bit is the area of the prompt that tells you where in the file system you are.

### Listing Files

We can list files and folders with the 'ls' command. The 'ls' command stands for 'list':


We can see here that folders are listed in a teal colour, while files are in white. The colour scheme may differ on your system, or you may not have one at all, so I'm not a fan of this default view.

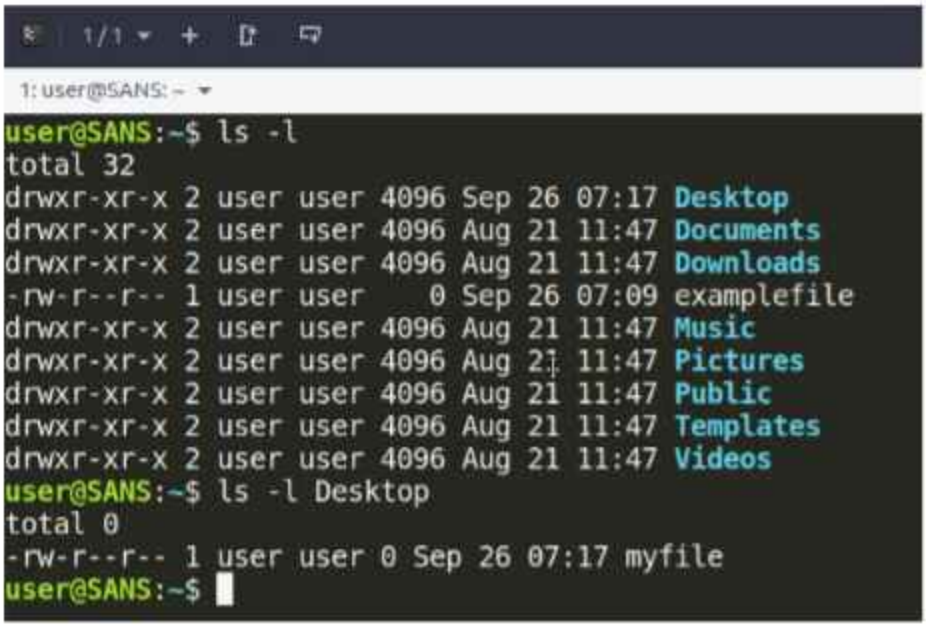
You can pass in the '-l' parameter to the 'ls' command to get it to print in long form ('-l stands for long'):




I find this view to be a lot clearer. The information here is: permissions, user, group, file size, creation timestamp, file/folder name.

We haven't covered file permissions yet, so don't worry too much about them, but you'll notice that for the permissions, every folder starts with a 'd', but the files start with a '-'. The 'd' stands for 'directory', so it explicitly tells us that this is a folder, or this isn't a folder. On a system where you don't have any colour coding, this can make your life a lot easier.

By default, 'ls' will list the files and folders in your current working directory. In this case, this is our home folder for now. You can also pass in a path to 'ls' and get it to list the files and folders in a different directory, like so:

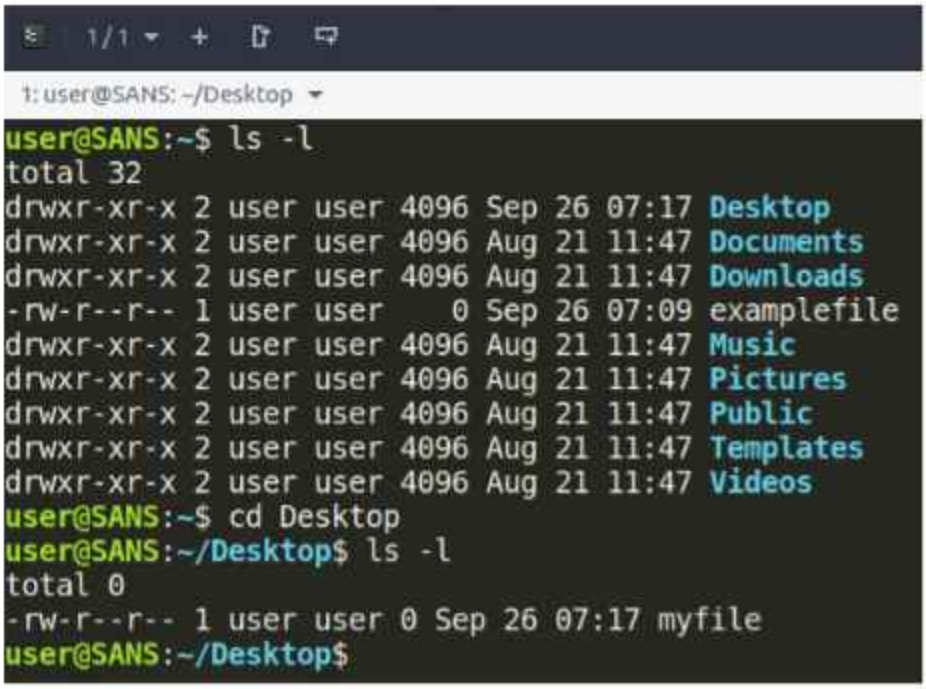



We first asked it to list the files and folders in the current working directory. We saw the 'Desktop' folder was interesting, so we then asked it for the files and folders in the

'Desktop' folder.

### Changing Directory

So far, we've been trapped in our home folder without the ability to move around the file system. Let's fix that. The 'cd' command allows us to change directory into a different folder. Yes, you guessed it, 'cd' stands for 'change directory'. Let's see how it works:




We found the 'Desktop' folder; this time, we change directory into it (notice how the prompt updates to show us where we are), and then we list the files and folders in the current directory again, and we see we are now in the 'Desktop' folder.

Now, to get back to the home folder, we could do one of three things. We could type:

```
cd /home/user
```

or we could type:

```
cd ~ (remember the tilde is shorthand for the current user's home folder)
```

or we could type:

```
cd ..
```

In this last case, the '..' is another kind of shorthand; it stands for the directory one level above the current directory, also known as the parent directory (not to be confused with the single '.', which is shorthand for the current directory). Since we are in /home/user/Desktop, the directory one level up would be /home/user, which is where we want to go. This shorthand will save you a metric tonne of time; make sure you drill it into your head.

Let's see how it works:

```
user@sans:~/Desktop$ cd /home/user
user@sans:/home/user$ pwd
/home/user
user@sans:/home/user$ cd Desktop
user@sans:~/Desktop$ cd ~
user@sans:~$ pwd
/home/user
user@sans:~$ cd Desktop
user@sans:~/Desktop$ cd ..
user@sans:~$ pwd
/home/user
user@sans:~$
```

You may have noticed our use of the 'pwd' command. The 'pwd' command stands for 'print working directory', so it will show us the path of our current working directory.

## Folder Structure

Now that we know how to get around in the terminal, we need to know where we are going.

The Linux file system starts at the root of the file system (known as '/'):


From the root of the file system, you can get anywhere. For example, '/home/user' was our home folder. Notice how the first '/' indicates the root of the file system? So it could be read as, go to the root of the file system, then go to the home folder and then go to the user folder.

Let's go over how the folders in the root of the file system are typically used (if we don't mention a folder, then it doesn't exist on every typical Linux system):

- **bin:** The bin folder is typically used to store executable files (binary files). These will usually be system files, as opposed to ones the user installed.
- **boot:** The boot folder holds the files that Linux uses during the boot up process. Better not mess around in here unless you know what you are doing.
- **cdrom:** The cdrom tray will usually be mapped to this folder, so if you have a cdrom plugged in you can usually access the files on it from here.
- **dev:** The dev folder will contain a folder and associated files for every hardware component on the system. For example, there is /dev/cpu, and there are even entries for your hard drives. Usually, you don't want to mess around in here.
- **etc:** The etc folder will usually contain configuration files for installed programs. If you need to change a setting for a program you've installed (for example, a web server), then this is the first place you'll want to look.
- **home:** The home folder contains the user directories for every user on the system (that a user can log into) with the exception of the root user. Our user is called 'user', so you can find our home directory in /home/user.
- **lib:** The lib folder contains shared libraries and kernel modules. These are resources that the system uses to function, usually best not to mess around in here either.
- **lost+found:** If your hard drive has errors and files get lost, the 'orphaned' files may get placed here.
- **media:** The folders here can be used for mounting USB keys and fioppy disks (if anyone can still use one). To mount one is basically to load the file system that exists on it so you can access the files.

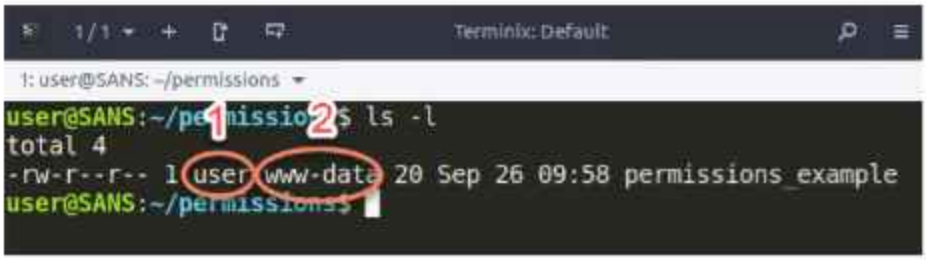
- **mnt:** The folders here can also be used for mounting external drives, USB keys and fioppy disks. It's up to you if you want to mount stuff here or in media.
- **opt:** The opt folder is usually empty to start with; any user-installed programs can go here (if you want). It stands for 'optional'.
- **proc:** The proc folder stands for 'process'. Every running program will have an entry in the proc folder along with associated files.
- **root:** The root folder is the home folder for the root user. Remember, it isn't in /home.
- **run:** The run folder is a temporary file system which stores runtime information for programs that start early during the boot up process.
- **sbin:** The sbin folder is used to store binary executables (similar to /bin), but the programs stored here are typically used for administrative purposes.
- **srv:** The srv folder usually holds data used by services running on the system (such as a web server, or an FTP server).
- **sys:** The sys folder contains information about devices on the system (as seen by the Linux kernel). Usually, you don't want to mess around in here.
- **tmp:** The tmp folder is a temporary file system. The files in there are temporary and will be deleted periodically, and after reboot. Programs will usually use it frequently. Just don't use it to store your important files!
- **usr:** The usr folder is the folder for user-controlled files. It has its own folder structure, which maps to the root folder structure. For example, there is a /usr/bin folder which is for user-installed binary executables.
- **var:** The var folder contains system files which tend to increase in size over time (hence it's a **var**iable size folder). Things like log files, the mail directory, and so on, go here.

Now, these are just the **typical** uses of these folders. We're talking about Linux, so if you want to put your user-installed binary executables in /var, no one is going to stop you. It is good to know where you can expect to find things, though, and if you can't find the configuration files for that program that you installed in /etc, you can always refer to the documentation for that program.

## File Permissions

File permissions in Linux is a topic that many people refuse to learn, but they only look intimidating. The truth is, they are quite easy and logical.

The first thing you need to know is that each file and folder is owned by a user and also by a group:




At position one is the user that the permissions example file is owned by. In this case, the file is owned by our user account which is called 'user'. At position two is the group this file is owned by. In our example, this file is owned by the www-data group, which is typically used by web servers.

Now, there are three permission modes which can be set for each owner of the file.

- r: Read Permissions
- w: Write Permissions
- x: Execute Permissions (if it's an executable, then you can execute it)

Simple, right?


The place where the permissions for this file are listed is circled in the image above. Each owner of the file gets their own set of permissions.

The first '-' is reserved for special permissions or to indicate if this is a directory or a file. If it is a directory the first '-' will be a 'd'. Other than that, ignore this first '-' for now.

The next 3 '-'s are reserved for the user permissions. This file is owned by our user, the creatively named 'user' account. Currently, the user owner of this file has no permissions at all, so it can't do anything with this file.

The 3 '-'s after the user permissions are for the group permissions. The file is owned by the group 'www-data', so any account that is a member of that group will have the

permissions shown there to access this file. Currently, no permission to do anything with this file at all.

Finally, the last 3 '-'s are for anyone else. If you aren't the user owner, and you aren't the group owner, then what access do you get to this file? Currently again, no permissions at all.


Here we've changed the permissions to show you what that looks like. This file now has the following permissions:

**The user owner, 'user' can:** Read, Write and Execute this file (because the permissions are rwx).

**The group owner, 'www-data' can:** Read and Execute, but NOT Write (because the permissions are r-x).

**Anyone else can:** Write, but not Read or Execute (because the permissions are -w-).

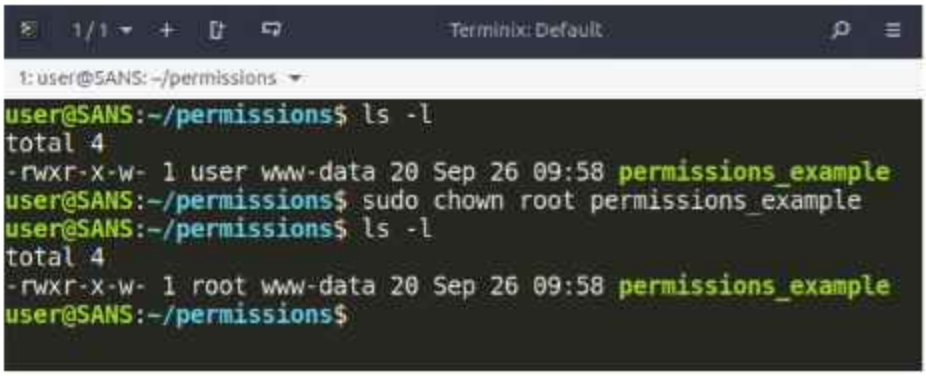
Notice how the file name is now in green? That's colour-coding showing us the file is executable because we are the user owner, and the user owner has permission to execute this file.

### Setting File Permissions

So we can all read file permissions now, right? Setting them is slightly trickier, but not much!

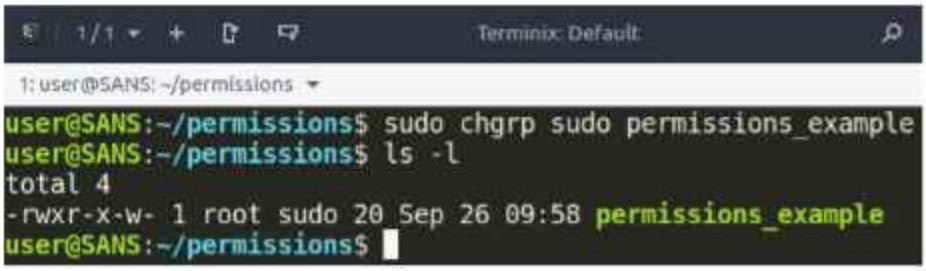
In order to change permissions you need to have the rights to do so. If you own the file that means you can. Sometimes you need to override this with higher permissions, by using sudo. It is possible to change permissions to a file so you can no longer access it, and in this scenario the root user is the last line of defence to be able to make changes. In the examples below, sudo is used because the "user@SANS" does not have right to change the file permissions.

We can use the 'chown' command to change the owner of a file to a different user account:



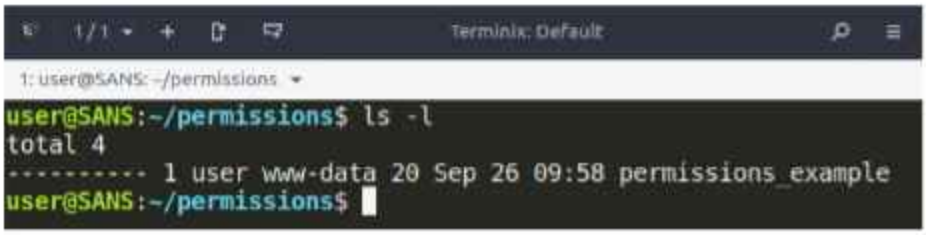

As you can see, we changed the account that owns this file from 'user' to 'root'. Notice the syntax of the 'chown' command takes the account you want to change the owner to first, and then the file you want to affect next.

You can also change the group owner with 'chgrp':



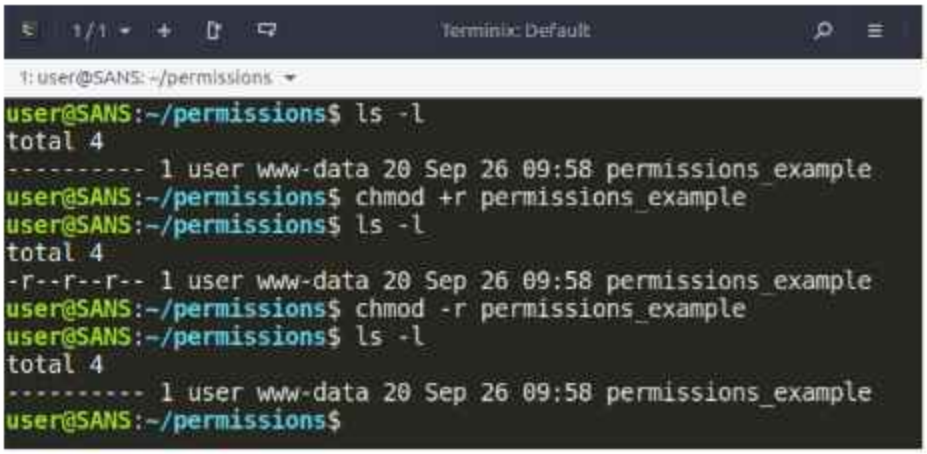

Now we changed the group owner from 'www-data' to 'sudo'. The syntax is very similar to the 'chown' command.

We're going to reset the permissions on our example file before showing you how to change the file permissions themselves:



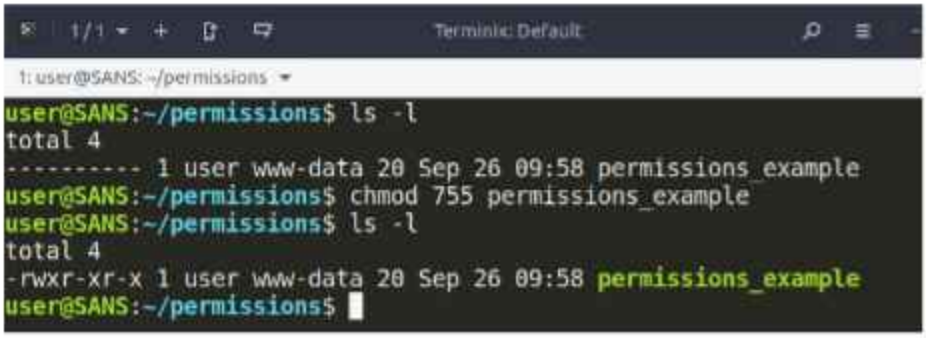

Here we go: the user and group have been changed back, and we have a blank set of permissions. Now, we can set the file permissions themselves using the 'chmod' command.

You can add read permissions using `chmod +r filename` for example:




Notice we added read permissions to every set of permissions, including the everyone set. This is a really quick and easy way to set permissions (it also works as +w or +x), but it isn't very safe, because it sets all three at once. You can also remove them with '-', as in '-r or -w or -x'.

A far better way of setting permissions is to use a numerical value to explicitly set every permission at once. For example:




So the question is, how did we get from 755 to those particular permissions? Actually, as long as you can count in binary, the answer is pretty easy. You just have to treat the permissions like a three-digit binary number:

| R | W | X |
|---|---|---|
| 1 | 1 | 1 |

Here we have three bits, one bit for read, one for write, and one for execute. If they are all on, then by converting the binary number 111 to denary, you get 7. So a 7 will set read, write and execute permissions.

| R | W | X |
|---|---|---|
| 1 | 0 | 1 |

Similarly, 5 in denary is 101 in binary, so that will set read and execute, but not write permissions.

We used 755 in our chmod command, so that means the user can read, write and execute. The group can read and execute, and everyone else can also read and execute.

Let's say we want to set a file to:

User: r-x Group: --x Other: -w-

This is a weird set of permissions, but let's calculate it anyway.

User:

| R | W | X |
|---|---|---|
| 1 | 0 | 1 |

0b101 = 5

Group:

| R | W | X |
|---|---|---|
| 0 | 0 | 1 |

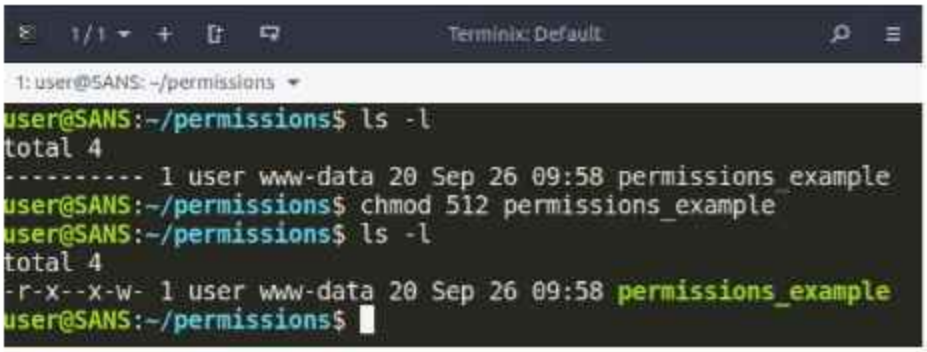
0b001 = 1

Other:

| R | W | X |
|---|---|---|
| 0 | 1 | 0 |

0b010 = 2

So the permissions we need to use are 512:




That worked perfectly.

### The Short Way

There is a shorter way than counting in binary every time. You just have to remember:

Read = 4 Write = 2 Execute = 1

Then just add them up to get your final permissions.

Setting read and execute? That's 4 + 1 = 5.

What about read and write? That's 4 + 2 = 6.

## Hidden Files

In just about every operating system, there is a way of designating files and folders as 'hidden' and therefore hiding them from the user. In Linux, the method is simply adding a '.' to the start of the filename. For example: '.myfile' is a hidden file:

```
user@sans:~/hidden$ ls
user@sans:~/hidden$ touch myfile
user@sans:~/hidden$ ls
myfile
user@sans:~/hidden$ touch .myfile
user@sans:~/hidden$ ls
myfile
user@sans:~/hidden$ ls -a
. .. .myfile myfile
user@sans:~/hidden$
```

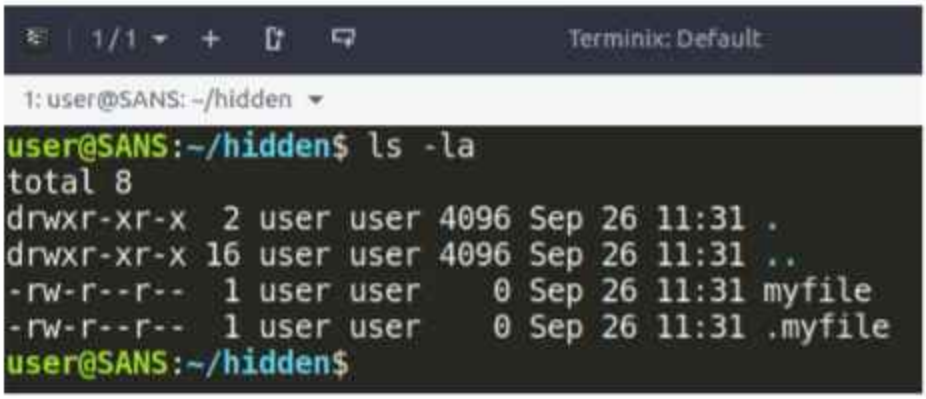
You haven't seen the 'touch' command before. Without going into specifically what it is used for, if you use the touch command on a file that doesn't exist, it will create a file with that name, which is empty.

We start with an empty directory, then we touch a new file called 'myfile' and then using 'ls' we can see that the file has been created.

Then we create another file called '.myfile' and when we use 'ls' again, we can't see it. However, if we use 'ls -a' we can see both files, including the hidden one. That's because the '-a' parameter for 'ls' stands for 'all' and it will show you all files in the directory, including hidden ones.

Did you notice that we also have '.' and '..' in the listing when we use 'ls -a'? Remember, these are the representations of the current directory and the directory one level above the current directory.

You can use the '-a' parameter alongside '-l' like so: 'ls -la' being executed and the resulting files and organised in a list with columns, and showing hidden files too.



You can actually string together multiple parameters using just a single '-', the order doesn't matter.

## Environment Variables

The Linux terminal can store data that can be used by many programs. This data is ephemeral, meaning it doesn't last after you close the terminal window, and this data is all stored in environment variables.

You can print an environment variable using the 'echo' command like so:

```
user@sans:~$ echo $HOME
/home/user
user@sans:~$
```

We can see the 'HOME' environment variable stores the value '/home/user'. Environment variables are denoted by a '$' symbol.

We can also set or modify an environment variable by using 'export':

```
user@sans:~$ export HOME=/tmp
user@sans:~$/home/user$ pwd
/home/user
user@sans:~$/home/user$ cd ~
user@sans:~$ pwd
/tmp
user@sans:~$/home/user$
```

There are a few things that happened here. First of all, we changed the value in the $HOME variable. Notice you don't use a $ symbol when you are using export to set the value.

Now that the terminal believes our home folder is /tmp, the prompt changed. It no longer shows a~ because the terminal no longer believes our home folder is /home/user. To validate that theory, we change the directory to the home folder (~) and use 'pwd' to find where we are. We can see we are in /tmp. Of course, our home folder hasn't changed; the home folder for our user account is still in /home/user, it's just that the terminal believes the home folder is now /tmp.

If we close the terminal window and open a new one, our $HOME environment variable is re-set back to /home/user when the terminal launches. Why? Because every time the terminal opens, it sets up the environment variables as it expects it to be. Remember, the environment variables don't exist when the terminal isn't open; they are created fresh every time.

So what environment variables are there? Well, you can print them all out with the 'printenv' command if you like. Here are mine:

```
user@SANS:~$ printenv
CLUTTER_IM_MODULE=xim
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=40;33;01:or=4
;41:tw=30;42:ow=34;42:st=37;44:ex=01;32:*.tar=01;31:*.tgz=01;31:*.arc=01;31:*.arj=01;31:*.taz=
1;31:*.lzma=01;31:*.tlz=01;31:*.txz=01;31:*.tzo=01;31:*.t7z=01;31:*.zip=01;31:*.z=01;31:*.Z=01
:*.lz=01;31:*.lzo=01;31:*.xz=01;31:*.zst=01;31:*.tzst=01;31:*.bz2=01;31:*.bz=01;31:*.tbz=01;31
*.rpm=01;31:*.jar=01;31:*.war=01;31:*.ear=01;31:*.sar=01;31:*.rar=01;31:*.alz=01;31:*.ace=01;3
:*.rz=01;31:*.cab=01;31:*.jpg=01;35:*.jpeg=01;35:*.mjpg=01;35:*.mjpeg=01;35:*.gif=01;35:*.bmp=
1;35:*.tga=01;35:*.xbm=01;35:*.xpm=01;35:*.tif=01;35:*.tiff=01;35:*.png=01;35:*.svg=01;35:*.sv
v=01;35:*.mpg=01;35:*.mpeg=01;35:*.m2v=01;35:*.mkv=01;35:*.webm=01;35:*.ogm=01;35:*.mp4=01;35:
*.qt=01;35:*.nuv=01;35:*.wmv=01;35:*.asf=01;35:*.rm=01;35:*.rmvb=01;35:*.flc=01;35:*.avi=01;35
.dl=01;35:*.xcf=01;35:*.xwd=01;35:*.yuv=01;35:*.cgm=01;35:*.emf=01;35:*.ogv=01;35:*.ogx=01;35:
.m4a=00;36:*.mid=00;36:*.midi=00;36:*.mka=00;36:*.mp3=00;36:*.mpc=00;36:*.ogg=00;36:*.ra=00;36
:*.spx=00;36:*.xspf=00;36:
LC_MEASUREMENT=en_US.UTF-8
LESSCLOSE=/usr/bin/lesspipe %s %s
LC_PAPER=en_US.UTF-8
LC_MONETARY=en_US.UTF-8
XDG_MENU_PREFIX=gnome-
LANG=en_GB.UTF-8
GDM_LANG=en_GB
MANAGERPID=1876
DISPLAY=:0
QT_STYLE_OVERRIDE=
INVOCATION_ID=72193dc4e5d444f1835af8743ddcffa7
COLORTERM=truecolor
ZEITGEIST_DATA_PATH=/home/user/.local/share/zeitgeist
XDG_VTNR=7
SSH_AUTH_SOCK=/tmp/ssh-G659AuuNCN0L/agent.1896
MANDATORY_PATH=/usr/share/gconf/budgie-desktop.mandatory.path
LC_NAME=en_US.UTF-8
XDG_SESSION_ID=c2
XDG_GREETER_DATA_DIR=/var/lib/lightdm-data/user
USER=user
DESKTOP_SESSION=budgie-desktop
QT4_IM_MODULE=xim
DEFAULTS_PATH=/usr/share/gconf/budgie-desktop.default.path
QT_QPA_PLATFORMTHEME=gtk2
PWD=/home/user
HOME=/home/user
JOURNAL_STREAM=8:24779
SSH_AGENT_PID=1967
QT_ACCESSIBILITY=1
XDG_SESSION_TYPE=x11
TERMINIX_ID=1c9a4e28-a8fe-41a5-8aa0-ea30d33b0b95
XDG_DATA_DIRS=/usr/share/budgie-desktop:/usr/local/share/:/usr/share/:/var/lib/snapd/desktop
XDG_SESSION_DESKTOP=budgie-desktop
LC_ADDRESS=en_US.UTF-8
DBUS_STARTER_ADDRESS=unix:path=/run/user/1000/bus,guid=d22470affd26e52fd768e6e259c4f549
LC_NUMERIC=en_US.UTF-8
GTK_MODULES=gail:atk-bridge
PAPERSIZE=letter
VTE_VERSION=4402
```

There are more, but they are too many for one screenshot to capture.

## PATH

Really, you don't have to worry about **most** of these environment variables. The one that you'll usually need to use most is the $PATH environment variable:

```
user@SANS:~$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
user@SANS:~$
```

Think about what these commands that we've been using are. They are programs, binary executables. The PATH environment variable contains a list of directory paths separated by a colon (:) which tells the terminal where to look for the equivalent binary executable when you type that command. The terminal will look in the first path first, and if it doesn't find the executable in there, it continues down the list checking every directory in the list.

Let's find where the 'ls' program is:


The 'which' command can show us where a program is installed on our system. In this case, it tells us that 'ls' is in '/bin/ls'. We can also see that '/bin' is one of the directories in our PATH, so that is why typing 'ls' on its own in the terminal works.

It's a much better system than having to type the full path to every executable. Imagine if you had to go around typing '/bin/ls' just to list the files and folders in your current directory. Hey, it could be worse: 'ls' could be in '/usr/local/bin' which would make the command you'd have to type: '/usr/local/bin/ls'. No thank you!

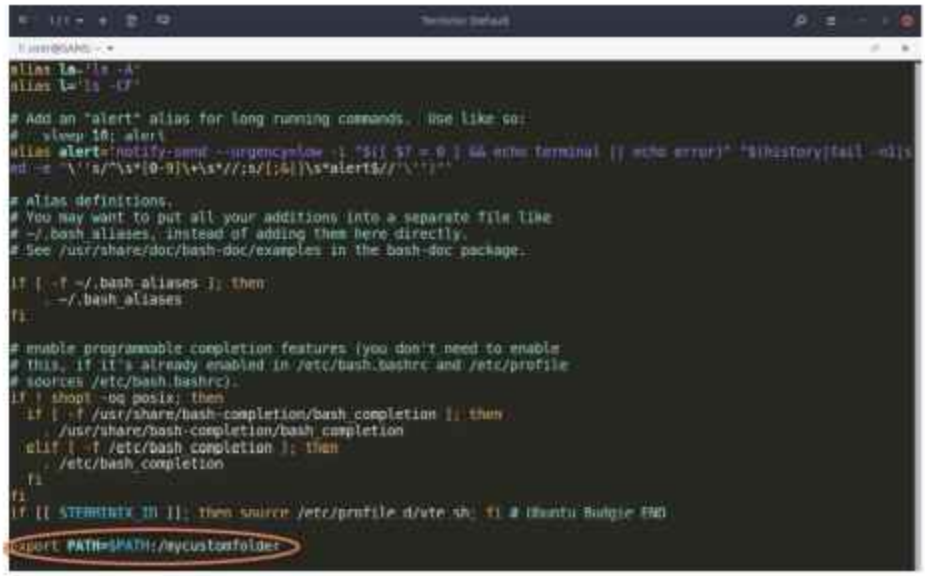
If you install a program on Linux and typing the command isn't working, it's quite likely that executable was installed into a directory which isn't in your PATH environment variable. In this case, you have to either move the executable to a folder in your PATH or update the PATH environment variable to add the directory it was installed in. The problem is, environment variables aren't saved so you'd have to update the PATH environment variable every time you launch a new terminal. There is a better way, though.

If you are running a 'bash' shell, which most people are (you can check with echo $SHELL), you can edit the .bashrc file in your home folder. This is a bash script (essentially just a list of terminal commands), which will get executed every time a new terminal window is opened. Notice it is a hidden file (the . at the start of it).

You can just scroll down the file (it can be rather intimidating, but it's quite easy, promise!) and add this line:

```
export PATH=$PATH:<your custom path here>
```

Like so:




Then simply save the file and close the terminal window. Open a new terminal window and echo $PATH to prove your modification worked:


Why does that work? Well, let's say your normal path is just '/bin', for brevity's sake.

If you do:

```
export PATH=$PATH:/mycustomfolder
```

then for all intents and purposes, you are doing:

```
export PATH=/bin:/mycustomfolder
```

# Linux Navigation

## Contents

In this module, we will be covering:

- Tab Completion
- Previous Commands
- Reverse Command Search
- Bash History
- Parameters
- Interrupts
- Clearing the Terminal

A lot of things that will make your life using the terminal much more comfortable!

## Tab Completion

The modern terminal has a lot of 'quality of life' features designed to make it easier to use. One of the most useful is tab completion. Most modern terminal shells support some form of tab completion.

```
user@sans:~$ ls
Desktop Documents Downloads Music Pictures Public Templates Videos
user@sans:~$
```

Let's say we have these folders in our current directory, and say we wanted to cd into the 'Templates' folder. We could type out:

```
$ cd Templates
```

But we don't have to. Tab completion can do most of the typing for us! Instead, let's try:

```
$ cd T
```

and then pressing tab. The terminal will automatically look in the current folder, and it will know you mean to type 'Templates', because it is the only folder there that starts with a capital 'T'.

This doesn't just work with 'cd'. Tab completion can also fill in the names of terminal commands you are trying to execute, or any files or folders you are trying to access. There is one snag though. Using the directory structure in the image above, what if we wanted to cd into 'Documents' and tried to use tab completion?

You'll find that using:

```
$ cd D
```

and then hitting tab doesn't work. Why? Because we have multiple matches; the terminal doesn't know if we want to go into Desktop, Documents, or Downloads. You'll have to add a few more characters:

```
$ cd Doc
```

and then hitting tab will do the trick in this case.

If you want to list all possible matches so far, you can hit tab twice in quick succession, and it will list all possible matches for you, so you know what the clash is and how to make it more specific. Like so:

```
user@sans:~$ cd D
Desktop/ Documents/ Downloads/
user@sans:~$ cd D
```

Here, we entered:

```
$ cd D
```

and then hit tab twice, and it printed out a list of all possible matches. Again, this also works for terminal commands:

```
user@sans:~$ wh
whatis whereis which while whiptail who whoami whoopsie
user@sans:~$ wh
```

Here, we entered:

```
$ wh
```

but then couldn't remember the exact command. By using double tab, we can see a list of all possible commands that are installed on this system (in the PATH) that start with 'wh'.

## Tab Completion Practice

Being able to navigate the file system and find files and binaries you need very quickly will accelerate your operations significantly. Tab completion is a powerful way to find files where you aren't perfectly sure of the name, or to resolve confiicts. Let's give it a go!

## Previous Commands

Another great feature for usability that you can find in most terminals is the ability to scroll back through previously typed commands. Just use the arrow keys on the keyboard for this. For example, to access the last command you typed, just hit the 'up arrow' key. You can keep pressing it multiple times to go further and further back through your command history. You can also use the 'down arrow' key to come back to your more recently typed commands if you went past what you needed.

You will inevitably come to rely on the previous commands history a lot if you spend any amount of time working in the terminal. Be careful though; it can be tempting to spend thirty seconds scrolling through your command history to avoid typing a command that would take a few seconds to type out. Everyone finds themselves doing that at some point.

## Reverse Command Search

Instead of using the arrow keys to navigate through your command history, you can actually perform a search. I find this is something that not many people know about, but in most terminals pressing CTRL + R in the terminal will bring up a search prompt. You can then enter your search term at the prompt and it will auto-complete with the most recent command that matches the search. For example:

```
(reverse-i-search)`whi': which ls
```

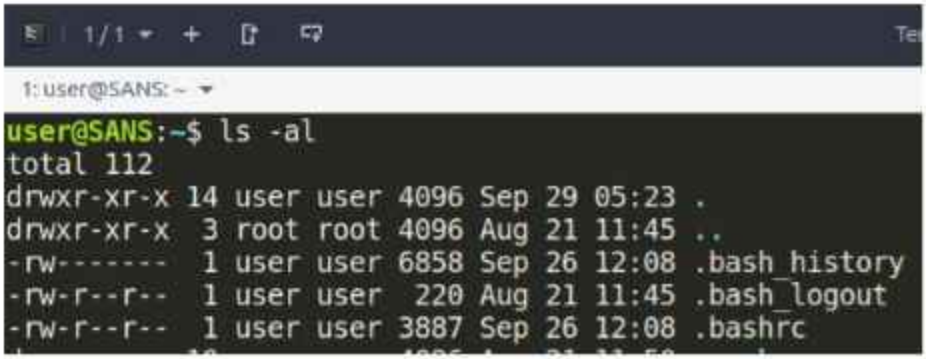
We typed 'whi' at the search, and the search found 'which ls'. At this point, we could hit 'enter' to re-run the command. If this isn't the command we need, and we want to go through other matches further back in the history, we can do that too. Just press CTRL + R again, and every time you do it will go further back in your history repeating the same search:

```
(reverse-i-search)`whi': which print
```

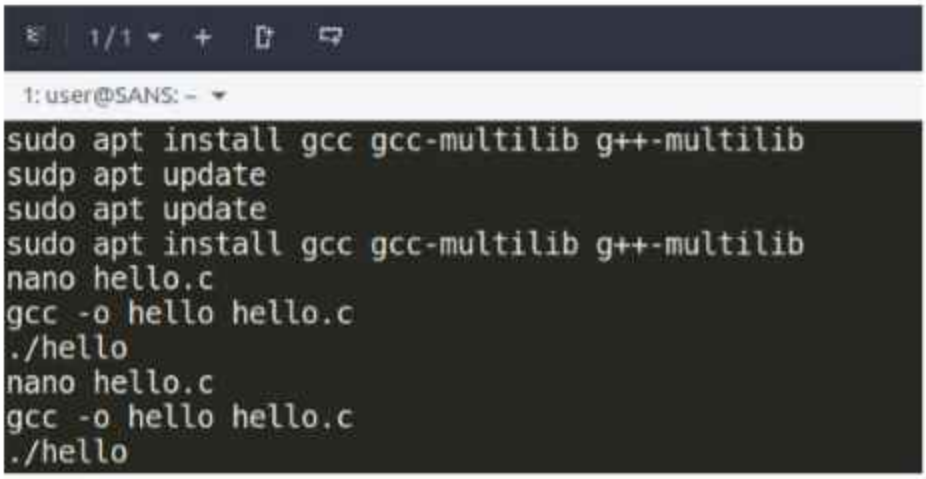
Make sure to use both the arrow keys and reverse command search liberally and you'll find using the terminal can be faster (even significantly faster) than using a GUI.

## History

Since you now know about previous commands and the reverse command search, you may be wondering where all your previously typed commands are stored. Every user on a Linux system has their own 'history' file which keeps track of every command run by that user in the terminal. Every terminal shell has a different name for the history file, but in the BASH shell, the file is stored in the user's home folder, and it's called '.bash_history'. Yep, it's a hidden file.



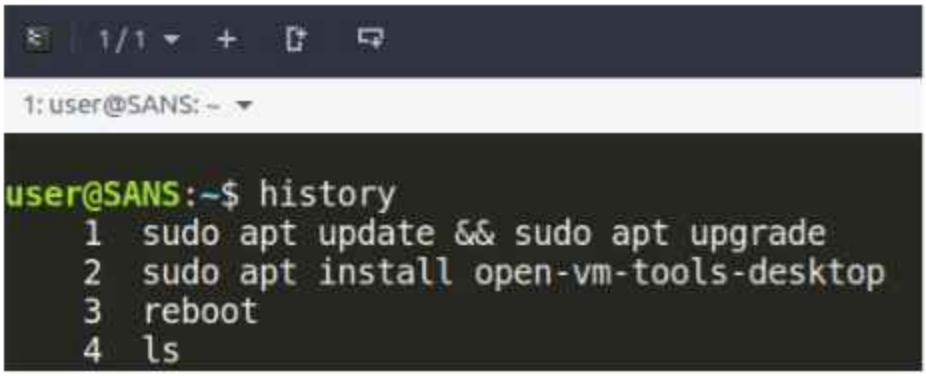

The history file is just a text file, so you can read it with any text editor:




Alternatively, there is a 'history' command that is available, which will let you read the history file. Just do:

```
$ history
```

And you'll get:

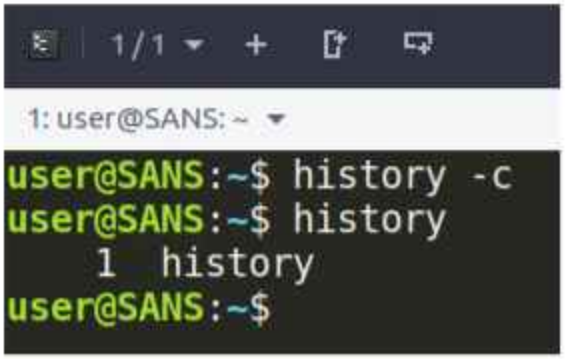



This is a little better because it numbers the commands.

You can also clear the history:

fi. Delete the .bash_history file (a new one will be created the next time you run a command)
fi. Then use the command `$ history -c`

By using 'history -c' you are telling the history program to clear (-c is for clear) the history file that is stored in memory:



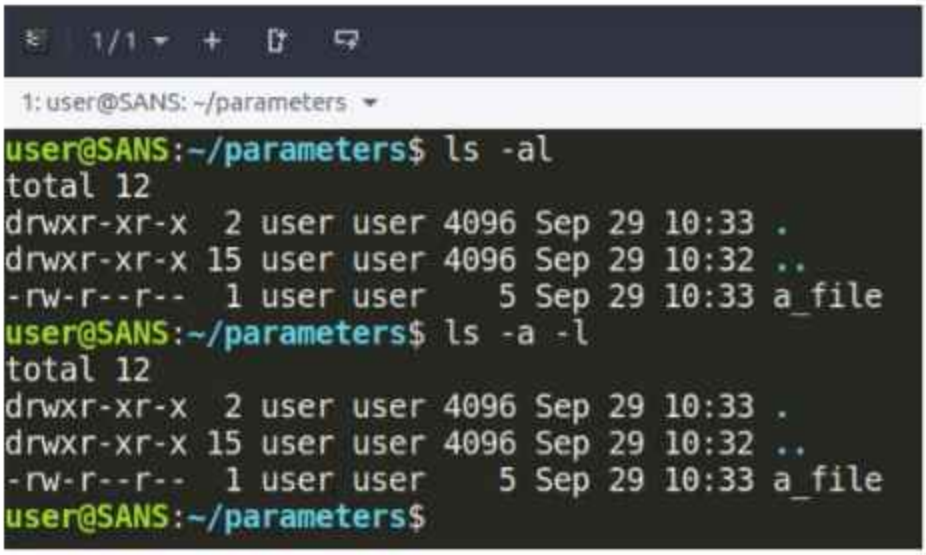

At the end of the terminal session, the file in memory is written to the .bash_history file in the user's home folder. To cover your tracks, you must clear both.

The history file is something that is easy to forget about. That means hackers sometimes get careless and forget to cover their tracks after they've been on a system too, so this is something worth knowing about.

## Parameters

We've already used a few Linux commands so far, and even some that have needed parameters, however, there are several different forms that parameters can take and we need to know all of them.

The first form is single letter parameters. These are like 'ls -a', where the single letter follows a dash (-). If you have multiple parameters, you can chain them together with a single dash, or use one dash for each parameter, like so:



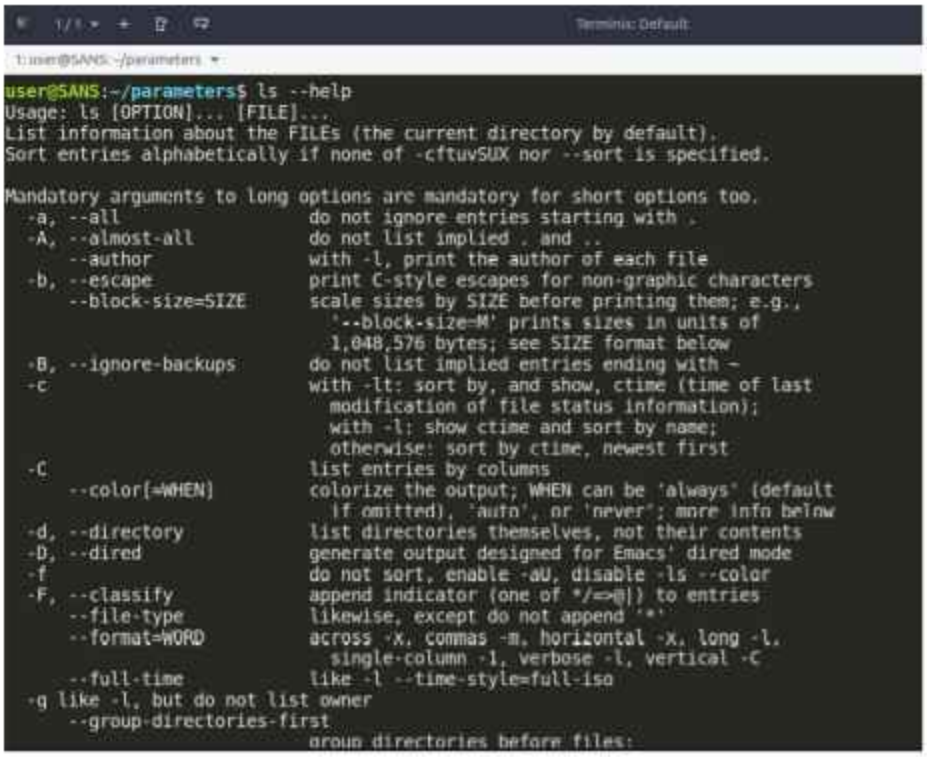

The next form parameters can take is full word parameters. These usually require a double dash (--), such as:

```
user@SANS:~/parameters$ ls –all
. .. a_file
user@SANS:~/parameters$
```

In this case, '--all' is the same as '-a'. Some commands only take a long form or a short form, some take both; it depends on the person who wrote the program in the first place, which can be annoying.

There are even cases where no dash at all is required (or accepted), such as with the 'ps' command, which you'll see later. So, I'm sure the burning question is: how do we know what the parameters are?

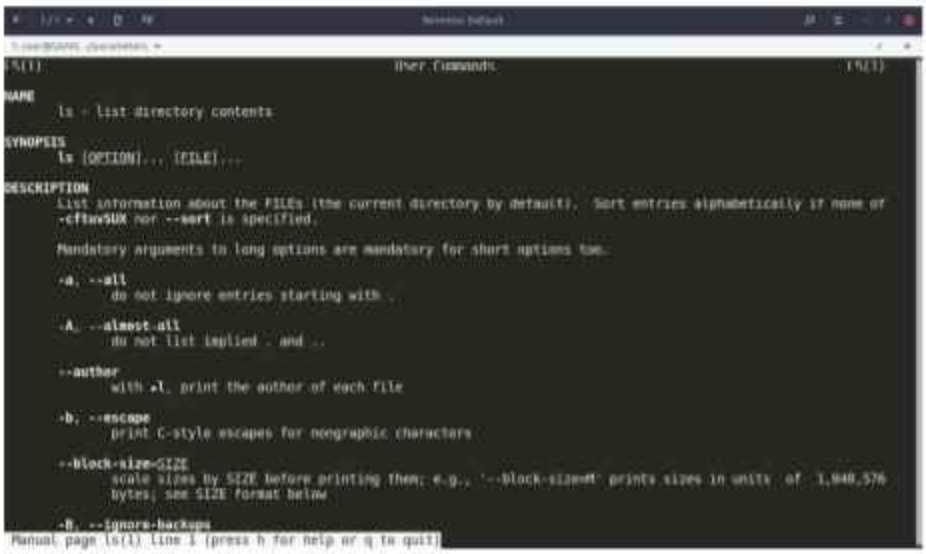
You can use Google to look up the commands, sure, but there is often a faster way. Most commands come with a built-in help page, which is usually accessed with either '-h' or '--help'. For example:




Most tools also come with a manual page which gets installed when the tool is installed. You can view manual pages with the 'man' command. That works like so:

```
$ man ls
```

To pull up the manual page for 'ls':



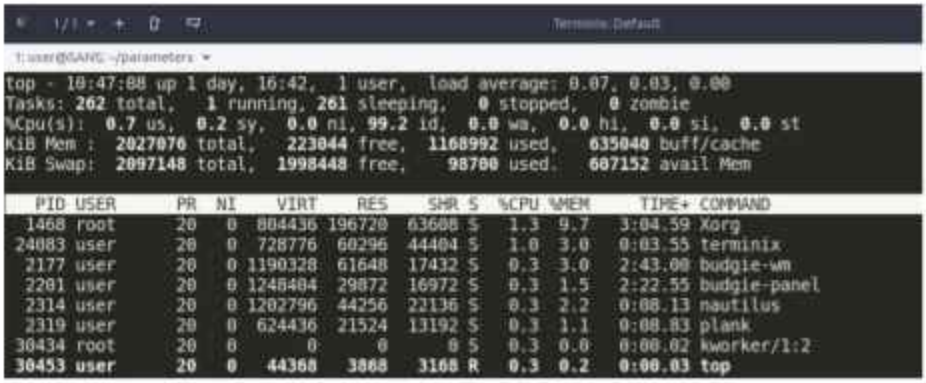

In this case, the manual page for 'ls' is almost identical to the '--help' page, but don't dismiss it! For **most** tools, the help page will give you a limited breakdown of available

parameters, while the man page will give you the full listing. You can scroll around the man page with the arrow keys, and to quit you have to press 'q' on your keyboard, as it tells you.

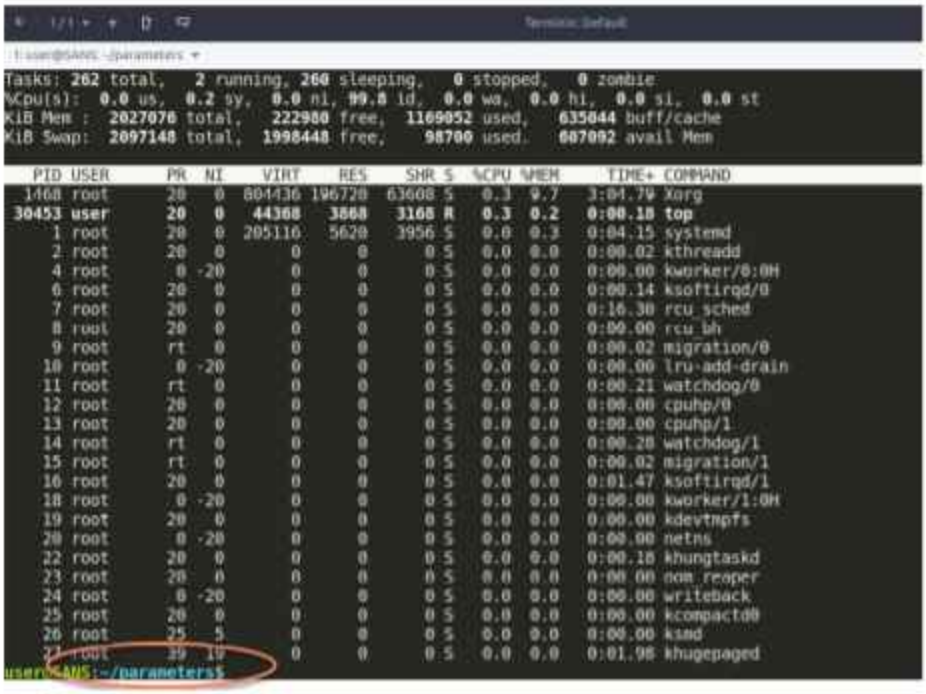
## Interrupts

A lot of Linux commands will run and give you a result right away, but some of them will keep running until you quit them. Some programs, such as 'man', have a specific way of quitting the program. Usually, it will tell you how in those cases, such as press 'q' to quit. If the program doesn't tell you how to quit, there usually isn't a special method; it just uses the method that is there by default. That method is to send an interrupt to the program, which tells it to quit. You can do this by pressing CTRL + C in the terminal.

We'll demonstrate using the 'top' command. 'top' is a command that shows you information about the system you are on, a bit like the task manager in Windows. You can see how busy the CPU is, how much memory is in use, and which processes are running. The view constantly updates with new information. This is what it looks like:




The only way to quit out of 'top' is to use CTRL + C:



You can see at the bottom that we have been dropped to a prompt again, and you'll notice the information above has stopped being updated (well, you would if it wasn't a still image).

This is just one kind of interrupt.

The next kind of interrupt is one you may well stumble upon by accident. A lot of people hit this key combination when they are trying to hit CTRL + C. The next interrupt is to suspend a running program, which pauses the program's execution. You can do that with CTRL + Z:

```
    27 root      39  19       0
    28 root       0 -20       0
[1]+  Stopped                 top
user@SANS:~/parameters$
```

Once the program is suspended, you have a choice in terms of how to proceed. You can resume it, by typing 'fg' which stands for 'foreground'. That will bring 'top' back up and show it to you again. You could also type 'bg' which will allow the process to keep running, except it will run in the background. You will be able to do something else in the terminal while the process continues executing in the background. If you want to bring a backgrounded task back to the foreground, you can just type 'fg'.

If you hit CTRL + Z by accident and you were trying to hit CTRL + C, just type 'fg' and then use CTRL + C to exit.

## Clearing the Terminal

Sometimes we just want to clear the terminal of the output that was on there already, so that we can start afresh. An example of where you might want to do this is after you quit 'top'; the last output of top remains on the screen, which can clutter your window.

There are two ways to clear the text on the terminal; the first is just to type:

```
$ clear
```

There is an even faster way, however. Just hit CTRL + L which has the same function as 'clear'.

# Linux Commands 1

## Contents

In this module, we will be covering:

- cp
- mkdir
- mv
- rm
- cat
- less
- find

## The cp Command

The cp command is short for 'copy'; it allows us to copy a file from one location to another (keeping the original intact).

Take a look:

```
user@SANS:~/cp$ ls
myfile
user@SANS:~/cp$ cp myfile mycopiedfile
user@SANS:~/cp$ ls
mycopiedfile myfile
user@SANS:~/cp$
```

The first parameter is the path to the file you wish to copy. The second parameter is the path where you want to save the copied file to.

You can also specify a full path to copy the file to, for example:

```
$ cp myfile /home/user/Desktop/mycopiedfile
```

You can also copy folders from one place to another, but it doesn't work by default. You have to use the -r or -R parameter. Take a look:

```
user@SANS:~/cp$ ls
myfolder
user@SANS:~/cp$ cp myfolder mycopiedfolder
cp: -r not specified; omitting directory 'myfolder'
user@SANS:~/cp$ cp -r myfolder mycopiedfolder
user@SANS:~/cp$ ls
mycopiedfolder myfolder
user@SANS:~/cp$
```

The -r parameter stands for 'recursive'; basically, it just means 'look inside any folders and copy those files too'. It also copies folders within that folder, so watch out for that if that isn't what you intend. For those times you forget about '-r', the error message shown above is helpful.

Don't forget about cp --help if you want to look at the options. There are a bunch that might be useful, depending on what you need to do, but for the most part cp on its own and cp -r are the only two you'll need 90% of the time.

## mkdir

The 'mkdir' command is used to create a directory (folder). It's easy enough:

```
user@SANS:~/mkdir$ ls
user@SANS:~/mkdir$ mkdir mynewfolder
user@SANS:~/mkdir$ ls
mynewfolder
user@SANS:~/mkdir$
```

You can also use a full path like so:

```
$ mkdir /home/users/mynewfolder
```

You can also make multiple directories at once using the '-p' parameter, which is useful:

```
user@SANS:~/mkdir$ ls
mynewfolder
user@SANS:~/mkdir$ mkdir -p afolder/asecondfolder/athirdfolder
user@SANS:~/mkdir$ ls
afolder mynewfolder
user@SANS:~/mkdir$ ls afolder
asecondfolder
user@SANS:~/mkdir$ ls afolder/asecondfolder
athirdfolder
user@SANS:~/mkdir$
```

And don't forget to use --help to view a more complete listing of parameters, but -p is by far the most useful one.

## The mv Command

The 'mv' command stands for 'move', and it's used for moving files and folders from one location to another. Unlike copy, a move copies the files and folders from one location to another, removing the original.

Take a look:

```
user@SANS:~/mv$ ls
afile firstFolder
user@SANS:~/mv$ mv afile firstFolder/afile
user@SANS:~/mv$ ls firstFolder/
afile
user@SANS:~/mv$ ls
firstFolder
user@SANS:~/mv$
```

You can see here we moved 'afile' into 'firstFolder' and there is no longer 'afile' in the current directory. Unlike the 'cp' command, you can move folders without having to use any parameters.

There are some useful parameters for 'mv', however, such as -n for no-clobber which prevents the move from overwriting a file that already exists at that location. Or -u (update), which only overwrites a file if the timestamp on the file you are moving is newer. Check out --help for more information.

The 'mv' command is also commonly used for renaming files. There is no rename command, so you have to use 'mv' to move a file to the same location but with a new name. Take a look:

```
user@SANS:~/mv$ ls
afile
user@SANS:~/mv$ mv afile arenamedfile
user@SANS:~/mv$ ls
arenamedfile
user@SANS:~/mv$
```

## rm

The 'rm' command stands for 'remove' (actually it's the initials of the author, but it's easier to remember it as remove). It is used for deleting files and folders. Take a look:

```
user@SANS:~/rm$ ls
afile
user@SANS:~/rm$ rm afile
user@SANS:~/rm$ ls
user@SANS:~/rm$
```

That's all it takes to delete a file. Did you notice, it never asked if you were sure about deleting the file? Yep, this is Linux after all. You are allowed to delete the file, so it just assumed you knew what you were doing. If you made a typo and deleted the wrong file? Tough luck, it's gone. Be particularly careful if you are using 'rm' as root because you could delete everything on the file system and corrupt the operating system.

Once again, you can delete folders with 'rm', but like with 'cp' you need to provide the '-r' parameter for recursive:

```
user@SANS:~/rm$ rm afolder
rm: cannot remove 'afolder': Is a directory
user@SANS:~/rm$ rm -r afolder
user@SANS:~/rm$ ls
user@SANS:~/rm$
```

And don't forget to look at --help for more options.

## cat

The 'cat' command stands for 'concatenate'. It's used for joining the contents of several files together and then printing the result to the terminal.

See:

```
user@SANS:~/cat$ ls
file firstFile secondFile
user@SANS:~/cat$ cat secondFile firstFile
The only things on youtube are
kittens...
user@SANS:~/cat$
```

Most commonly it is used on just a single file to print the contents to the screen quickly. For example:

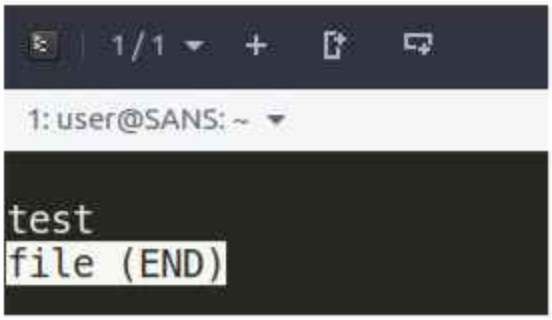
```
user@SANS:~/cat$ ls
file
user@SANS:~/cat$ cat file
kittens...
user@SANS:~/cat$
```

But you can give it as many files as you want to concatenate. Again, look at --help for a more complete list of options.

## less

The 'less' command is used for reading long files. It will open the file in an interactive program where you can use the arrow keys to scroll through the file. This program cannot be quit using CTRL + C as usual; instead, you must hit the 'q' key to exit from it.

Take a look:




At this point, hitting 'q' will exit the reading of the file.

There is an older version of the 'less' command which was called 'more', which is sometimes installed on older systems. You can generally use either. The geeky joke is, 'less is better than more', which should tell you which to use if you have both installed.

The good part about less is that it doesn't load the whole file into memory at once, it only loads a portion of the file at a time. This is good if you want to read a text file that is several gigabytes in size, because if you try to open that in a normal text editor the whole file will try to get loaded into memory, and if you don't have enough RAM your text editor will crash. With 'less', however, you can read a text file of any size, no matter how large.

## The find Command

The 'find' command is used to find files on a system. There are a lot of options with this command, too many to cover them all here. However the most commonly used method is to find a file by filename, so we'll show you this one:

```
user@SANS:~/find$ ls
onefile threefiles twofiles
user@SANS:~/find$ find . -name onefile
./onefile
user@SANS:~/find$ find . -name "*files"
./threefiles
./twofiles
user@SANS:~/find$ find . -name "*file*"
./threefiles
./onefile
./twofiles
```

The find command's first parameter is the directory you want to search in (it will include all subfolders). If you want to search the whole file system, you can direct it to search in '/', or if you want to search in the current folder and all subfolders you can direct it to search in '.' (remember, this is a shortcut for the current directory).

The next parameter is the search method; in our example, we are searching by filename. The third parameter is the search term. If we know the full name of the file, we can just throw it in as we did with "onefile".

If you want to match based on a partial name, you need to use the wildcard character, which is the asterisk ". The wildcard represents an unknown part of the name, so in our second search we searched for '*files' which would find any files with a name that ends in 'files'. In our third example, we went one step further and searched for any files with 'file' somewhere in the filename.

The find command has a huge amount of options. You can also search by date the file was created, or the owner of the file, or by which files have what permissions. The combinations are endless, which is why we won't be covering them here. Do find out more in the 'find' man page (remember, 'man find').

# Linux Commands 2

## Contents

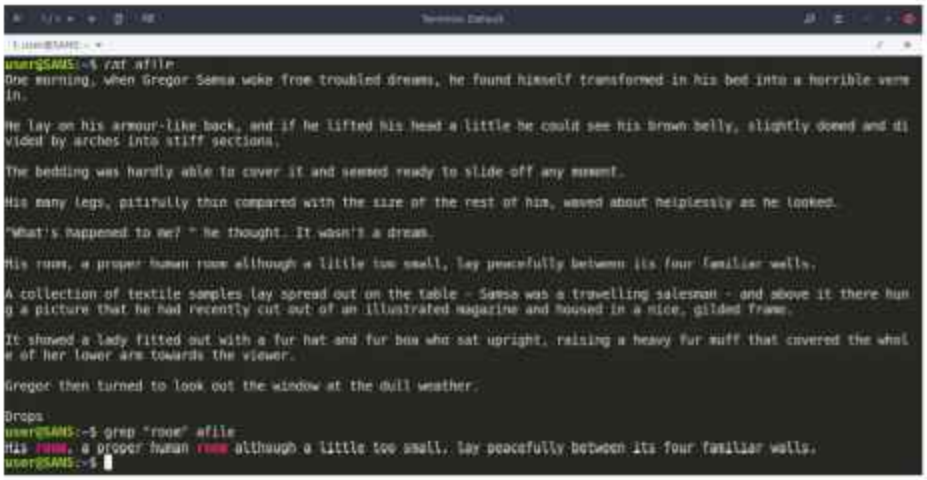
In this module, we will be covering:

- grep
- which
- apropos
- nano
- vim
- file
- strings
- wget

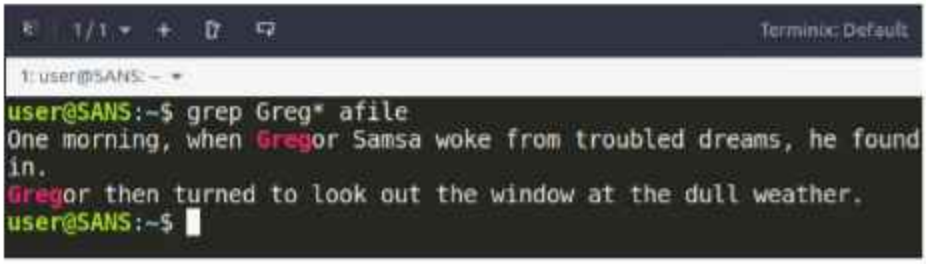
These are some of the most useful and powerful commands on your Linux system for editing files, downloading data and even getting in to security use cases. These commands will teach you how to use more powerful commands, but also will be invaluable every day in your career!

## The grep Command

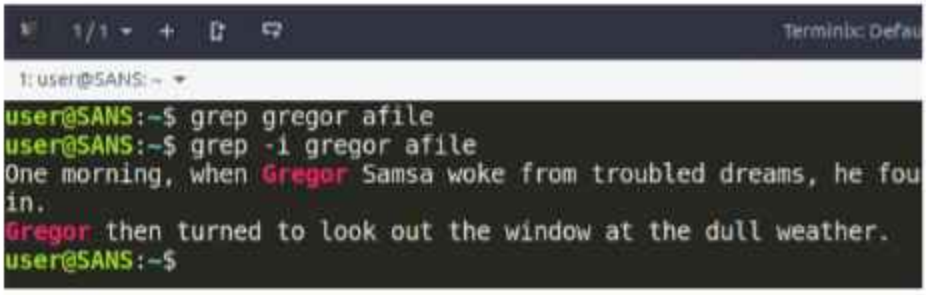
The 'grep' command is used for searching for text within files. Take a look:




Here we have a block of sample text, and with our grep command, we found the line that contains the word 'room'. The first parameter to grep is the search term, and the second is the file to search in. Similar to the 'find' command, you can use a wildcard (*) operator like so:




Keep in mind that, by default, grep is case-sensitive. You can make it case insensitive using the '-i' parameter:




Don't forget to look at more options with 'grep --help'.

## which

The 'which' command can show you where in your PATH a tool is installed. For example:

```
user@SANS:~$ which ls
/bin/ls
user@SANS:~$
```

If you try to run which on a command which doesn't exist in your PATH, then you will get no results. This is a good indication that you need to either:

- Move the program you installed into a folder in your PATH

or

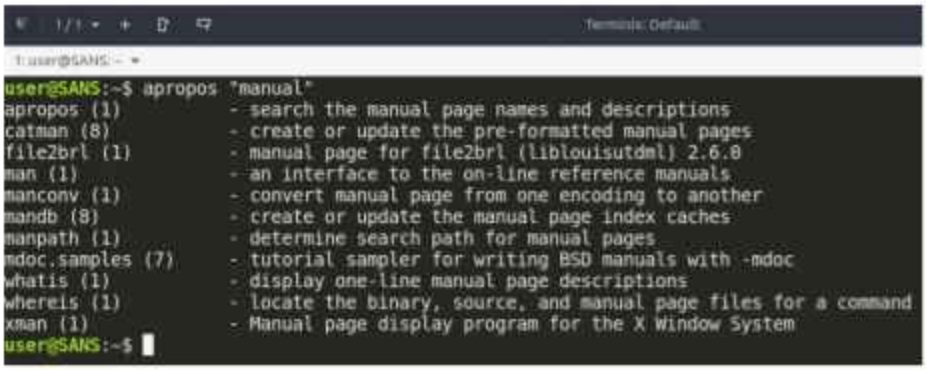
- Add the folder that the program was installed into to your PATH.

```
user@SANS:~$ which foo
user@SANS:~$
```

## apropos

You already know about the 'man' command. This is the command used to pull up a manual page on tools installed on your Linux system. Similar to 'man' is 'apropos'. The 'apropos' tool is used to search man pages for keywords, usually to find the 'appropriate' tool to use in a particular situation.

For example, 'which tool could I use to display the manual for a tool?":




We searched all the available man pages for 'manual', and got the results above. One of them is 'man', which is the tool you want to use to view man pages. You could then do:

```
$ man man
```

...to view the manual page of the man program. The 'apropos' command is used to find the 'appropriate' tool for a particular job. It's particularly useful when you don't have access to the internet for some reason, such as if you are working on a long fiight.

## nano

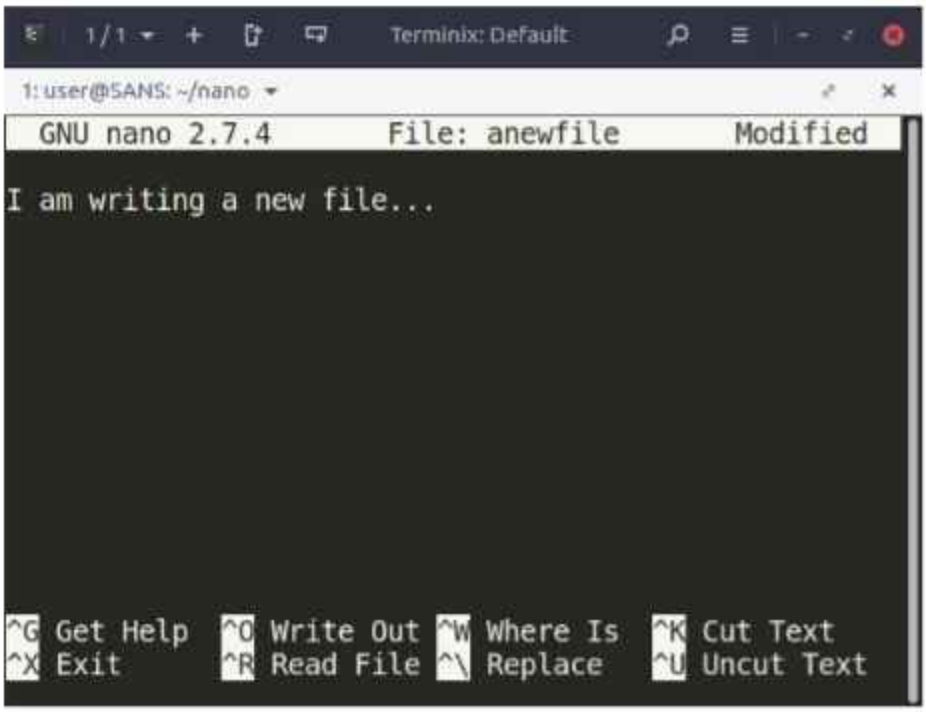
The 'nano' tool is a command-line based text editor. It is installed on most modern Linux distributions by default, and it's nice and easy to use.

To edit a file, you just run 'nano' on the filename that exists.

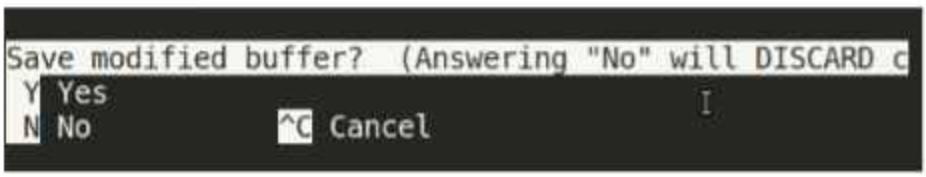
To create a new file, you just run nano on a filename that doesn't exist, like so:

```
$ nano afile
```

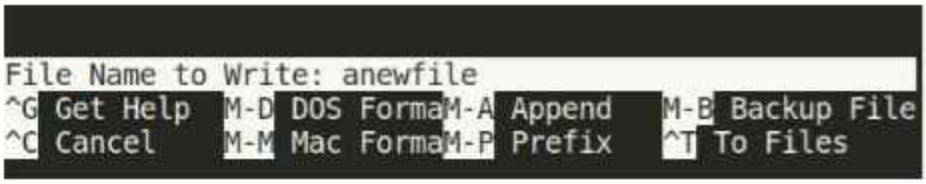
It even gives you a nice prompt to show you how to search and exit, amongst other options:




And you can see here, to exit we just hit CTRL + X. If we have made changes, we will be prompted to save and then put in the filename to save as. The default option is to overwrite the existing file:




then:



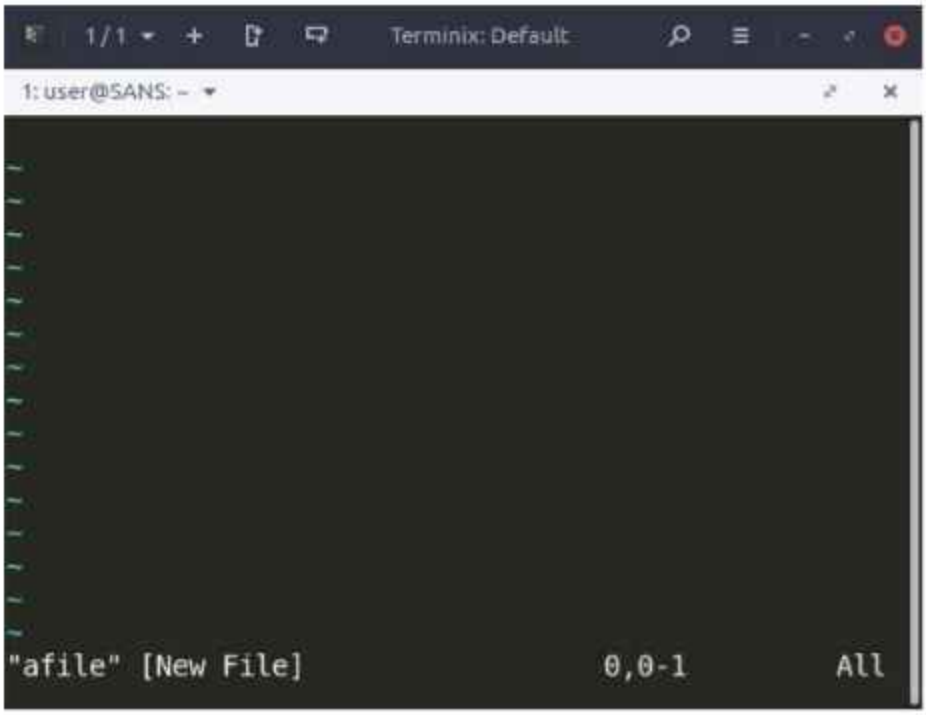

It's a really nice and easy text editor. You'll see why we're emphasising it's ease of use when we get to the next text editor, Vim.

## vim

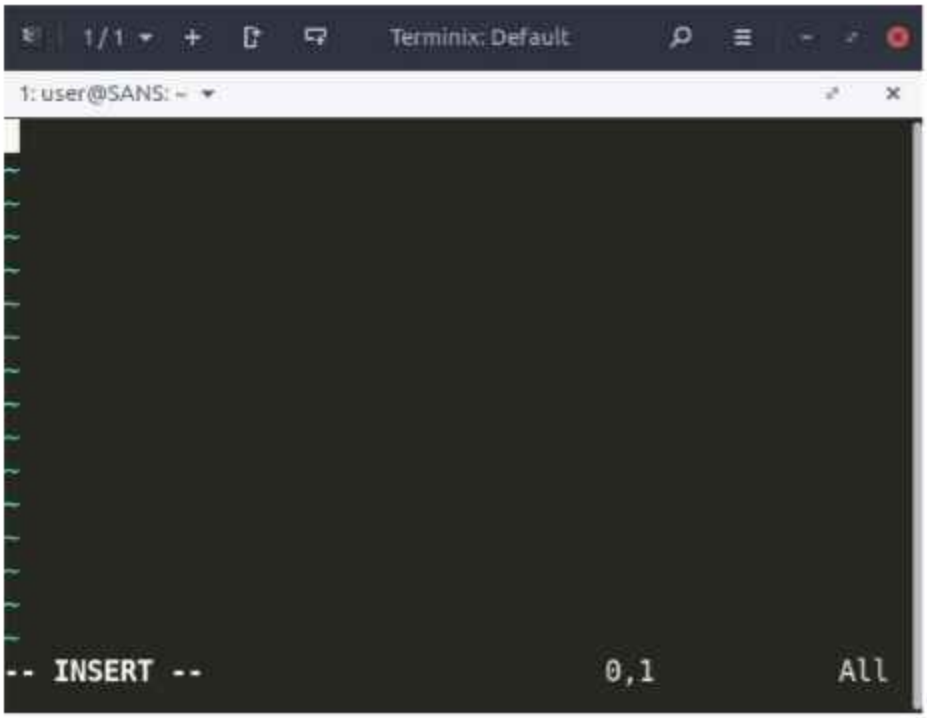
The 'vim' command is another text editor, similar to 'nano' except it is a lot more powerful. The downside is, it is also a lot harder to use than 'nano'.

vim is actually a modern version of the original program 'vi', which you might end up with on some systems. Usually, on newer systems, typing 'vi' will get you 'vim', but sometimes you might still have the older version. We'll be talking specifically about 'vim' here and not its predecessor.

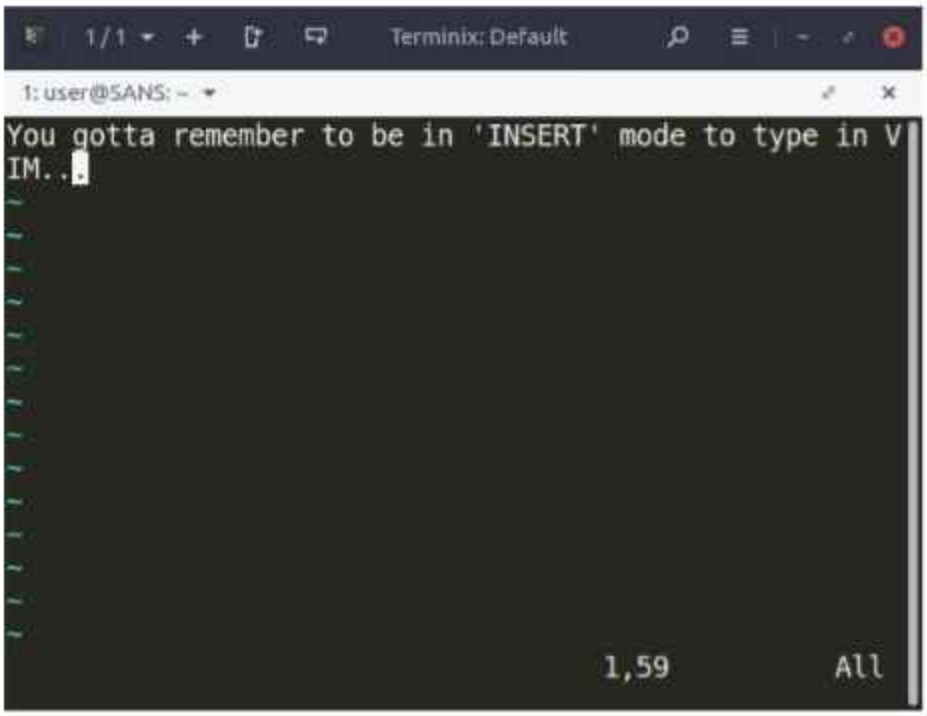
Similar to 'nano', if you launch 'vim' and pass it a file name, it will open that file, or create it if it doesn't already exist.




Here we have 'vim' when we ran it on a file that doesn't exist. It has created 'afile', and it's telling us it is a 'New File'. At this point, you might be expecting to just start typing, but first you need to enter into 'insert' mode by hitting 'i' on your keyboard:




You can see here we are now in 'INSERT' mode and we can begin adding text to our file. Once we're finished adding text to the file, we have to exit from INSERT mode by hitting CTRL + C. This will dump us back to the default mode:




From here, we have a few possibilities. If you want to quit, then you just type:

```
:q
```

and then press return. However, if you have unsaved changes, it will not quit and will instead throw an error. If you want to force it to quit even though you have unsaved changes, you have to type:

```
:q!
```

If you want to save and quit you can either do:

```
:wq
```

or:

```
:x
```

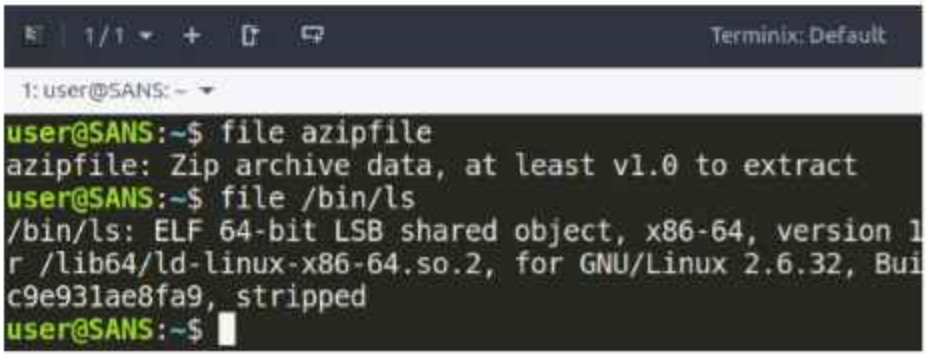
You may have guessed that 'q' stands for quit, and 'wq' stands for write-quit. 'x' is just a shortcut for 'wq', in case you prefer to type a single character instead of two (what amazing efficiency savings!).

There are a lot of commands in vim, which you can go ahead and learn if you want. Although vim may seem complicated (and we don't really have time even to scratch the surface of it in this course), once you know the commands you can fly around text files much faster than with 'nano'. It's also worth being familiar with both 'vim' and 'vi' because you don't always get a choice as to what is installed on a system. Particularly in the world of work, where you may end up with access to a system where you can't install 'nano' just because it's the only text editor you know how to use.

## file

The 'file' command can tell us the filetype of a file. You may have noticed that Linux isn't keen on using file extensions, instead what matters is the contents of the file. Specifically, every filetype has its own file header which is something like a signature identifying it. The file header is universal, even files created on Windows use them, it's just that Linux uses them to tell the type of file, while Windows relies more on file extensions (which, let's face it, are basically just part of the file name).

The way the file command works is by reading the file header of the file and comparing it against a database of file headers to tell you what type of file something is. Take a look:




Here we ran 'file' on two files. The first file is a zip file, which we removed the file extension from. The 'file' command accurately tells us that it is a zip file.

The second time we ran it on the 'ls' command, which is a binary executable file, and 'file' tells us that it is an 'ELF' file (which is the Linux version of an EXE on Windows: an executable file).

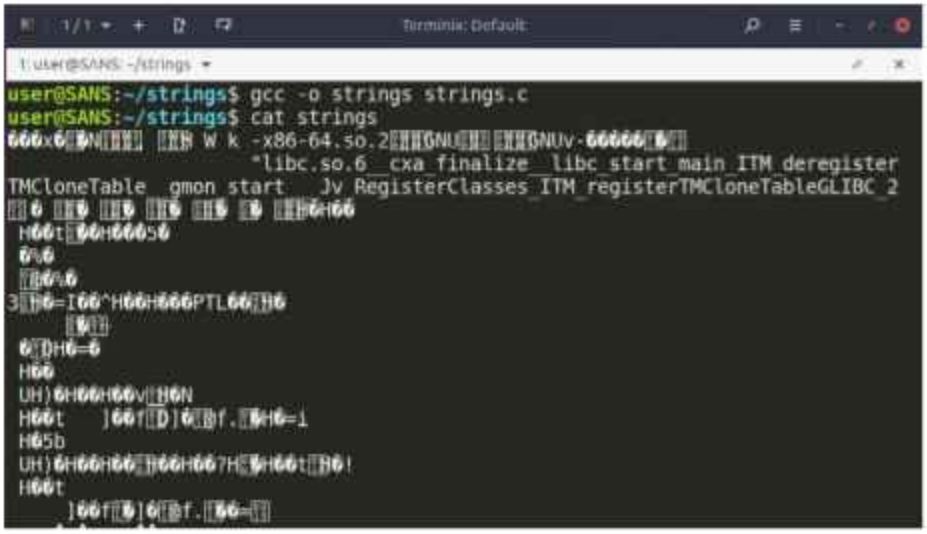
Altogether, this is a very useful command and one you should use often.

## The strings Command

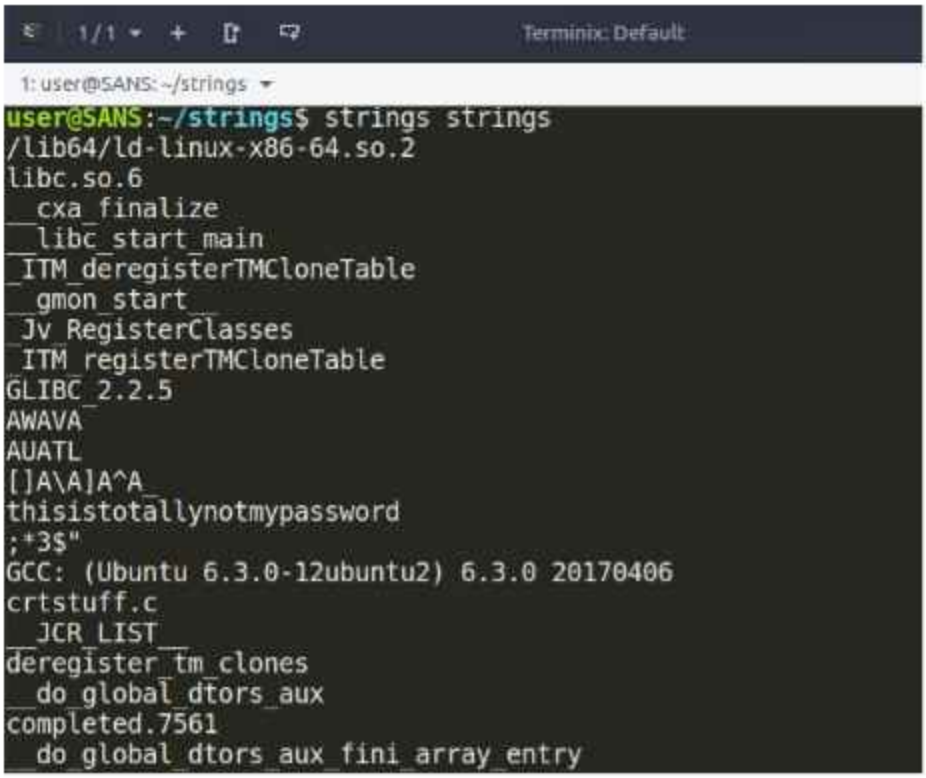
The 'strings' command is used for pulling text sequences out of binary files. To demonstrate, here is a C program that doesn't do anything:

```
int main() {
   char *password = "thisistotallynotmypassword";
   return(0);
}
```

Here we have the password stored in the program. If we compile the program and print it with 'cat', we get a bunch of junk alongside some valid text:




Although there are some valid ASCII characters here, a lot of it is junk. If we use 'strings' on the same file, however:




We only get sequences of ASCII characters that are 3 or more characters long. This is far more readable, and you can even see the password that was stored in the program.

You may want to check out the man page of the strings command for more options, including encoding and length of search options. It is very powerful and frequently used in cyber security.

## wget

The 'wget' command stands for 'web get'. It allows us to download files from the internet from the command line. Take a look:

```
user@SANS:~/wget$ wget https://www.google.com
--2017-10-04 10:57:09--  https://www.google.com/
Resolving www.google.com (www.google.com)... 216.58.198.228
Connecting to www.google.com (www.google.com)|216.58.198.228|:443... connecte
d.
HTTP request sent, awaiting response... 302 Found
Location: https://www.google.co.uk/?gfe_rd=cr&dcr=0&ei=9yDVWYTQO8PW8gee_afwDQ
 [following]
--2017-10-04 10:57:11--  https://www.google.co.uk/?gfe_rd=cr&dcr=0&ei=9yDVWYT
QO8PW8gee_afwDQ
Resolving www.google.co.uk (www.google.co.uk)... 216.58.214.3
Connecting to www.google.co.uk (www.google.co.uk)|216.58.214.3|:443... connec
ted.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [text/html]
Saving to: 'index.html'

index.html              [ <=>                ]  11.17K  --.-KB/s    in 0s

2017-10-04 10:57:13 (28.0 MB/s) - 'index.html' saved [11440]

user@SANS:~/wget$
```

And the result is:

It looks like Google's homepage, but take a look at the URL. It's actually the file we just downloaded.

## Chaining Commands

This module is not huge, but it provides a very useful concept for us in our future scripting. Chaining commands and basic scripting. Let us take a look at an example:

```
ls; sleep 5; ls
```

This little snippet lists files, waits 5 seconds and then runs ls again. We can chain commands together to run one after the other on one line.

We can also use pipes and redirects if we want to. For example:

```
cat /etc/passwd | grep root; sleep 2; ps aux | grep root
```

This command will search /etc/passwd for details of the root user, and then sleep for 2 seconds, before listing running processes that reference root.

## Chaining Commands Demo

In this section we walk through an example of how to chain commands together. We will use some commands that you may not yet be intimately familiar with, but the ability to pass information between commands and build up your desired capabilities is the real point.

Firstly we will use the `ps aux` command. We then layer this together with grep to filter for a specific set of processes that match the name `smbd`. This is done as follows:

```
ps aux | grep smbd
```

This produces a list of the corresponding processes, but unfortunately also matches the grep itself! We want to remove this from the list as a false match, which we can do with another grep and the negation option!

```
ps aux | grep smbd | grep -v grep
```

Now we want to filter for the PID field, which is the second field (defined by tabs between data by default). We can use `awk` to do this. Please note that `awk` has a remarkable set of capabilities - it is a whole language designed to filter and manipulate data. Adding `awk` we now have:

```
ps aux | grep smbd | grep -v grep | awk '{print $2}'
```

This gives us a list of PIDs that we can now process, using the xargs command. Xargs enables us to take items delimited by spaces, other characters or returns. The default handles new lines which is exactly the format of our data! It will pass the command as an argument to any command that we specify.

```
ps aux | grep smbd | grep -v grep | awk '{print $2}' | kill -9
```

The `kill -9` which hard terminates a process will be executed for each of the lines, terminating the processes one by one. Running one of our earlier `ps aux | grep smbd` commands against will demonstrate these have been terminated.

The power of this system is that each command you learn, such as awk, python, tee, cat, grep (and many more), can be tied together to form new combinations. You can use each command as a part of the jigsaw puzzle you assemble to solve your specific issue. It takes

practice, but this is extremely powerful and will supercharge the velocity with which you complete tasks in the future.

# Linux Architecture and Components

## Contents

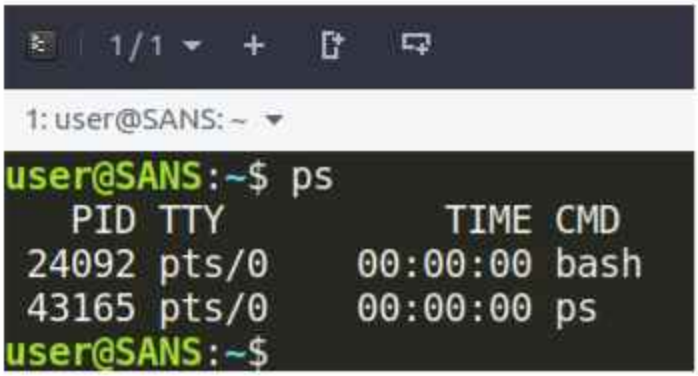
In this module, we will be covering:

- Processes
- Pipes
- Redirects
- The passwd file
- Scheduled tasks
- Package managers
- Packages
- Building a program from source code
- SSH

These skills will set you up as a power user of Linux with practice!

## Processes

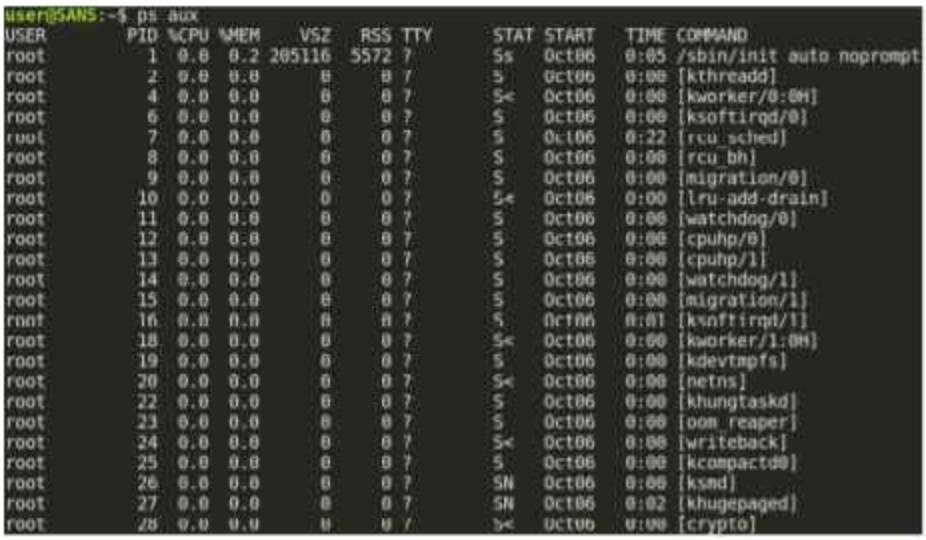
A process is just a running program of some kind. You previously saw one method of viewing running processes by using 'top', however top is not a complete list of all processes running on the system, just the ones that are using the most resources. To see a full list, we can use the 'ps' command:




By default, 'ps' will only show you processes running under your current user. If you want to view a more complete list, you'll have to add some parameters. There are two ways to pass parameters to the 'ps' command: Linux syntax or BSD syntax. Which you use is up to you, but most people end up using BSD syntax.

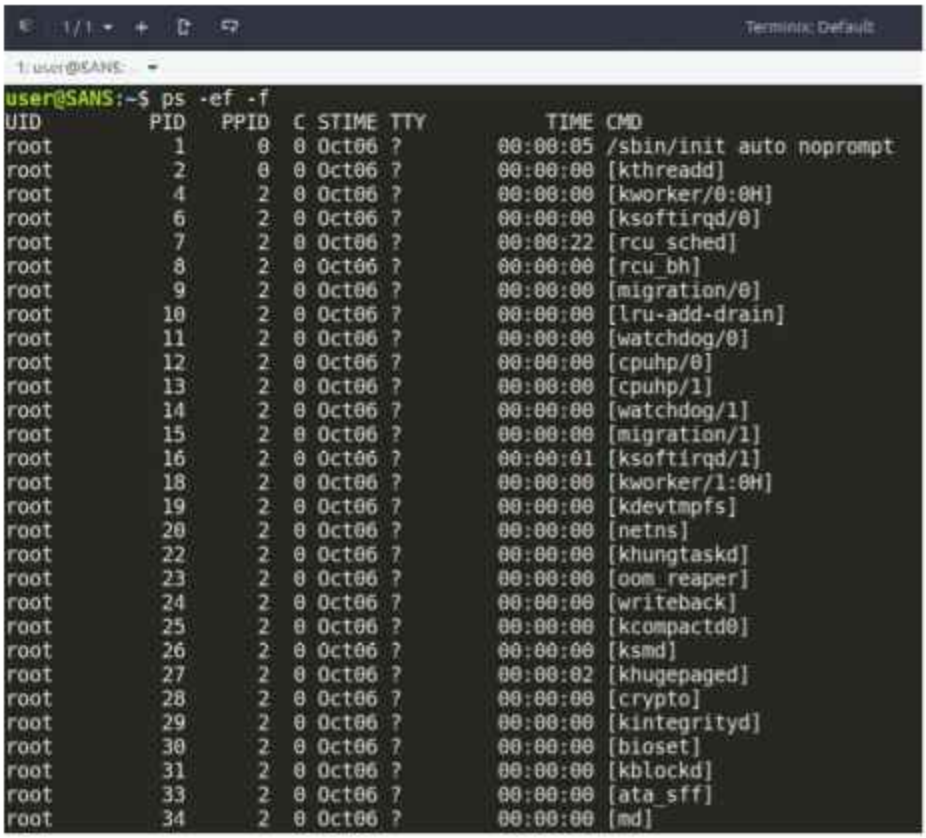
To see a full list of processes running on the system:

BSD Syntax:




Notice the parameters don't have a dash ('-') as a prefix? This is BSD syntax.

The equivalent in Linux syntax would be:

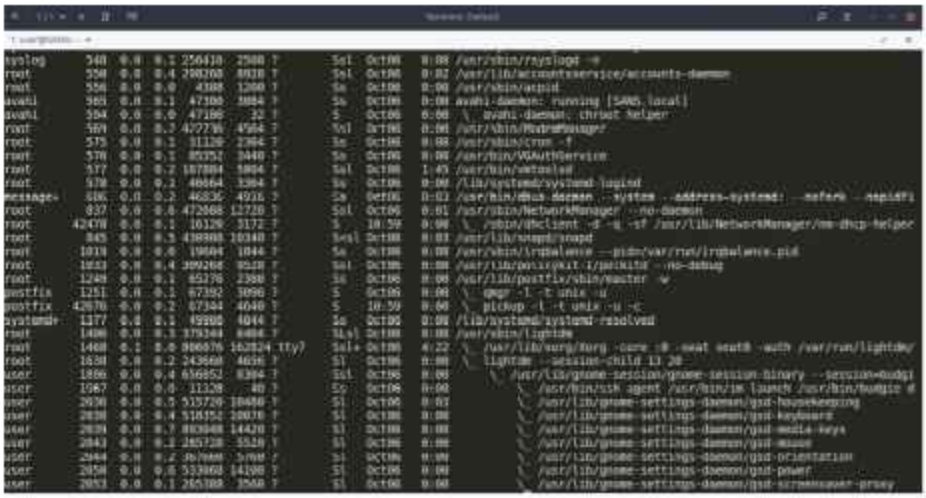



Another thing you might find useful is to get a full list of processes along with subprocesses and more detail on the parameters the processes were launched with. In BSD syntax:

```
$ ps auxf
```

Notice the extra 'f' at the end?

The result looks like this:



The number of options in the 'ps' command is extensive, so be sure to look them up for yourselves to find which ones you prefer to use. My go-to is:

```
$ ps aux
```

and if I need more information:

```
$ ps auxf
```

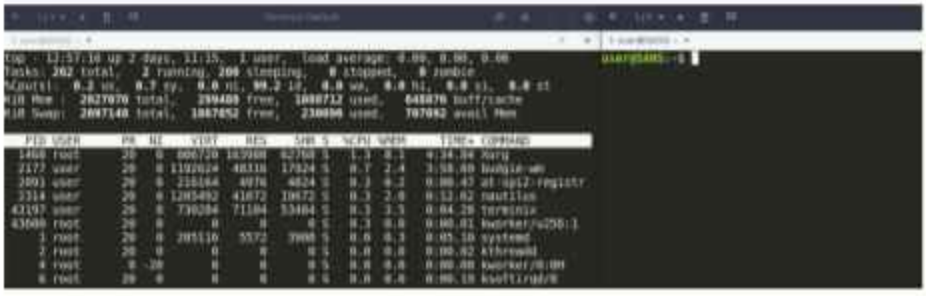
It can be useful to pick your own go-to and memorise the command, if you need slightly different behaviour from time to time, you can still look at the man page.

### Killing It

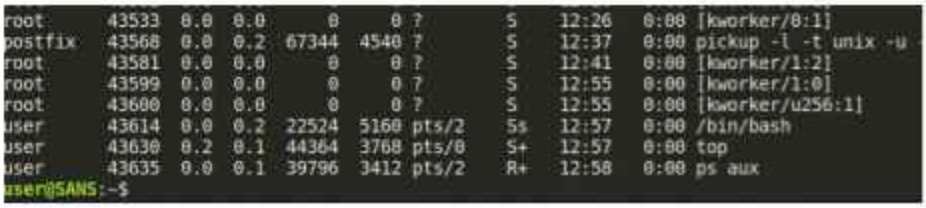
Now that you know how to find the running process, let's show you how to kill one.

We'll open two terminal windows. In one we'll run 'top' and in the other terminal window we'll kill the top process and watch it exit.

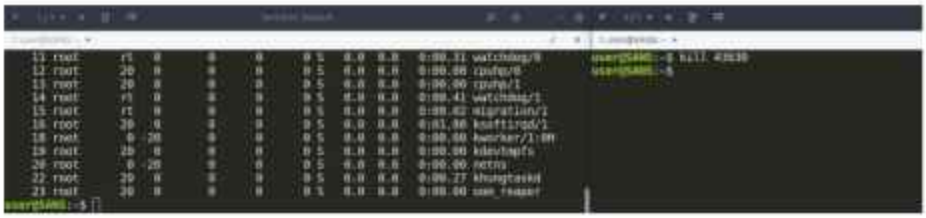
First, we'll run 'top':



Now to find the 'top' process in ps:




Notice 'top' is the second one from the bottom? There is a number in the second column from the left, 43630. This is the process ID, also known as the PID. We can use this number to kill the process with the 'kill' command, like so:



Notice how 'top' in the first window exits and dumps us back to the command prompt after we run the 'kill' command against the top process' PID?

Sometimes you may try to kill a process, and nothing will happen. If that is the case, the process may be unresponsive. You can force-quit a process using:

```
$ kill -9 <<PID>>
```

where <> is your PID number.

## Pipes & Redirects

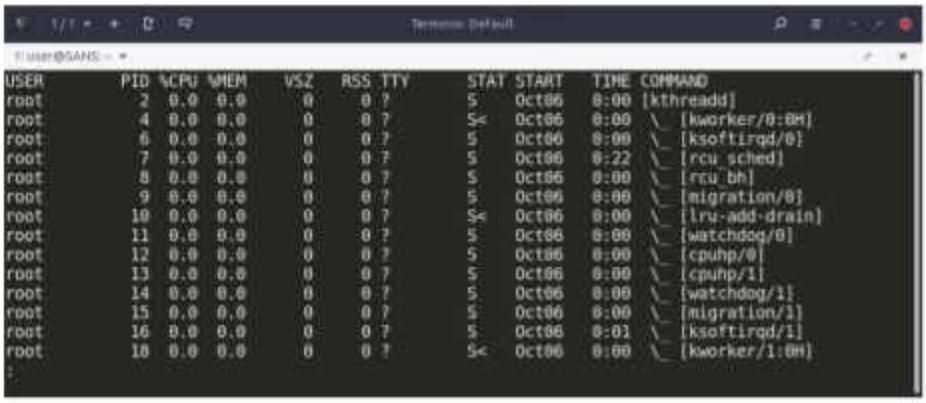
### Pipes

The Linux terminal offers us an excellent way of combining two or more programs to achieve a complicated task that couldn't be done with just one program alone. This is the pipe '|' character, which can be used to send the output of one command and feed it in as the input to another command. In this way, we can chain commands together.

Since we just learned about the 'ps' command, let's use that in our example. The 'ps' command can produce a long output, so what about combining it with 'less' so we can scroll through the output at our leisure?

```
$ ps auxf | less
```

This will run the 'ps auxf' command and then take the output it would produce and use that as input to the 'less' command. Running it, we get:

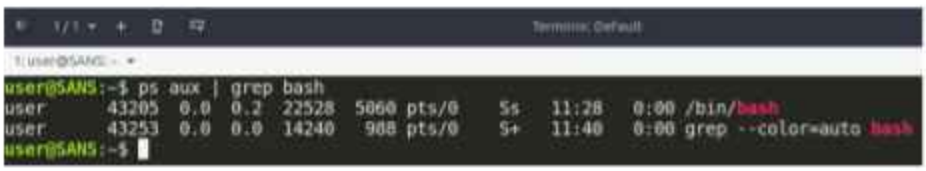



You can see we are at the start of the output and we can use the arrow keys to scroll through. It's very convenient for working with long command output.

What about if you want to search for a particular process? Let's combine 'ps' with 'grep':

```
$ ps aux | grep bash
```

And that produces:



Notice the second instance is our grep command running to search for 'bash'? We can safely ignore that, or if you don't want it to appear in your output at all:

```
$ ps aux | grep bash | grep -v grep
```


In this case, take the output of 'ps aux', feed it into grep to produce a list of matches, then feed that list of matches into grep again to remove any lines containing 'grep'.

We can keep going, if you like. How about opening the result of all of that in 'less'?

```
$ ps aux | grep bash | grep -v grep | less
```

The possibilities are endless! It's extremely useful, and you'll want to get familiar with this.

### Redirects

Redirects take the output of a command and write it to a file. Similar to a pipe, but think output to a file instead of to another program.

There are two forms of redirects: the first is overwrite, which uses a single 'greater-than' sign ('>'). The overwrite will create a file for the output if it doesn't already exist, but if a file does already exist it will overwrite it entirely:

```
user@SANS:~/redirect$ ls
user@SANS:~/redirect$ echo "hello one!" > afile
user@SANS:~/redirect$ ls
afile
user@SANS:~/redirect$ cat afile
hello one!
user@SANS:~/redirect$ echo "hello two!" > afile
user@SANS:~/redirect$ cat afile
hello two!
user@SANS:~/redirect$
```

You can see the first time we ran the command the file didn't exist, so it was created. The second time we ran the command, the contents of the file were replaced completely. Be

careful using the single redirect! It is very easy to overwrite the contents of an important file!

The second type of redirect is the append or double redirect, which uses two 'greater-than' signs ('>>'). The append will create the file if it doesn't exist, but if the file does already exist it will add the output to the end of the file instead:

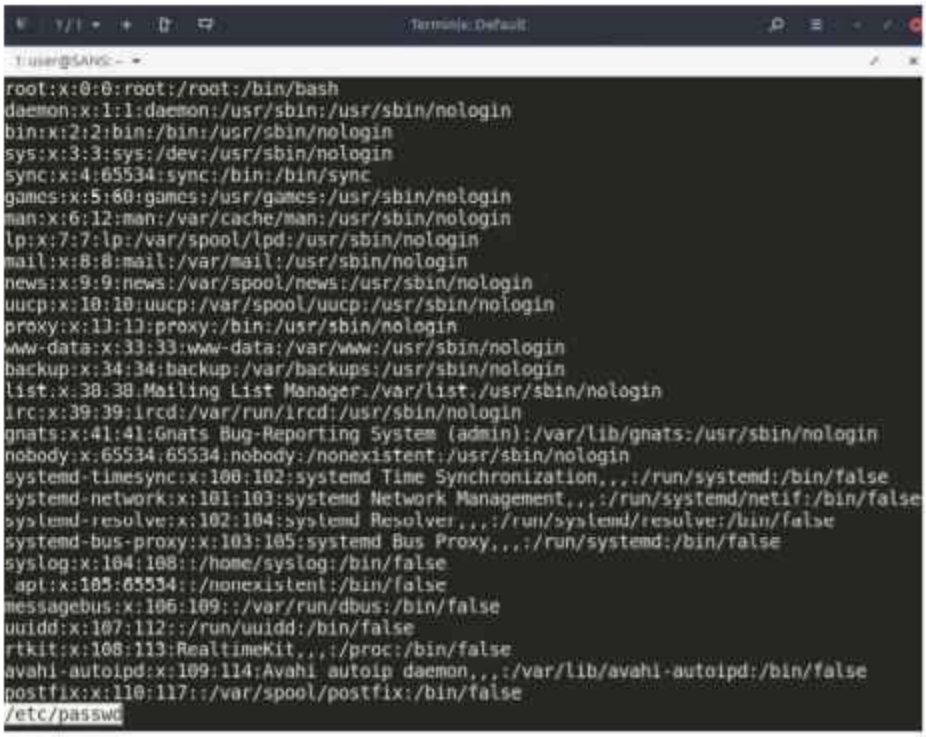
```
user@SANS:~/redirect$ ls
user@SANS:~/redirect$ echo "hello one!" >> afile
user@SANS:~/redirect$ ls
afile
user@SANS:~/redirect$ cat afile
hello one!
user@SANS:~/redirect$ echo "hello two!" >> afile
cat afile
hello one!
hello two!
user@SANS:~/redirect$
```

The first time we ran the command, the file didn't exist so it was created. The second time we ran the command, the output of the command was appended to the end of the file. This is the safer redirect for obvious reasons, but there is a time and place for each.

## Passwd File

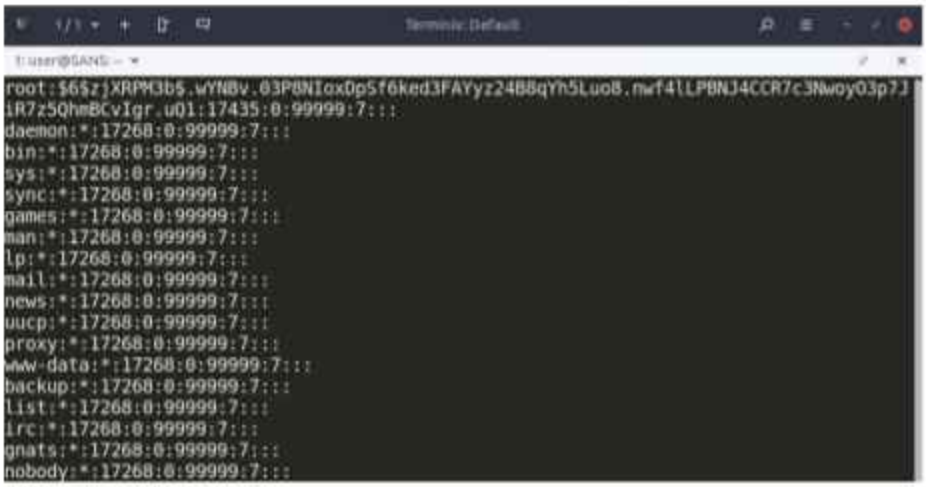
The 'passwd' file is a key file on any Linux system; it holds information about what user accounts exist on the system. On older Linux systems, it also stores each user's password hash (remember a hash is a type of one-way encryption). On modern systems password information is no longer stored in the 'passwd' file, it is instead held in the 'shadow' file.

The passwd file is located at '/etc/passwd' and looks something like this:




See that 'x' by every user account? That is where the password hash used to be, but an 'x' means the password hash is in the shadow file instead.

The shadow file is in '/etc/shadow' and looks like this:



See that big random string by the 'root' account? That is the password hash. The other accounts don't have one because if you look them up in /etc/passwd, you'll see they are all set to 'nologin', which means they are accounts that can't be logged into. These accounts are for software that is installed on the system. For example, 'www-data' is used by web servers.

So why the split? Well, it's for security reasons. In the old days, the passwd file used to store the password hash, but the problem is that all account data is also stored in the 'passwd' file, so a lot of programs need to access it. That meant the passwd had to be readable by any user on the system, which means anyone with an account on that system could steal your password hash and take it home to try to crack it.

This problem was solved by creating a 'shadow' file, which stores only the password hash for every account. The 'shadow' file is readable only by ''root', because the 'root' user is the only one that needs to be able to read the password hashes. All other programs that need access to other account information continue to use the 'passwd' file, which is still readable by anyone, but no longer contains password hashes.

Watch what happens if a non-root user tries to read /etc/shadow:

```
user@SANS:~$ less /etc/shadow
/etc/shadow: Permission denied
user@SANS:~$
```

## Scheduled Tasks

Every operating system has some way of scheduling tasks so they can be performed sometime in the future, and Linux is no different. The Linux version of scheduled tasks is called 'cron', and the place that all the scheduled tasks are stored is called the 'crontab'.

To view all the scheduled tasks for your user, you can use:

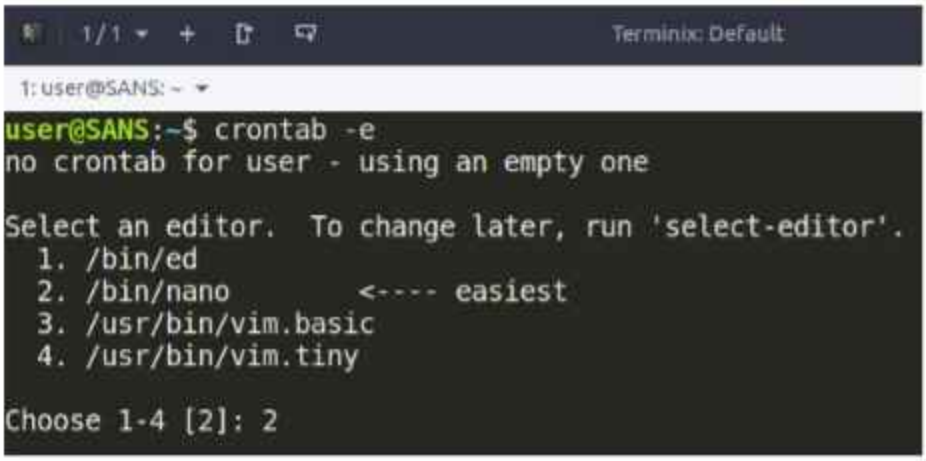
```
$ crontab -l
```

and the result is:

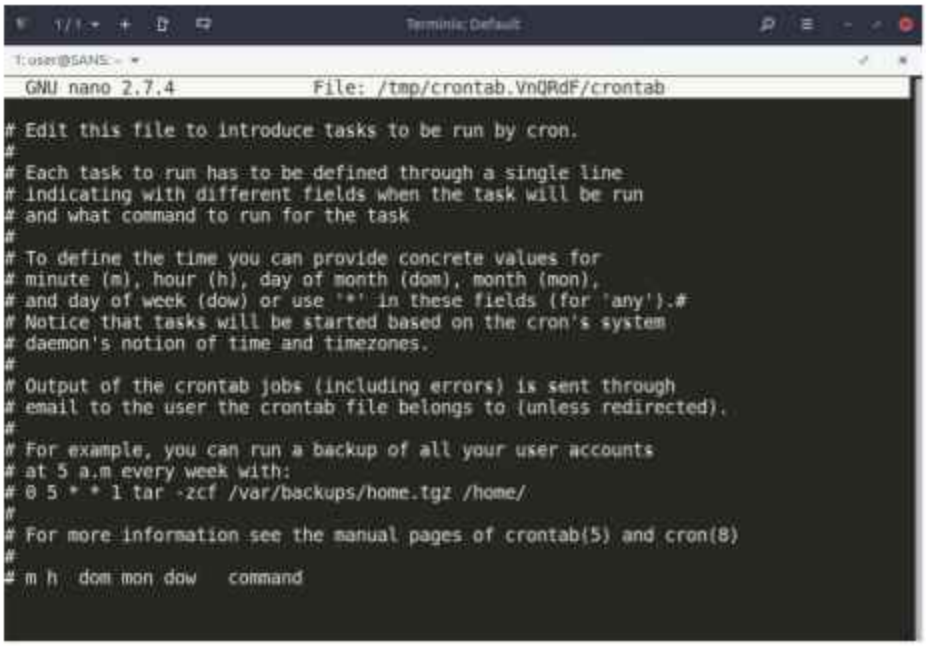
```
user@SANS:~$ crontab -l
no crontab for user
user@SANS:~$
```

In this case, we don't have any cron jobs set up for our user yet, so let's add one by editing the crontab. We can edit the crontab with:

```
$ crontab -e
```




So, here it is asking us which text editor we want to use to edit the crontab. It correctly tells us nano is the easiest, so we'll just use that by entering '2'.




Here we have a blank crontab file, we just have to add an entry. We'll use the 'date' command to append the date to a file so we can see the command running over and over. This is the exact command we are going to set up in cron to run:

```
user@SANS:~$ date >> /home/user/test
user@SANS:~$ cat /home/user/test
Sun Oct 8 12:12:37 PDT 2017
user@SANS:~$ date >> /home/user/test
user@SANS:~$ cat /home/user/test
Sun Oct 8 12:12:37 PDT 2017
Sun Oct 8 12:12:50 PDT 2017
user@SANS:~$
```

So, our expected result is to get an entry in the 'test' file every time the cron runs.

Each entry in the crontab must use the following structure:

```
Minute Hour DayOfMonth Month DayOfWeek Command
```

You can use a wildcard operator (*) to specify 'every', such as every minute, or every hour. You can also use a comma separated list, for example:

1,7,14,22 for the DayOfMonth would run the command on the 1st, 7th, 14th, and 22nd day of every month.

You can also use a dash ('-') to specify a range of dates, so for Month you could use 1-7, which would run the command on the 1st through to the 7th month.

For the command, you should use the absolute path to the command wherever possible, which means finding where the 'date' command is installed using 'which'.

Which tells us that the 'date' command is installed in '/bin/date' so we'll use that and add an entry, like so:

```
31,59 * * * * /bin/date >> /home/user/test
```

Therefore, our command will run on the 31st and 59th minute of every hour, on every day of the month, on every month and every day of the week. Now we just have to wait until the appointed time to see if our file is updated.

We cleared the file and then waited for the 31st and 59th minute to pass, and we get:

```
user@SANS:~$ cat test
Sun Oct  8 12:31:01 PDT 2017
Sun Oct  8 12:59:01 PDT 2017
```

It looks like it is working as expected. The date was printed 1 second past the 31st minute and the 59th minute like clockwork.

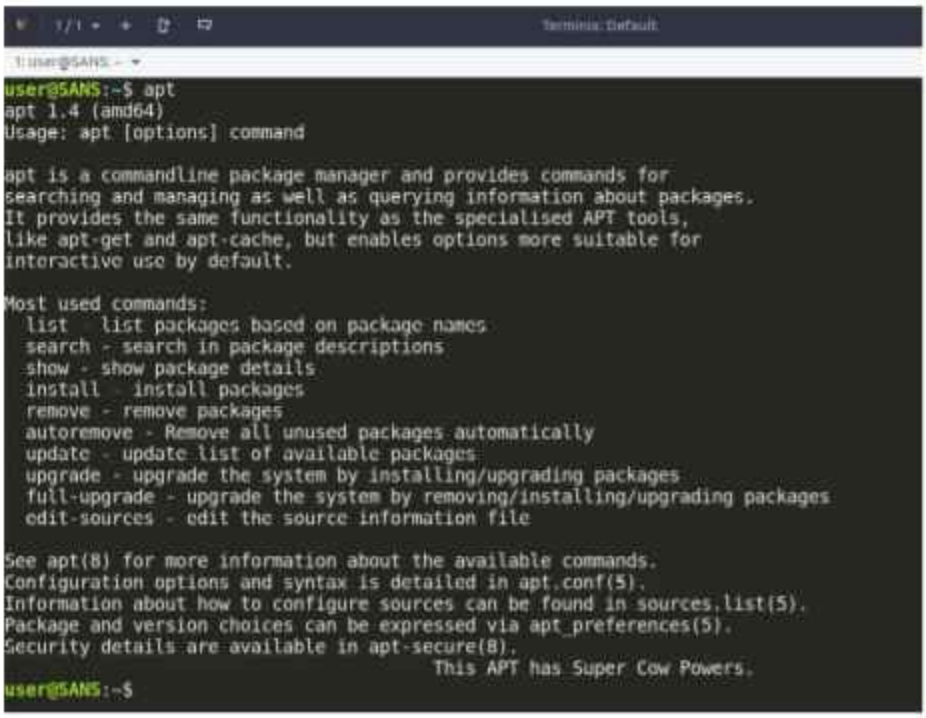
## Package Managers

Package managers are programs that are designed to simplify the installation of new software on Linux. They are present on most modern Linux distributions, and they make installing software easy. There are two common package managers; you'll mostly have access to one or the other.

### apt-get

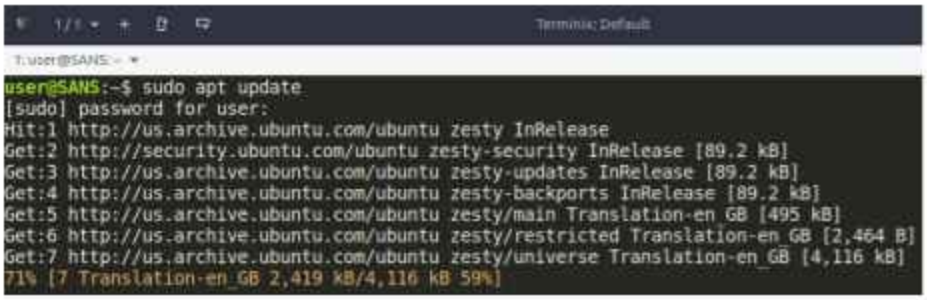
The 'apt' package manager is present on most Linux distributions that have been based on 'Debian'. Debian is a popular distribution that many other distributions are built on top of. The popular and user-friendly 'Ubuntu' distribution has also been based on Debian, so it has the 'apt' package manager.

To access it, you can use either 'apt-get' or 'apt'. Either will usually work. Take a look:




The way 'apt' works is by having a list of 'sources'; these are servers which contain information about available packages, download links and version information. The first thing you should do is make sure your sources list is up to date. You can do this by using the 'update' parameter, remembering to run 'apt' as root:

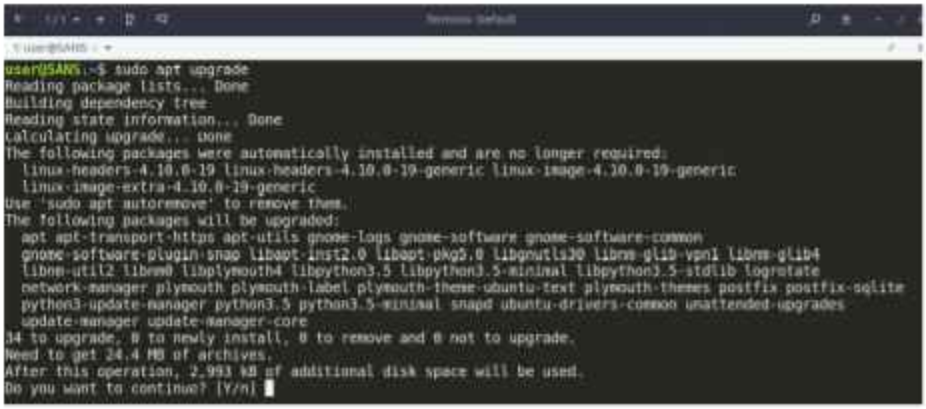
```
$ sudo apt update
```




You can see the new sources are being downloaded. It's important to note; this is just updating the list of available packages and versions. It isn't updating any software that is already installed.

If you do want to update software that has already been installed on your system, you'll have to use the 'upgrade' parameter, like so:

```
$ sudo apt upgrade
```

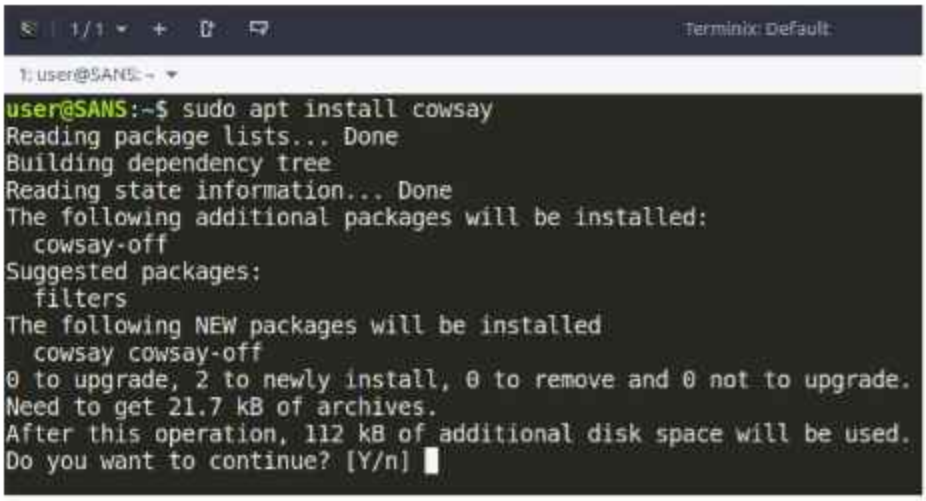



Here we have 34 software packages which can be upgraded. By entering 'y' and then hitting return, they will all be automatically upgraded to the latest version available in our sources list.


The upgrade process is automatic and seamless.

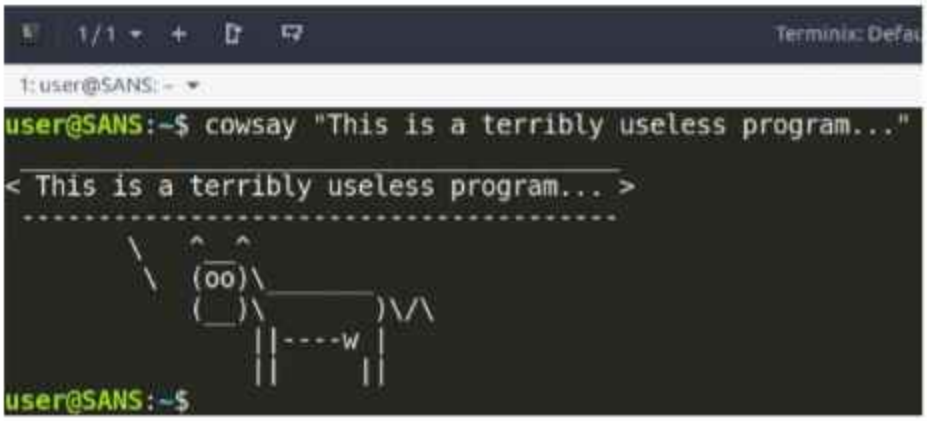
You can also use the package manager to install new software. We'll install the 'cowsay' program now:

```
$ sudo apt install cowsay
```




Once we agree, the 'cowsay' package and any additional packages that 'cowsay' needs to function will automatically be installed for us:


And to check if the program is installed, we'll run it:




And yes, that is all the 'cowsay' program does. Once we realise it's a useless (but mildly amusing) program, we can remove it with:

```
$ sudo apt remove cowsay
```

Don't forget to take a look at the man page for more information on 'apt'.

**yum**

The next package manager on the list is 'yum', which is a package manager found in 'Fedora' and Fedora-based distributions. The idea is similar to 'apt' except you don't need to update the sources list yourself. It gets updated automatically whenever you run a command that involves looking up sources.

To upgrade all your installed software:

```
$ sudo yum update
```

Or if you'd prefer to upgrade just one specific package:

```
$ sudo yum update cowsay
```

To install a package:

```
$ sudo yum install cowsay
```

And to remove a package:

```
$ sudo yum remove cowsay
```

## Packages

Sometimes you'll find a piece of software that isn't available in your package manager. You might be able to find it 'packaged' for your Linux distribution, however. A package usually ends in either '.deb' or '.rpm', depending on which type of distribution it was packaged for.

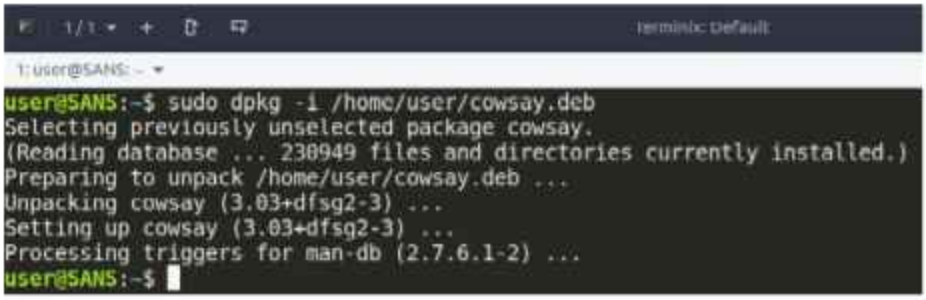
### dpkg

A '.deb' file is a package that was generated for the 'Debian' distribution or a Debian-based distribution, such as Ubuntu. You can install these packages with a program called 'dpkg'.

```
user@sans:~$ ls
cowsay.deb Desktop Documents Downloads
user@sans:~$
```

Here we have the 'cowsay' Debian package downloaded from the internet. We'll now install it with 'dpkg':

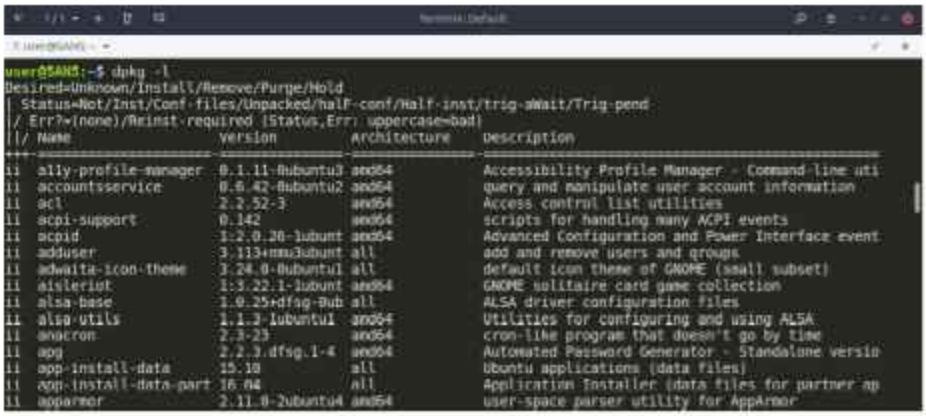
```
$ sudo dpkg -i /home/user/cowsay.deb
```




Now we can run 'cowsay' as per usual. Unlike with a package manager, however, dpkg doesn't take care of installing dependencies. So, if you install a program that requires several other packages to be installed, you'll have to read which ones are required and make sure they are installed yourself.

You can list installed packages with:

```
$ dpkg -l
```




And also remove a package using:

```
$ sudo dpkg -r cowsay
```


### rpm

An '.rpm' file is a package generated for the Fedora distribution and Fedora-based distributions. You can install these packages with the 'rpm' command, like so:

```
$ rpm -Uvh /home/user/cowsay.rpm
```

You can list installed packages with:

```
$ rpm -qa
```

And once you have the package name you want to remove with the above step, you can remove it with:

```
$ rpm -e cowsay
```

Like 'dpkg', 'rpm' won't automatically install any dependencies, so you'll be responsible for finding and installing any software that the software you installed requires in order to function.

## apt-get Installation Walkthrough

The apt command is one of the most popular package managers, and in this module we walk over finding, installing and then removing software. The ability to use apt enables quick resolution of missing software and satisfying dependencies.

## Building From Source

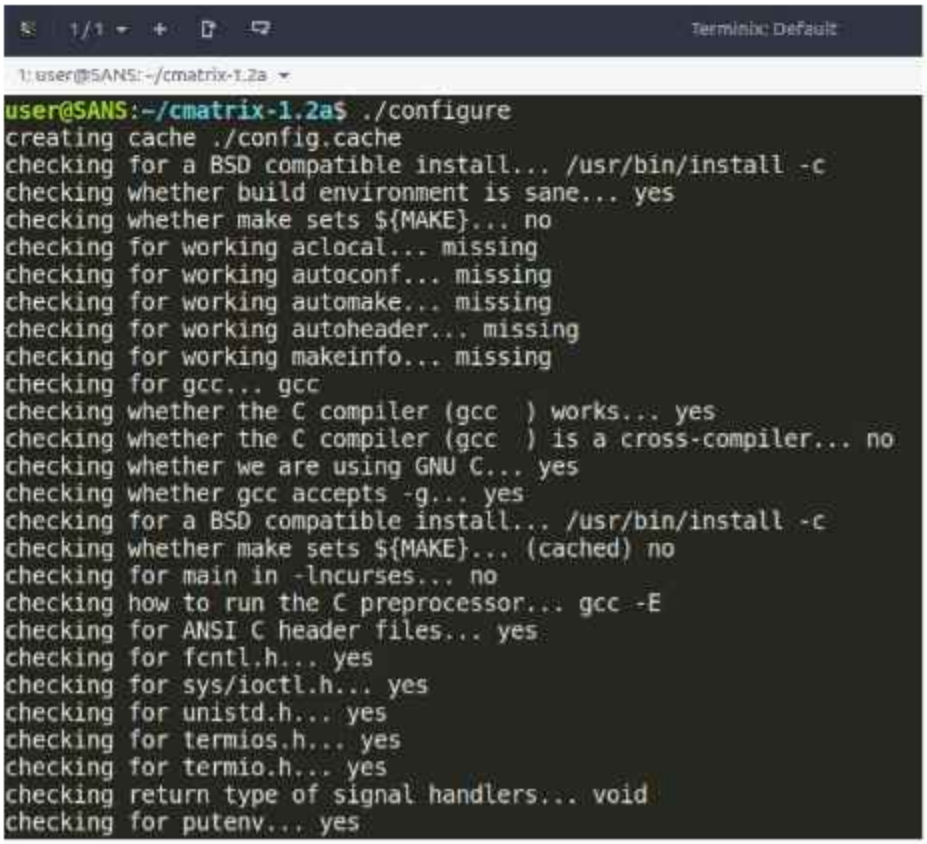
Sometimes you may not be able to find a package in your package manager, and there might not be a pre-made package for your distribution available. Don't panic, though, because you can still install it as long as you can find the source code for the program. You'll just have to compile it from scratch.

Sometimes this process can be quite smooth, but sometimes it can be quite troublesome, and you'll end up searching on Google for answers and fixes. The process can vary depending on the software you are trying to install, so it's best if you check the documentation before trying to install it. Sometimes the source code folder will contain a text file called 'INSTALL', which provides installation instructions that you should also read.

For most programs, the procedure is as follows:

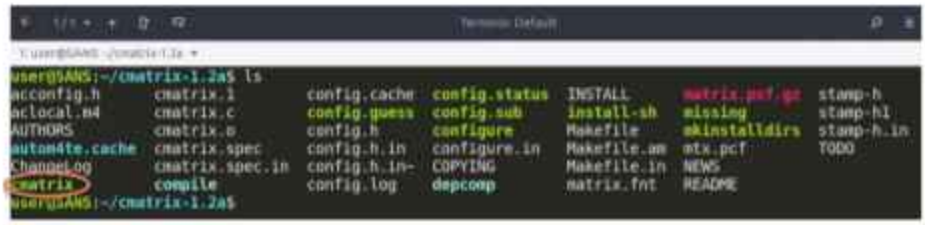
fi. Change directory into the folder that contains the source code. In our example, we'll be installing the program 'cmatrix'.


fi. Run the 'configure' executable script; this will generate a 'makefile', which is built for your system. Basically, all the compiler options that will generate a working executable for your system and processor architecture will be stored in the makefile.

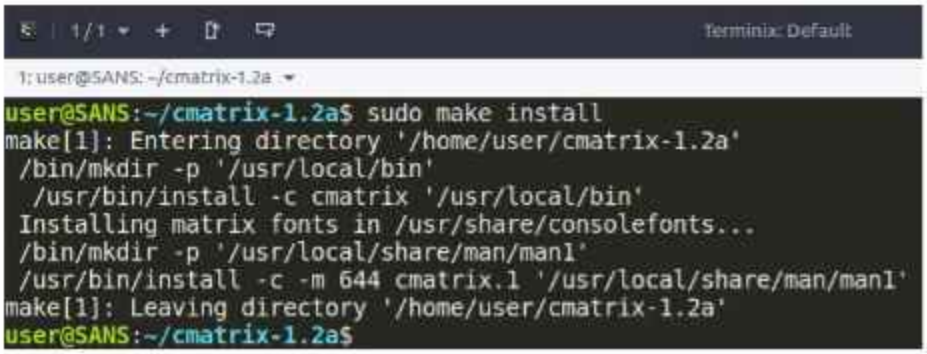



fi. Once the makefile has been generated, run 'make' to compile the source code into an executable.


You can see we have a warning, but it isn't an error, so it will have run successfully nevertheless. A warning means the program compiled, but there might be unexpected behaviour when running it. Typically, this is up to the programmer to fix, and we won't be able to do anything about it without editing the code, We can ignore it, assuming the programmer tested their code and it works as intended. At this point, we have the 'cmatrix' binary in our current folder:



fi. However, it isn't installed in a folder in the PATH just yet. To do that, we have to run `sudo make install`:




This will take the 'cmatrix' binary and any other required files and install them in system folders, so that cmatrix can be accessed by typing the command no matter which folder you are currently in. Let's leave the 'cmatrix' folder and try the command 'cmatrix':

That's right; it is another useless program which just makes the terminal window look like something out of the cult movie 'The Matrix'.

Altogether, this was a fairly smooth installation, however we did run into an error in running 'make' the first time. We were missing the 'libncurses-dev' package which caused 'make' to throw an error at us. After installing 'libncurses-dev' with 'apt', we were able to progress. You may well have your own issues installing software this way. Unfortunately, the only way to solve these issues is to either Google search for them or find the cause yourself.

Another potential issue is 'dependency hell' where the software you are trying to install requires several more pieces of software to be installed. You go to install those, and they require even more software to be installed and so on. There is no real fix for this unless the software is available from a package manager. If it isn't then all you can do is keep installing software until everything you need is finally installed.

## Using SSH

When it comes to Linux, one service is more important than them all (arguably). That service is SSH. SSH stands for Secure SHell; it's a way of letting people log into a Linux computer over the internet. It's terminal only, so you won't be able to get a GUI, but having been through this course and become familiar with the text-only way of doing things, I'm sure that doesn't phase you at all.

Most Linux distributions come with SSH enabled by default. All you need is the username and password to your Linux computer (and of course you'll need to set up port forwarding on your router if you are behind NAT.), along with the IP address. To log in to an SSH server, you'll want to use the 'ssh' command if you are on Linux or Mac OS. If you are on Windows, you can download an SSH client such as PuTTY.

The command to log in to SSH is:

```
$ ssh username@ipaddress
```

Once you connect, you'll be asked for the password to that account. Submitting the password will get you to a prompt, similar to if you were sitting at that computer.

Some SSH servers are configured to use a keyfile (a keyfile is just a text file that contains an encryption key) instead of or as well as a password to log in. If a keyfile is required, you have to specify the location of the file when you connect with:

```
$ ssh -i /path/to/keyfile username@ipaddress
```

## Customising Your Shell

You can modify your user experience of the command line in a simple remarkable number of ways. In this video we will:

- Explore command completion to expand tab functionality
- Explore the `zsh` shell as an alternative to `bash`
- Install a plug in manager to expand the functionality with community capabilities
- Configure the look and feel with themes, using powerlevel10k and pure
- Modify the intelligence provided on the command line to be aware of version control capabilities like git, and command exit codes
- Configure the vim text editor with plugins, including syntax highlighting
- Get a custom version of cat called ccat that colorises output
- Explore shortcuts for executing commands with sudo

DO NOT PANIC if you find this video a lot to take in. We cover concepts and commands like `git` in lots more depth in the programming section. If you have not used a command line interface extensively before you will likely find this walkthrough fast - but it is built up from what you have been practicing in the Linux modules. Take your time to go back over them and watch this video carefully.

You should try this yourself on your own system, and use the community and Google as a tool to customise plugins to your liking. If you use a virtual machine you can take snapshots and experiment, reverting whenever you make a change you do not like!

## Search Superpowers

## Contents

This module will provide a look into how search engines work, and the most efficient way to use them.

We will be covering the following components:

- How search engines work
- The best way to construct a Google search
- Keywords and colons in Google
- Wildcard operators
- Quotes
- Google as a calculator
- Troubleshooting simple problems using Google
- Alternative search engines

After completing this module you will be able to search Google with style and efficiency; perform advanced searches using keywords to narrow results by filetype; and troubleshoot basic computer problems using Google and other search engines.

## How Search Works

Search engines are a key aspect of life in the digital age. We use them all the time, but how many people can really get the most out of a search engine? The problem is that not many people understand them.

Search engines run software called crawlers. They will start by visiting a page, then looking at all the links on that page, making an index of all the words on that page. The crawler will then follow the links on the page to the next page and do the same there. Eventually, the crawler will have indexed every publicly accessible page on the internet, or at least that is the idea.

The actual algorithm that search engine providers use to display relevant search results to us is secret (a bit like KFC's secret sauce), but we do know a few generalities. To continue with our food theme, imagine you search for 'pizza'. The first search result you see will be the page that most other pages that contain the keyword 'pizza' link to. The idea is, if many pages that contain the keyword all link to one page, that page is most likely to be an authority on that search term.

## Constructing a proper search query

The average internet user might conduct a search query as follows:

```
What should I do if my computer can't connect to the internet?
```

The above example is the wrong way to perform a search using a search engine. It will work, but you'll actually get lower quality results this way.

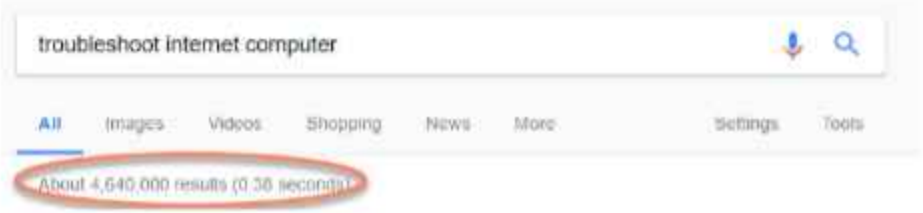

Here is the bad search. Notice the huge number of results we are seeing? It's going to be a problem filtering the good results from the bad.

Google is a keyword-based search engine, meaning each word is taken on its own, and not as part of a sentence. Google is going to look for pages which contain all of those words and rank them in priority order. The more keywords you have, the broader your search and thus the less relevant your search results will be.

In fact, to make it work better, Google will try to strip out some of the unnecessary words automatically, such as 'I', 'we', 'the' and so on. Of course, Google isn't perfect, and it won't catch everything, so a well-constructed search is key.

```
troubleshoot internet computer
```

This is a far better search term, likely to get you more relevant results. Notice how it doesn't make sense as a sentence though. The key is, we identified the words that are most relevant to the search. The order doesn't matter either.




You can see with the proper search term we've cut down the number of results significantly, and the pages we do get are ones that are relevant to what we want to know.

## Commands and Colons

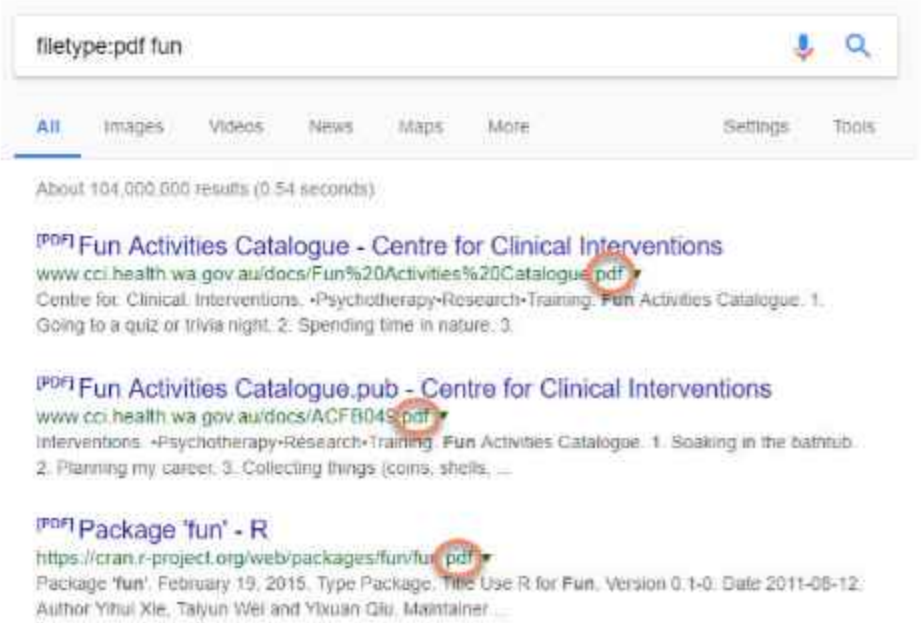
Now that we know how to make effective Google searches, let's take it a step further. Google actually has a series of commands which can be used to refine searches.

Did you know you can restrict a Google search to a particular site using the `site:` keyword? Say I wanted to search https://bbc.co.uk for the word 'hacker'. I would construct my search like so:

```
site:bbc.co.uk hacker
```


Notice how all the search results are from bbc.co.uk?

There are plenty more keywords like this. Amongst the most useful is filetype which can narrow down a search to files of a particular type.




In this example, I searched for files of type 'pdf' which contain the word 'fun'. Notice the links all end in .pdf here.

You can find Google's full list of commands here.

## Google Dorks

You may laugh at some of these, thinking they are pointless, but you would be shocked at some of the content that Google has indexed. Clever Google searches that can be used to find content that people never intended to be put online are called 'Google Dorks'. You can use Google to find:

- Unsecured security cameras
- Passwords (yes, people put files with passwords in them online)
- Vulnerable software
- Documents that were obviously never supposed to be shared with people

And much more...

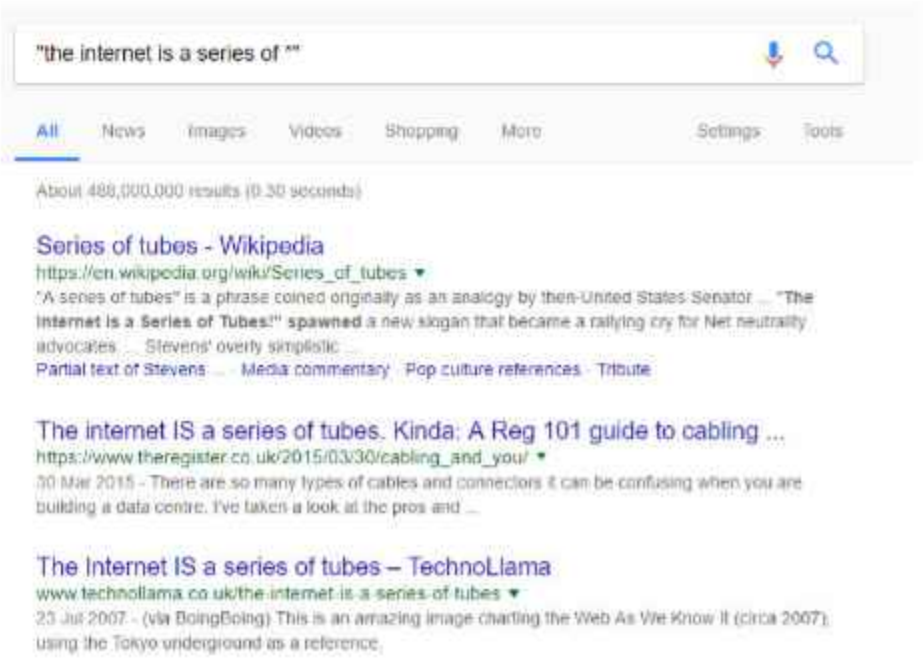
## Wildcards

Imagine the following scenario: you need to perform a search, but you don't know one of the words you need to search for. For this, you can use a wildcard operator. A wildcard will match any word.

The wildcard operator is the humble asterisk (*).

So let's try it out:

```
"The internet is a series of *"
```



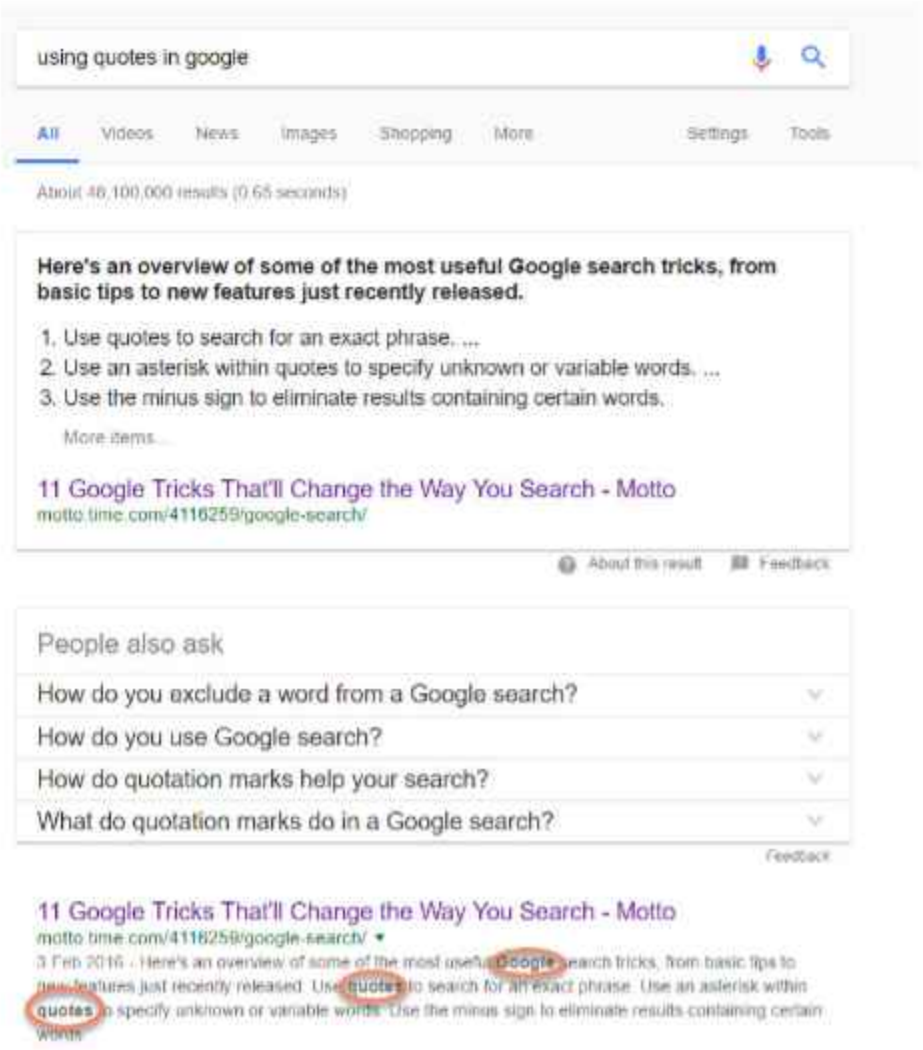

Notice here all the search results are for 'The internet is a series of tubes', even though I didn't include tubes in the search term. Oh, by the way, the internet **is not** a series of tubes, it's a geek joke. But geeks are cool these days, right?

## Quotes

We've said earlier that Google is a keyword search engine, so each word is taken on its own. Well, what if you don't want that to happen? What if you care about the order of the words?

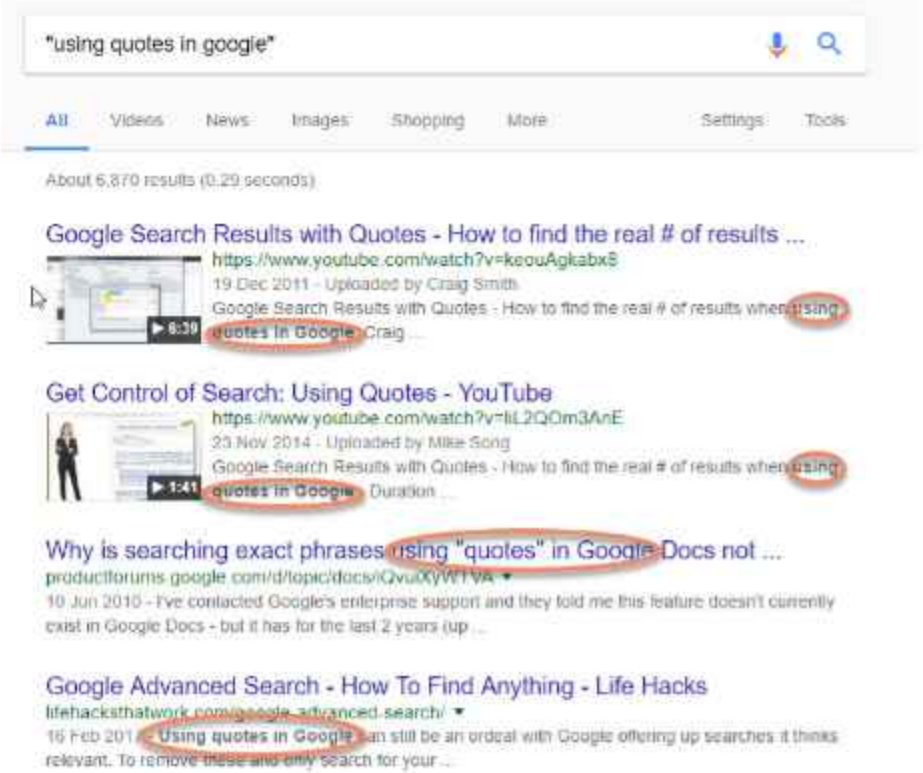
using quotes in google

This search term, without quotes, gives us different results to the search term with quotes. Here it is without quotes:




Notice that in the result, the bold words are what Google matched the result to. The words are spread out in the text.

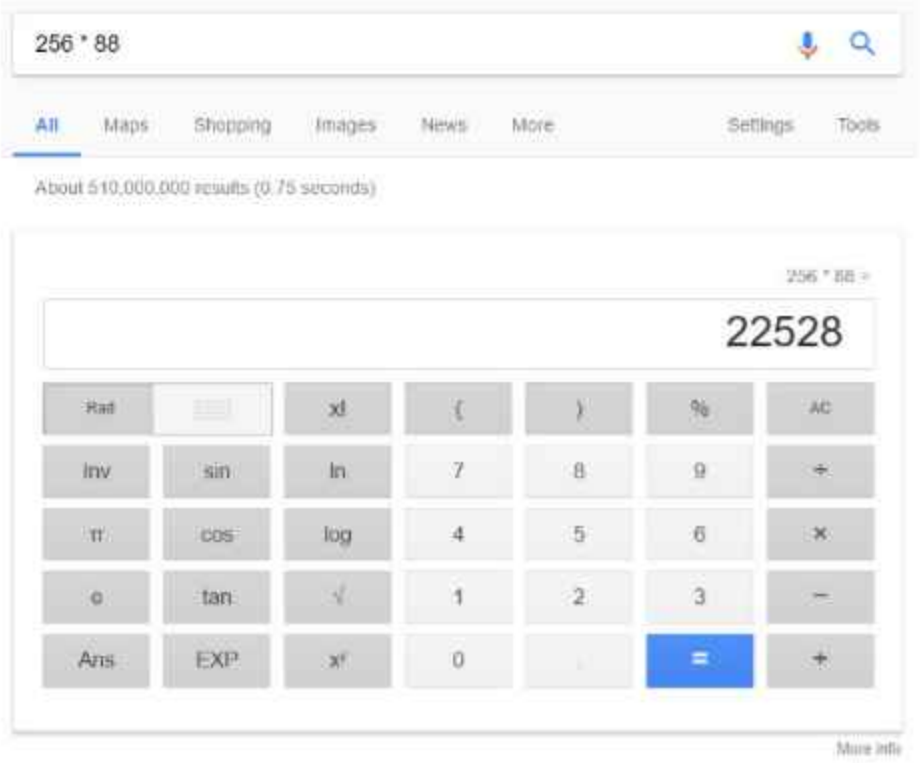
The same search with quotes:



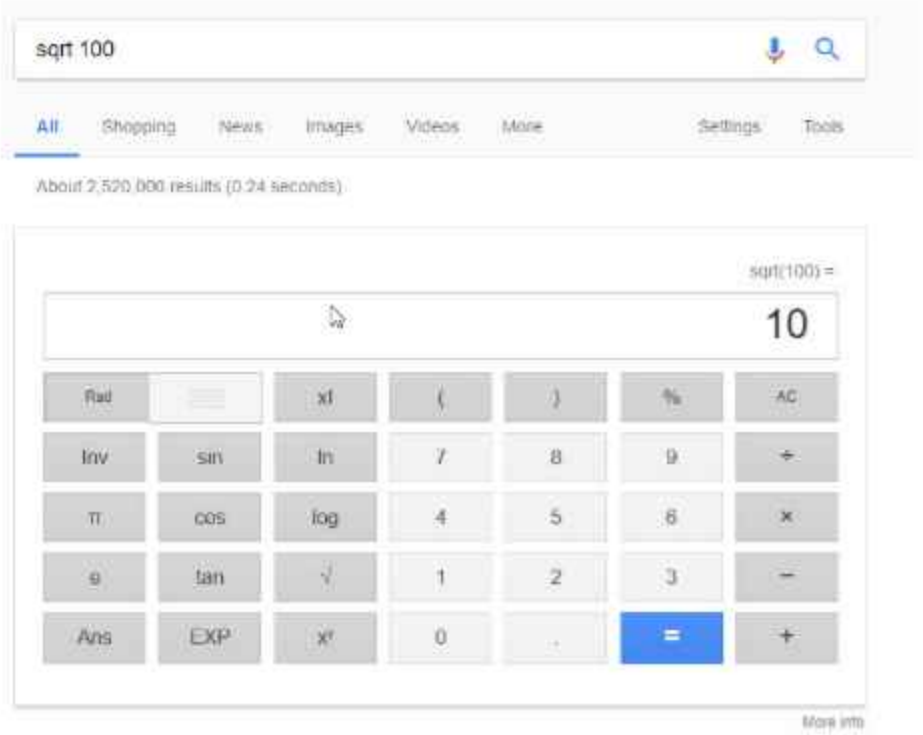

Notice here the exact phrase is used on the pages and the words are kept in the right order.

## Google as a Calculator

Google can be used in place of a calculator too. Just enter your calculation in Google. For example:




or




or just get to the blank calculator by searching on 'calc'.

## Troubleshooting

One of the most common uses for Google in the IT industry is to troubleshoot problems. Unless you are on the absolute cutting edge of research, you are bound to have a problem that someone else has already solved. The problem is, you may not know how to approach getting the answer you need. In this section, we will take you through the process of troubleshooting a problem with a (fake) piece of software on a computer.

Let's say we're using Macrosoft Letters (a fake word processing application) to write an important document. It's late, your boss needs it tomorrow first thing, but the application keeps crashing every time you try to open it.

The first search term will be quite generic, as you don't know much by this point.

```
macrosoft letters crash open
```

This search might lead you to some forums where someone is complaining that this is happening to them. Someone responds, saying "What does the error message in the log file say?", but there are no more posts on that forum thread.

The key piece of information here is that there should be a log file where the error message is written. So the next step is to find that log file.

```
macrosoft letters log location
```

This search term might lead you to the official Macrosoft Letters documentation which tells you the file is located in the 'My Documents' folder in a folder called 'Macrosoft Letters' and the log file is called 'logfile'.

Opening the log file in your text editor, you see the error message says: "Error 42. Application quit unexpectedly."

That isn't a very useful error message, but you can at least search for it:

```
macrosoft letters error 42
```

That search term might lead you to another help forum where someone claims to have solved the issue by deleting a preference file located in 'My Documents' then 'Macrosoft Letters' and then 'prefs.ini'.

Because you don't want to risk anything, you make a copy of the preference file to your desktop before deleting it from the Macrosoft Letters folder. Then you launch Macrosoft

Letters again, and it opens!

## Alternative Search Engines

No two search engines are the same. They all have their own search algorithms for displaying search results, which they won't share with others. This means that the same search with two different search engines will undoubtedly produce different results. That can be a good thing; you may be struggling to find a piece of information you need with your normal search engine. Why not try a different one?

Here are some that you might want to try:

- Google
- Bing
- Yahoo
- DuckDuckGo
- archive.org

## Google: In Practice

James Lyne shares some real-world examples of how cyber security practitioners can use Google to their advantage.

# WWW and Serving

## Contents

Web applications and websites represent a huge volume of modern business and consumer applications.

In this module, we will be covering:

- Web Servers
- HTTP Protocol in Depth
- HTML
- JavaScript
- PHP
- Client Side vs Server Side
- Cookies & Local Storage

## Web Servers

To understand what happens when you visit a web page, we first need to talk about web servers. Web servers are software applications that accept and process requests according to the HTTP protocol. At the simplest level, a web server will send HTML back to the browser, which will use that HTML to render the web page.

There are two main types of web servers. The first is the generic web server: these are multi-purpose applications that serve files that exist in a certain folder on the operating system. The second type is the custom web server: these are typically programs that are purpose-built to serve a particular site. Something like NodeJS falls into this custom category.

### Generic Web Servers

The two most popular web server products that fall under the generic category are currently Apache and Nginx (pronounced engine-ex). Both of these products are configured to use a folder as the 'web root'; that folder will contain the files needed to run the website. The folder should include an index file, named either index.html or index.php: this is the file that is sent when a request is made to '/'. For example, if you visit https://www.google.com/ you will get the index page at the top level of the web root directory.

Generic web servers are easy to set up and require no programming knowledge. Most of the internet still runs on them, and that is unlikely to change. All the files in the web root folder are accessible by visiting the site that the web server is configured to serve.

### Custom Web Servers

Custom web servers are purpose-built programs designed to run one specific site. Instead of serving files directly out of a folder, the routes usually need to be programmed into the software. In other words, the code of the web server will define what happens when a user tries to access a certain path or route. The code may say, if the user is browsing to '/help', then send this HTML as a response, for example.

These types of web servers are often used by large or complex web applications because they grant more freedom than generic web servers. Features that are needed can be added, while features that are unnecessary can be ignored. The downside is that you need to be a programmer to serve even a simple website using this method.

### Port

Typically web servers will listen on either port 80 (HTTP Unencrypted), or 443 (HTTPS Encrypted). These are the default ports, so if you go to your browser and access a site with http:// or https:// you will automatically be assumed to be using these ports. Web servers

can be configured to listen on non-standard ports, but that means anyone who needs to access them will need to put the port number at the end of the domain, like so:

https://some-fake-domain.fake:8008

In the above example, 8008 is the port number that our web server is listening in on.

## HTTP Protocol in Depth

The HTTP protocol comes down to a series of requests and responses. The browser makes a request to the web server, which returns some kind of response. This is what is powering your web browser when you connect to web sites and interact with them.

An HTTP response consists of two parts: the response header and the response body. The response header contains metadata (data about other data) such as a timestamp of the response, the web server software that sent the response and other factors, which we will cover later.

### Requests

There are several different kinds of requests which are supported by the HTTP protocol:

- **GET:** A GET request asks to retrieve a specified resource. When you visit a page such as 'https://some-fake-domain.fake/about.html' you are asking to retrieve the 'about.html' file from the web root.
- **HEAD:** A HEAD request asks to retrieve a specified resource, but without the response body. In other words, retrieve only the metadata without the data. This type of request is not common and is more often used when developers are testing their site.
- **POST:** A POST request is used to send data to the web server without expecting anything back. This is commonly used by HTML forms, for example.
- **PUT:** A PUT request is used to ask the web server to store the data sent in the request at the path requested. So for example, a PUT request to 'https://some-fake-domain.fake/about.html' would add the data sent to the web server in the PUT request as 'about.html' (if authorised). If 'about.html' already exists, it is overwritten with the new data. This one is also not common.
- **DELETE:** A DELETE request deletes the resource specified in the request (if authorised). Again, this one is not commonly seen.

There are others, but they are even less frequently used. For the most part, the ones you should pay most attention to are the GET and POST requests.

Requests can also contain other information than just the page being requested. For example, the user agent (an identification string used by the browser to tell the site which browser is being used to view the page), the date, the content type (in the case of POST or PUT requests), etc. The list is really very large, so it is best to look it up.

### Response

The HTTP response is split into the response header and the response body. The response body contains the data that was retrieved. For example, if you send a GET request to 'https://some-fake-domain.fake/about.html' then the response body would contain the contents of 'about.html'. The response header, on the other hand, will contain metadata

about the request, including usually valid request types for that page, the response length, date, name of the server, type of web server being used and so on. Once again the full list is very large so if you need it, it is best to look it up.

One very useful fact to remember is the definition of HTTP response codes. There are many of these but as we start to dissect web traffic and protocols in depth later being able to recognise a web request that 'worked' vs failed is invaluable. A key example is:

- 200: OK, I will provide you with the answer to this request.
- 302: Found, but redirect over here as this resource has moved.
- 404: File not found, this request doesn't match something I can serve.
- 500: Whoah, something went wrong! Internal server error. This commonly happens if you are trying to run server-side code and a fiaw was triggered that was not handled; these can be interesting if you run in to them in a security test!

You can find exhaustive lists online and explanations by searching for HTTP response codes.

#### HTTP/1.0 or HTTP 2.0

HTTP/1.0 has been around for a really long time and has a number of arguable inefficiencies. For example, it is a text-based protocol so you can read it. This is not the most efficient way for computers to communicate where a binary protocol can yield much more information packed into a short space. The design goal of HTTP/2 is to reduce latency. It achieves this with a number of enhancements:

- Fundamental re-work of the protocol
- HTTP header compression
- Pipelining of requests
- Multiplexing multiple requests over a TCP connection
- Fixing the head of line blocking issue with HTTP/1.0 (this is where you have to wait for the first part of the communication to complete before you can do more)

It is, therefore, harder to 'just read' but with the right tools does not present significant difficulty. HTTP/3 is just around the corner too, with the proposal and some active implementations turning up, but at the time of writing it has not shifted to default or widely available in browsers.

## HTML

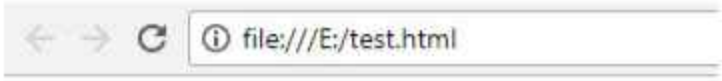
We've mentioned HTML a few times so far, but we haven't explained it, really. HTML is a markup language, rather than a programming language. In other words, it is a way of describing data. HTML uses tags, which are enclosed in "less than" and "greater than" signs, such as`<html></html>`

The first tag is opening the HTML tag. The second tag is closing the tag: notice the forward slash at the start of the tag name ('/'). An HTML page is usually formatted like so:

```
<html>
  <head>
  <title>This is the site's title tag.</title>
  </head>

  <body>
  <h1>This is a heading</h1>
    <p>This is a paragraph of text...</p>
  <p>The second paragraph goes here.</p>
  </body>
</html>
```

And the result is:




# This is a heading

This is a paragraph of text...

The second paragraph goes here.

At its most basic, this is all HTML is. Of course, if you want to make things look pretty, there is significantly more to it than that. We won't be covering that in this course.

It is important to note that HTML on its own is static, which means it doesn't change based on user input. If you want a system where a user signs in to see their account settings, you need more than HTML. You'll need something that generates HTML individually for that user and sends it. We'll get to that later when we talk about PHP.

## JavaScript

HTML is not a programming language, but JavaScript is. JavaScript is a programming language that is designed to run inside a browser. It can be used to make changes to HTML even after it has been loaded onto the page. Take a look:

```
<html>
 <head>
      <title>This is the site's title tag.</title>
  <script>
        alert("Hello, World!");
  </script>
  </head>

  <body>
  <h1>This is a heading</h1>
     <p>This is a paragraph of text...</p>
  <p>The second paragraph goes here.</p>
  </body>
</html>
```

Notice the script tags above; here is where the JavaScript code goes. You can also leave it in an external file and reference it, which is neater. In this case, we are just instructing the page to pop up with an alert box like so:


It is important to realise that you can see JavaScript if you view the page source:


This is why it isn't suitable for password protecting pages, because inevitably the password will be visible on the page even if you try to make it as obscure as possible.

## PHP

If we want truly interactive pages, we need to use a server-side programming language, which generates HTML and sends it to the clients browsing the page. This is where PHP comes in, although there are other server-side programming languages which you could also use.

The difference between PHP and JavaScript is that PHP is executed by the web server and then the result of that execution (usually HTML) is sent as a response to an HTTP request. The user never gets to see the PHP even if they view the source code of the page. With JavaScript, the code is executed in the user's browser. In other words, JavaScript is client-side, and the user is ultimately in control of it. PHP is server-side, and the user doesn't get a choice as to what happens; it just happens, and the result is sent to them.

To run PHP, we need the web server to support it. Up until now, in our tests we have just been opening the HTML file in our browser, and that has worked, but from now on we need to load the page through a web server. Let's see how it works:

```
<html>
  <head>
  <title>This is the site's title page.</title>
  </head>

  <body>
  <h1>This is a heading</h1>
     <p>This is a paragraph of text...</p>
  <p>The second paragraph goes here.</p>
     <p>The result of 2 + 2 is <?php echo(2 + 2); ?></p>
 </body>
</html>
```

Take a look now:


# This is a heading

This is a paragraph of text...

The second paragraph goes here.

The result of 2 + 2 is 4

Notice the output is "The result of 2 + 2 is 4"? We asked PHP to print the result of 2 + 2, and so it printed 4 to the page. If we view the source code, however:

```
<html>
  <head>
    <title>This is the site's title page.</title>
  </head>

  <body>
    <h1>This is a heading</h1>
    <p>This is a paragraph of text...</p>
    <p>The second paragraph goes here.</p>
    <p>The result of 2 + 2 is 4</p>
  </body>
</html>

```

There is no sign of any PHP in our page source. That is because the PHP code was executed on the web server and the resulting HTML was sent to us in the request. Server-side languages are the real heavy lifters of the internet; they are the reason we can have e-commerce and a variety of other interactive sites.

Think about a simple login and registration process. Once you submit the registration form, your account details will be saved by a server-side language into a database. When you go to log in, you will submit your account details, and the server-side language will look in the database to see if your username and password match. If they do match, the server-side language will send you to your account page which has your username on it, and various other things which are unique to your account. The page is not the same for every user. This is all done by server-side languages, such as PHP.

## Cookies & Local Storage

### Cookies

Now, we have to talk about cookies, but this is not as delicious as it initially seems. In the context of web browsing, a cookie is a tiny file that a web server can create on a visitor's computer. The file can hold any small amount of data, depending on what the developers of that site want to store.

The most common use of cookies is to save a unique identifier, called a session ID, after the login process is complete. Every page you visit on that site after logging in will cause your browser to transmit that session cookie to the site, which will tell the site not just that you are logged in, but which account you are logged in as.

Cookies have an expiry date, which is set by the site when it creates the cookie. This is the reason why you can log into a site and then close the browser and come back to the site, and you'll find you are still logged in, providing you didn't leave it too long. Unfortunately, cookies are also commonly used to track you, although not in the way most people imagine.

Only the domain that gave you a cookie can ask for that cookie back, so for the most part, sites can't know which other sites you commonly visit. There is one exception to this rule, though, and that is advertising. The reason is that adverts on websites are not usually served by that web server; instead, they are embedded from the advertising company's web server. This means that the advertising company's server can give you a cookie through the ad to identify you as if you had visited their site directly. The ads embedded in sites can also ask for that cookie back from your browser, and since the advertising company knows which site the ad you saw was embedded on, they know which sites you commonly visit. This is why if you start searching for a t-shirt to buy, a lot of the ads you will see after that will be for t-shirts.

### Local Storage

Sites can also store files on your computer (to a 5MB total file size) in the browser's local storage. This can be a sneakier way for sites to track users as most people have heard that they can clear their cookies to stop being tracked, but a lot of people still don't know about local storage, so they don't know that they should clear it too.

Local storage is intended to provide much more powerful and sizeable storage to applications based on how the internet and web applications have developed. Whilst it can be used to track and store data that might be undesirable to you, it is also often used in powering many of the richer web application interactions you might have. Modern web browsers really are incredibly feature-rich and complicated!

# Networking 1

## Learning Objectives

After completing this module, you should be able to:

- Name common pieces of network hardware and explain their purpose.
- Explain the purpose of different types of network addresses.
- Explain the purpose of packets and protocols.
- Explain the differences between TCP and UDP.

## Module Content

This module will provide a brief overview of the inner workings of a network, and will aid you in understanding how computers communicate with each other, and for what reasons.

We will be covering the following components:

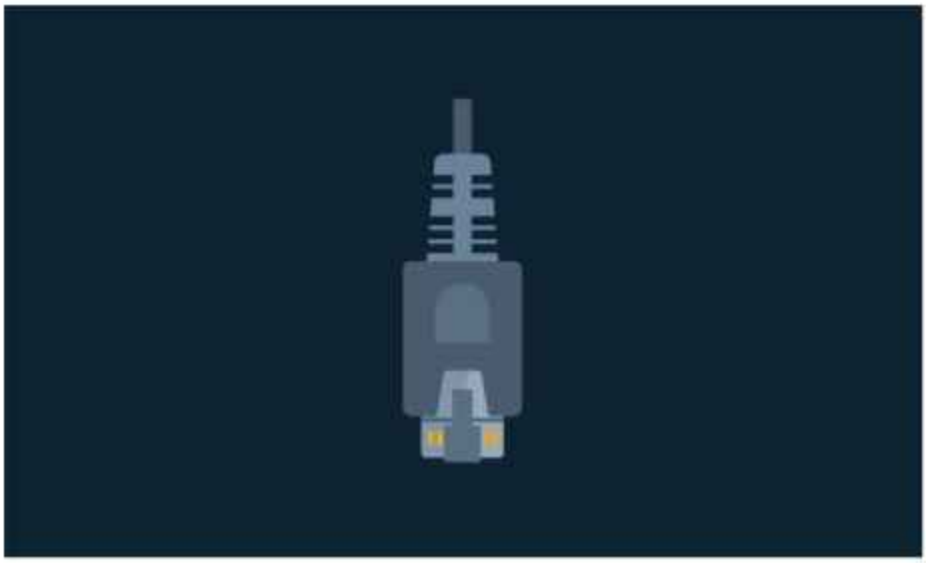
- Network Hardware
- IP Addresses
- MAC Addresses
- Packets
- Protocols
- TCP Protocol
- UDP Protocol

## Types of Networks

A computer network is a set of computers that are connected (or networked) together. The largest network in the world is, of course, the internet. In fact, the internet is an example of a WAN (Wide Area Network), which is a network covering a large geographical area. Smaller networks, such as office or school networks are called LANs (Local Area Networks).

When we talk about computers, we mean anything with a processor in it. That means desktop computers, laptops, phones and many others. Even some kettles now connect to the internet. And yes, that does mean a kettle that can connect to the internet has a processor.

Typically we connect computers over a LAN using an Ethernet cable or radio signals (WiFi). The LAN will have a router, which connects to the internet.



This is what an Ethernet cable looks like. The colour of the cable may differ, but it's only cosmetic and doesn't indicate anything about the cable.

## Topologies

The way a network is physically laid out is called the network 'topology'. There are several types of network topology; we'll cover a small subset of these.

Perhaps the most common network topology seen in LANs is the 'star' topology. This is where each computer in a network connects to a central point, such as a switch. This is what is most likely to be present in your home network.

Here is an image of a star network topology:

The benefit of the star topology is that it is easy to maintain, any computer or cable in the network could fail, and the others would not be affected. Of course, if the central point fails then that is a different story.

Another notable topology is the "bus" topology, in which all computers are connected by the same Ethernet cable. At the ends of the Ethernet cable lie line terminators, which discard any data that has not been read by a computer. This topology is not very common as only one computer can communicate at any one time.

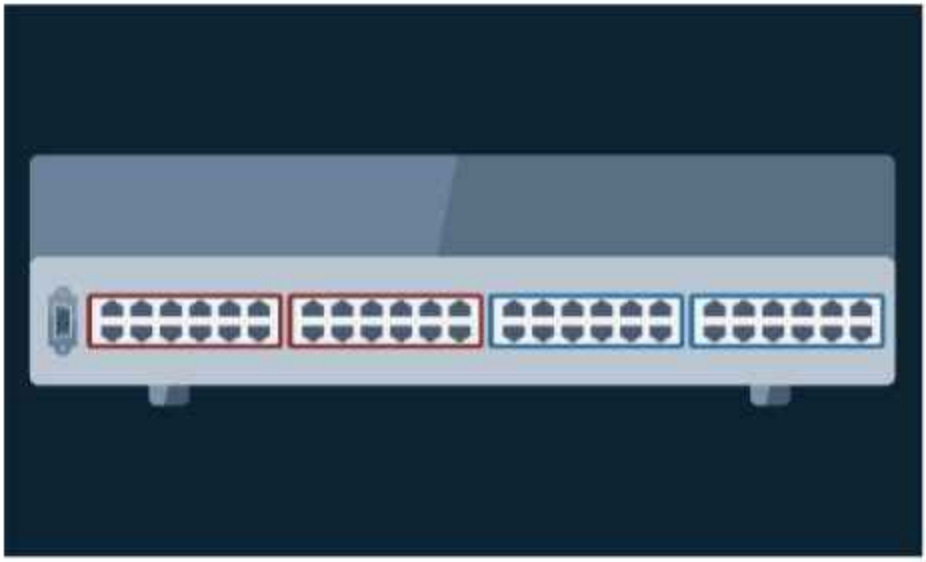
Here is an image of a bus network topology:

There are many other topologies, such as the 'token ring' and 'ring' topologies. We won't go into them in too much detail. If you are setting up your home network, we suggest you use the star topology!

**Switches**

A switch is a device that connects computers on a network together. A switch is the device that sits in the middle of a star network topology. Many computers can be connected to the switch, and the switch will receive all data on the network and decide which cable to send the data through, based on which computer the information is destined for.

Here is an image of a switch:



**Hubs**

A hub is similar in function to a switch, in that it connects computers on a network together. Unlike a switch, it isn't 'smart'. It receives data, but it doesn't know which computer to send it to, so it sends it to every computer connected to the hub. It is then up to the computer to receive that data and decide if it was intended to receive that data or if it should discard it.

Hubs were used before switches became commonplace, but they are still used today in one very particular capacity. When we talk about wireless access points (sometimes the functionality is built into the router, and we call them wireless routers), we are talking about wireless hubs. Unlike a switch where data can be sent over wires, a wireless hub can only broadcast data over radio waves, so there is no way to direct data to only one device. That means all wireless access points are actually hubs.

Here is an image of a hub:

**Routers**

A router is a device that connects two networks together. Most commonly this will be connecting the LAN to the internet, but it can be used to connect any networks together. The router will make the decision about which piece of data needs to go to which network.

Here is an image of a router:

Note that most routers these days include the functionality of a switch, but the number of ports they offer is limited. If you need more ports, you can buy a switch and plug it into one of the LAN ports on the router.

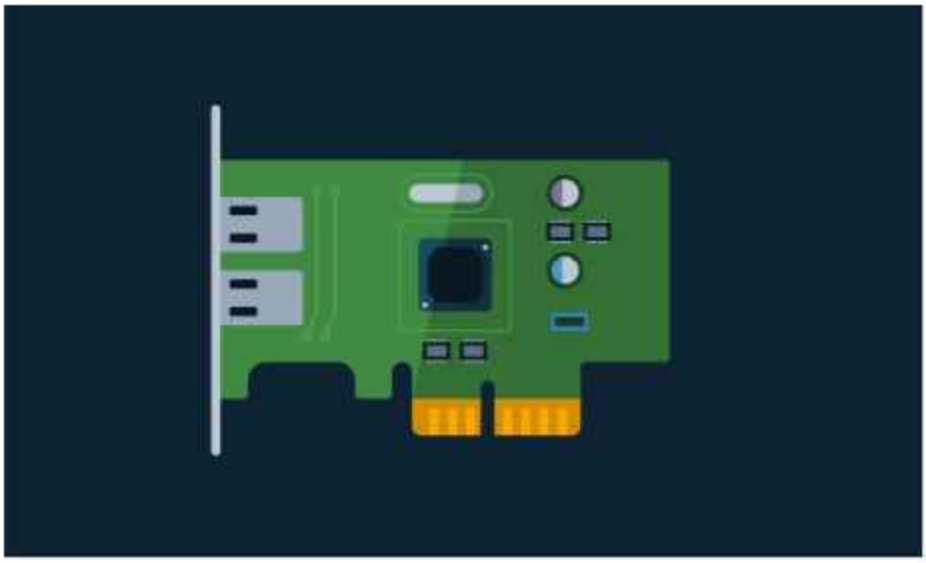
**NICs**

A NIC, or a Network Interface Card, is a piece of hardware attached to a computer (usually internally), which allows the computer to interface with a network. A computer with an Ethernet port has a NIC inside of it, which provides the means to connect to a network.

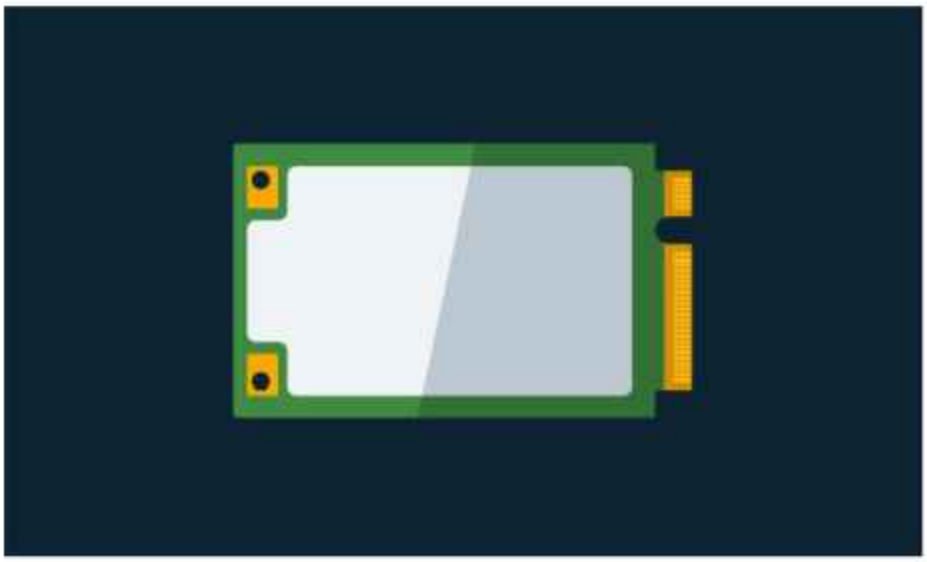
Computers which have wireless built-in will have a NIC inside of the computer that provides the appropriate hardware for connecting to a wireless network. Computers that can use both wireless and Ethernet will have 2 NICs.

It is also possible to connect an external NIC to your computer via USB; this can be useful when your computer does not have Ethernet or wireless capability.

Here is an image of an Ethernet NIC:



And an image of a WiFi NIC for a laptop:



## MAC Addresses

A MAC address, or Media Access Control address (also known as a hardware address), is an address that is burned into the NIC when it is produced. MAC addresses are meant to be globally unique; no two NICs are supposed to have the same MAC address. In practice, it's hard to verify the claim, but the likelihood of two computers on the same local network having the same MAC address is so small it's almost irrelevant. If there are two computers on the same network with the same MAC address, connectivity issues may occur, however.

MAC addresses are used to identify computers on a local network. They enable a switch to know which cable to send information down, in order to send data to a particular computer. They are used together with private IP addresses, but the difference is that private IP addresses can change, while MAC addresses are meant to stay the same.

Here is an example of a MAC address: 34:13:65:76:09:86.

### IP Addresses

An IP address is similar to a street address, in that it identifies your computer on the network and allows other computers on the network to find your computer. Each NIC (Network Interface Card) is assigned an IP address when it connects to a network. There are two types of IP addresses that we are going to look at for this module: public and private, sometimes called external and internal.

An example of an IP address is: 192.168.0.77

### Private IP Addresses

A private IP address is one assigned to your computer on the local network (LAN). Private IP addresses have to fall within certain ranges and are only accessible to computers on the same LAN. People on a different LAN cannot access your computer using the private IP address. Each private IP address must be unique on the local network; if two computers share the same private IP address then things will start going wrong. Typically, one computer will be unable to access the network, but other problems may manifest.

### Public IP Addresses

A public IP address is internet facing. In most home and small business network setups, the entire LAN will have only a single public IP address, which is shared by all computers on the LAN. The ability to share an IP address is handled by something called Network Address Translation (NAT), which we will cover in a later module.

## Packets

A data packet is a unit of data, which can be transmitted over the network. All data must be placed into (encapsulated into) packets before it can be transmitted. The packet contains not just the data (the payload), but also other information, such as the destination IP address and MAC address.

Think of it like sending a letter. You write the letter and place it into an envelope, putting the recipient's address on the envelope before posting it. Data packets are much the same; they contain data to be transmitted over the network, and the packet headers include other data such as where the data is going, amongst other things.

Packets are typically quite small pieces of data, so any large piece of data must be split up into several packets before it can be transmitted over the network. The maximum packet size depends on several factors, such as the protocol used in the packet, but typically you will see a maximum packet size of about 1500 bytes - 65535 bytes (64 KiB).

## Protocols

A protocol is a set of rules, which governs how two or more parties interact with one another. For example, diplomatic protocol applies to how nations interact with one another. Language is also a type of protocol; there is an agreed upon sentence structure, and an understanding of which words mean what.

Interaction between computers also requires protocols; these are rules which define the format of data. Additionally, some protocols have checks in place to make sure the data was transmitted successfully, to make sure the data was not corrupted in transit, and other such rules.

In this module, we will cover two key protocols: TCP (Transmission Control Protocol) and UDP (User Datagram Protocol).

## TCP Protocol

The TCP (Transmission Control Protocol) is widely used on the internet. It's a protocol designed with the**reliable** delivery of data in mind. That makes it perfect for the majority of common tasks on the internet (such as web browsing).

The protocol involves a connection setup, called the 'TCP handshake', which prepares two computers to talk to each other. Once the handshake is performed, the data is sent between computers. After the TCP connection is done, the TCP teardown closes the connection. The TCP protocol includes measures to re-transmit data that was lost in transit, or corrupted en route to the destination. Of course, the downside of such measures is that TCP is comparatively slow to transmit data.

## Ports

A port is a communications channel for applications running on the operating system to listen to. For example, a web server will typically listen on port 80 and port 443. When you connect to a website, you send the web request to the IP address of the server, and the port that the web server is listening on. The port is needed to separate communications destined for the web server application from other communications that the server might also need to receive.

Ports are numbered, between 0 and 65535 (notice how 2^16 is 65536, which if you count the 0, gives you a maximum value of 65535). No two applications on the computer can listen to the same port. Once an application is listening on that port, it is known as being bound to that port.

The TCP protocol has its own port range, which is separate from other protocols. For example, if you have TCP port 80 and UDP port 80, they may have the same number, but they are different ports altogether. You could have one application listening on TCP port 80 and a different one listening to UDP port 80, and there would be no confiict.

## UDP Protocol

The UDP (User Datagram Protocol) is key to many applications that require fast data transmission at the expense of reliability. That's because, unlike TCP, UDP does not need a connection handshake. It also does not detect if a data packet is missing, it never asks about re-transmission, and if the data it gets is corrupt, it will just ignore it instead of asking for it to be re-sent.

You will frequently see it being used in video chats, for example, when it doesn't make sense to wait on a single dropped frame to be re-transmitted when the user will not notice 1 out of 30 frames missing in that one second. Other applications where UDP is commonly used are VOIP (Voice over Internet Protocol, aka voice chat), online video games, and generally any other application that cares more about speedy data transmission than reliability.

## Ports

Just like with TCP, UDP has 0 - 65535 ports. It is **important** to note that these ports do **not** confiict with TCP ports; they are entirely separate.

# Networking 2

## Contents

In this module, we will be covering:

- Different types of IP Addresses
- Network Address Translation
- Subnetting and Subnet Masks
- TCP Protocol in more depth
- UDP Protocol in more depth
- Protocols in more depth

This will lay the foundations for abuse and attack later in your studies.

## Internet Protocol Version 4

The version of IP addresses most people are familiar with is Internet Protocol version 4 or IPv4. These are written in the format: xxx.xxx.xxx.xxx, for example: 192.168.0.1. IPv4 addresses are 4 bytes long, between each period (.) is one byte. Why is this important? Well, IPv4 is old; so old that when it was conceived no one thought computers would be as popular as they eventually became. When IPv4 was conceived, no one ever thought that four bytes wouldn't be enough, but that is the reality we are facing today.

Think back to our data-1 module where you learned about counting in binary. 4 bytes is 32 bits of data, so 2^32 or 4,294,967,296 possible IPv4 addresses. That's a lot of addresses, but frankly, we've used them all. In fact, we knew this problem was coming a while ago.

## Network Address Translation

Network Address Translation or NAT was designed as a way to delay the problem of running out of IPv4 addresses. If you recall from the last module about public and private IP addresses, NAT is the reason we have a distinction at all. Someone thought, 'what if we took a whole bunch of IP addresses and designated them as private?'. That means everyone in the world can use the same bunch of IP addresses on their local network and never have to worry about confiicts. You can have your IP address set to 192.168.0.1 and someone halfway across the world on a different network could also have the IP address of 192.168.0.1, and you would never clash. The problem with that is, you could never talk to each other either, so you need at least one 'public' IP address for the network. That public IP address can be shared across every computer on the network. The router is responsible for converting the packets that come in from public IP addresses to private IP addresses and vice versa.

Ultimately, NAT was only ever meant to be a stop-gap solution, but it was very effective at delaying the coming problem. It meant that many computers could access the internet with only a single IPv4 address used up. It was so effective that even when a new internet protocol standard was released, which solved the IP address shortage problem, many people refused to switch to it and stuck with IPv4 and NAT.

## Internet Protocol Version 6

The new internet protocol standard, which was designed to solve the IP address shortage, was released as IPv6. IPv6 addresses are really long, and so they have rules about how they can be shortened, which frankly just increases the complexity. That is one of the reasons why there has been so much resistance to switching to the new protocol.

An IPv6 address, without any shortening rules, looks like this:

2001:0db8:0000:0000:0000:ff00:0042:8329

After applying shortening rules, it looks like this: 2001:db8::ff00:42:8329.

We'll show you how the shortening rules work in subsequent modules.

IPv6 addresses are 16 bytes long, two bytes between each: 16 bytes is 128 bits (compared with IPv4's 32 bits), which means there are 2^128 possible IPv6 addresses, or 3.4 x 10^38 possible addresses. To give you an idea of how many this is, that is enough IPv6 addresses for every atom on the surface of the earth to have one, and we still wouldn't run out.

There are still a few challenges to be worked out with IPv6, but it is genuinely being used out on the internet now, particularly in many countries that were late to the internet party and therefore didn't get assigned many IPv4 addresses. For the rest of us, IPv6 seems like it's always just around the corner, but there's always one reason or another not to make the switch. Eventually, it will be forced on us by necessity if nothing else.

One of the most interesting things to consider is the lack of NAT. In an IPv6 world, each computer could have its own public IP address, and would not need to do NAT anymore, but NAT somewhat by accident is also a decent firewall. If you are behind NAT, computers on the outside cannot connect directly to you, unless you set up a NAT forwarding rule at the router to allow it (if you ever hosted an online game server, you're probably familiar with this particular task). Think about it, if someone on the internet wants to connect to your computer, they have to use your public IP address, which is also shared by the other computers on the network. When the router receives the connection request, it has no idea which computer on the local network that connection was intended for, so it drops it. Once we stop doing NAT, many computers that were protected pretty much by accident will suddenly be exposed to the internet.

## Subnets

A subnet is, just as it sounds, a sub-network. It's a way of splitting up a network into segments. You often see it on local networks inside larger organisations, where one subnet might be assigned to one department and a different subnet to another. You always need at least one subnet, even if it spans the entire network.

An IP address actually consists of two parts. One part is the network identifier, and one part is the host identifier. The network identifier identifies the network, and the host identifier identifies the individual computer connected to that network. The problem is, in the internet protocol, the part that is the network identifier and the part that is the host identifier are variable. We therefore need something to tell us how big the network identifier is and how big the host identifier is. The subnet mask is what tells us this:

Take 192.168.0.1 for example, and let's say the computer with this IP address has a subnet mask of 255.255.0.0. That means the first two bytes (192.168) are the network identifier and the rest is the host identifier(0.1). So if the last two bytes are the host identifier, this local network can theoretically have 2^16 (65,536) computers connected to it before running out of space.

If you need to connect more computers to the network than that, you need to change the subnet. If you used the subnet mask of 255.0.0.0 instead, the network identifier becomes 192, and the rest is for the host identifier. That means you could theoretically have 2^24 (16,777,216) computers connected to the network.

If you wish to connect fewer computers, you could use a subnet mask of 255.255.255.0, in which case the network identifier becomes 192.168.0, and the host identifier becomes .1. That means a possible maximum of 254 computers could theoretically join that network (with .0 and .255 being reserved).

Of course, the trade-off is in the network identifier. If you need more hosts, you can have fewer networks, and if you need more networks, you can have fewer hosts on each one.

You can have other subnet masks than just the ones listed above, such as 255.128.0.0. We won't cover how to calculate them here, but you should know they exist. You can calculate the number of networks and hosts using a subnet calculator, which you should be able to find with a simple Google search.

## Classless Inter-Domain Routing (CIDR)

Before we begin, yes CIDR is pronounced 'cider', like the drink. CIDR is a shorthand way of writing a subnet mask. Let's take the example from above; to show the network 192.168.0.0 with a subnet mask of 255.255.0.0 then in CIDR notation that would be: 192.168.0.0/16. The /16 is the number of bits that is the network identifier. In this case, 192.168 is the network identifier, that is 2 bytes or 16 bits. The same IP with the subnet mask of 255.0.0.0 (network identifier is 192.) would be written 192.168.0.0/8, and with a subnet mask of 255.255.255.0 (network identifier is 192.168.0), it would be 192.168.0.0/24.

## Private IP Ranges

### IPv4

In IPv4, by common agreement, there are certain IP addresses which are private and should never be routed to the internet. The reserved address space is:

- 10.0.0.0/8
  - IP addresses: 10.0.0.0 -- 10.255.255.255
- 172.16.0.0/12
  - IP addresses: 172.16.0.0 -- 172.31.255.255
- 192.168.0.0/16
  - IP addresses: 192.168.0.0 - 192.168.255.255

Anyone can use these IP addresses on their local network without any issues (because of NAT).

There is also 127.0.0.1 which always points back to the computer you are sending data from. In other words, it's a way for the computer to send data to itself. This is also known as 'localhost', or the 'loopback' address.

### IPv6

Although IPv6 does not need NAT and therefore doesn't need reserved addresses in the same way that IPv4 does, there are still a handful. The only one you need to know at this stage is the loopback address:

::1 is the localhost or loopback address on IPv6.

Actually, IPv6 presents an interesting problem relating to subnets. Simply put, if we use the same subnets that we do on IPv4, then some subnets could have more potential addresses in them than the whole of the IPv4 address range. That means, if you're looking for something on a network, you better have some idea of where to find it because, while on IPv4 you could scan the entire address range of your subnet looking for a device, that isn't practical on IPv6.

## TCP Protocol

As you learned in the previous networking module, TCP is all about the reliable transmission of data. Now we'll show you how the protocol actually achieves that.

## TCP Handshake

The initial connection setup is called the TCP handshake.

The computer that initiates the connection (Computer A) sends a packet with the 'SYN' (synchronise) flag enabled to the computer it wishes to connect to (Computer B). This packet contains a sequence number, which is initially randomly generated. For our example, let's say it is 0.

| *Source* | *Destination* | *Prot* | *Info* |
|---|---|---|---|
| Computer A | Computer B | TCP | `63410 > 1337 [SYN] Seq=0 Len=0` |

Computer B will respond with a packet with the 'SYN' and 'ACK' flags set (synchronise/acknowledge). This packet will contain a new sequence number that is randomly generated, for our example let's call it 0. It also contains an acknowledgement number, which is the sequence number that Computer A sent, incremented by 1.

| *Source* | *Destination* | *Prot* | *Info* |
|---|---|---|---|
| Computer B | Computer A | TCP | `1337 > 63410 [SYN, ACK] Seq=0 Ack=1 Len=0` |

Computer A will respond with a packet with just the 'ACK' flag (acknowledge), and this packet will contain the sequence number that Computer B sent, incremented by 1.

| *Source* | *Destination* | *Prot* | *Info* |
|---|---|---|---|
| Computer A | Computer B | TCP | `63410 > 1337 [ACK] Seq=1 Ack=1 Len=0` |

After this, the connection between the two computers has been established, and they can send data to each other. By monitoring the sequence and acknowledgement numbers, either side can tell if any data is missing and can ask for it to be re-transmitted.

### TCP Transmission

Once the connection is set up, data can be sent:

| *Source* | *Destination* | *Prot* | *Info* |
|---|---|---|---|
| Computer A | Computer B | TCP | `63410 > 1337 [PSH, ACK] Seq=1 Ack=1 Len=12 [Data = Hello World]` |

Notice how the acknowledgement number here jumps from 1 to 13 all of a sudden. Well, the length of the data in the packet is 12 (Hello World is 12 characters (if you count the space and the invisible \n newline character at the end), so in ASCII encoding that is 1 byte per character.) And 12 + 1 is 13. The reason the acknowledgement number is incremented by the length of the previous packet is to say, "I received 12 bytes of data, if that wasn't correct then we have a problem and you should re-transmit that packet."

Of course, in the handshake and the teardown, no data is being sent (length=0), but we increment it by one, even so, to show we received the packet, even though it didn't contain any data.

**TCP Teardown**

When the connection ends, this is called the teardown. The process is as follows:

The computer that wants to destroy the connection sends a 'fin' packet (finish) with the current sequence number. Note, even though it says it is a FIN/ACK packet below, the ACK is acknowledging the previous transmission, it isn't required to terminate the connection.

| *Source* | *Destination* | *Prot* | *Info* |
|---|---|---|---|
| Computer A | Computer B | TCP | `63410 > 1337 [FIN, ACK] Seq=13 Ack=1 Len=0` |

Computer B will respond with an 'ack' packet (acknowledge), the packet will contain a sequence number and an acknowledgement number which is Computer A's sequence number incremented by 1.

| *Source* | *Destination* | *Prot* | *Info* |
|---|---|---|---|
| Computer B | Computer A | TCP | `1337 > 63410 [ACK] Seq=1 Ack=14 Len=0` |

Computer B will send a 'fin/ack' packet (finish/acknowledge). The packet will contain a sequence number and an acknowledgement number which is Computer A's sequence number incremented by 1 (still 14 because Computer A's sequence number didn't change).

| *Source* | *Destination* | *Prot* | *Info* |
|---|---|---|---|
| Computer B | Computer A | TCP | `1337 > 63410 [FIN, ACK] Seq=1 Ack=14 Len=0` |

Computer A will respond with an 'ack' packet (acknowledge). The packet will contain an acknowledgement number which is Computer B's sequence number incremented by 1.

| *Source* | *Destination* | *Prot* | *Info* |
|---|---|---|---|

| | | | |
|---|---|---|---|
| Computer A | Computer B | TCP | `63410 > 1337 [ACK] Seq=14 Ack=2 Len=0` |

**Reset**

If for some reason, the connection cannot be torn down gracefully using the protocol above, one part of the connection can terminate abruptly by sending an 'rst' (reset) packet, which will terminate the connection immediately.

## UDP Protocol

If the last section on TCP gave you a headache, then this one will be a blessed relief. UDP is an incredibly simple protocol. There is no connection handshake, and no teardown, simply because UDP does not care if the data gets to the intended party or not. There are no sequence or acknowledgement numbers. The data is sent, and that is the end of it.

Here is a UDP packet being sent between two computers, with the data "stuff" in it:

| *Source* | *Destination* | *Prot* | *Info* |
|---|---|---|---|
| Computer A | Computer B | UDP | `50183 > 1337 Len=6 [data = stuff]` |

That's all there is to UDP. If the packet never reaches Computer B, then nothing happens and neither side knows the packet never got there. As far as Computer A knows, the packet was sent, and as far as Computer B knows, it never received anything and didn't know to expect any data.

It's the lack of any of these additional features that TCP has that makes UDP so much faster and therefore ideal for real-time applications such as video chat. Realistically, you don't care if a single frame in a video never reaches the destination, you wouldn't want to pause the whole video to wait for the data for that single frame to be re-transmitted, since most people wouldn't notice the missing frame. At worst, the video quality may degrade for a short time as a result of dropped packets, which ultimately is a better experience than waiting for missing frames or frames appearing out of order in the video.

## Protocols

We already covered what protocols are in the Networking 1 Module. To review, briefiy; a protocol is an agreed-upon standard that governs how two parties interact.

Computers use protocols extensively to communicate with each other. TCP, UDP and IP are all protocols. There are other types of protocols. The simplest to look at are application-level protocols such as HTTP (Hyper Text Transfer Protocol), used in browsing websites; and FTP (File Transfer Protocol), used for uploading and downloading files to and from an FTP server.

## HTTP

Let's look at what happens when you visit a webpage on the internet over the HTTP protocol.

### Computer A -> Webserver

Computer A sends a request to GET the file /test from the web server:

```
GET /test HTTP/1.1
```

As part of the request, computer A sends some headers to give additional information to the web server:

```
Host: 127.0.0.1:8000
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/60.0.3112.101 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
DNT: 1
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.8
Cookie: csrftoken=DBxRWJ14K4gJ4BjCWnAEIQ7pYDywitjDIeMQSq4ZIiVXxbuFkeebqSwRgufcTkGa
```

### Webserver -> Computer A

The web server responds with a code 200 OK (meaning the file exists and can be displayed).

```
HTTP/1.0 200 OK
```

The web server also sends some headers of its own, giving the web browser some extra information:

```
Server: SimpleHTTP/0.6 Python/2.7.8
Date: Mon, 28 Aug 2017 14:43:57 GMT
Content-type: application/octet-stream
Content-Length: 11
```

And finally, the data in the file is sent (in this example, the file 'test' only contained the word "testing..."):

```
testing...
```

We will be covering protocols in more depth in later modules, including the specifics of the HTTP protocol. For now, let's look at FTP:

## FTP

Here is an example of logging into an FTP server, don't worry too much about the specific commands. We'll look at it in more detail in later modules:

**FTP Server:** 220 pyftpdlib based ftpd ready.

**Computer A:** OPTS UTF8 ON

**FTP Server:** 530 Log in with USER and PASS first.

**Computer A:** USER someguy

**FTP Server:** 331 Username ok, send password.

**Computer A:** PASS somepass

**FTP Server:** 230 Login successful.

You can see this is slightly different to the HTTP protocol; the HTTP protocol tries to put everything necessary for the request together and send it all in one go. The FTP protocol is more of a back and forth conversation. You'll also note that the password was just sent in clear-text without any encryption. This is one of the downfalls of FTP and also why many people are using SFTP and other variants now.

## Walkthrough with Wireshark

In this walkthrough, we will see first-hand the differences between TCP and UDP using Wireshark as a tool to monitor the network connections. We will observe how TCP has states and fiags that enable it to identify if data transfers were successful or not, where UDP is much simpler and lighter-weight. We will also touch on the use of the follow functionality in Wireshark to reconstruct a network connection between two parties.

This is a great way to be able to study networking 'live' with real systems and to dissect protocols you may be less familiar with.

# Networking 3

## Contents

In this module, we will be covering:

- Email
- Email Servers
- SMTP Protocol
- POP3, IMAP and Exchange
- Email Spoofing & SPF, DKIM

## Email

Let's now look at how email works. At its most basic, an email is just text which is sent from one email server to another and placed in a folder called your inbox. That inbox then syncs with your computer (or is accessed through a webmail client like the Gmail web interface). The fact that it is text is important; there is no kind of encryption when the email is sitting in your inbox. Until recently, there was no kind of encryption when the email was in transit either. These days most modern email servers support STARTTLS, which uses a similar kind of encryption used to secure HTTP, to encrypt the email while it is in transit over the network.

The process of sending an email is as follows:

fi. You create the email on your computer and hit send.
fi. Your email is transmitted to your mail server (known as the outgoing mail server, because it is sending the email).
fi. The outgoing mail server will look up the mail server responsible for handling email for the domain you are sending the mail to (known as the incoming mail server, because it is receiving the email).
fi. The outgoing mail server will transmit the email to the incoming mail server.
fi. The incoming mail server will look up the email account and save the email in the correct inbox, in the correct format.
fi. The recipient's email client will sync their inbox, and they will see the unread email come into their inbox.

There are two protocols at work here, the first is SMTP (Simple Mail Transfer Protocol), and the next is the protocol responsible for syncing the inbox (usually POP3, IMAP or Exchange). Some email servers support all three protocols used for syncing the inbox, but it depends on the mail server's setup.

For the SMTP protocol, the most common mail server software includes:

- Microsoft Exchange
- Postfix
- Sendmail
- Qmail

They each have different features, but the same ultimate responsibility: to send and receive email messages between mail servers.

## SMTP Protocol

The SMTP protocol is ancient, it can trace its roots practically back to the start of the internet, but it only started becoming widely used in the 1980s. As with all older protocols, it has its foibles, and there have been several attempts to switch to a more modern alternative, but SMTP is simply too deeply entrenched to change to a different protocol now.

SMTP is a connection-oriented (TCP, remember?) text-based protocol, similar to the way HTTP is a text-based protocol. Each connection can be re-used to send multiple emails, called SMTP transactions. An SMTP transaction consists of just three commands:

- **MAIL:** The MAIL command establishes the return path (return address), bounce address (where to send an error message if the mail delivery fails), sender, amongst other things.
- **RCPT:** The RCPT command establishes the address of the recipient. You can use the RCPT command multiple times to establish multiple recipients.
- **DATA:** The DATA command signifies the start of the message text. The DATA section of an email consists of an email header and an email body. The email header contains metadata about the email, and the email body is the contents of the email itself.

Here is an example of the SMTP protocol:

```
S: 220 smtp.areallyfakedomain.fake ESMTP Postfix
C: HELO smtp.areallyfakedomain.fake
S: 250 smtp.areallyfakedomain.fake, I am glad to meet you
C: MAIL FROM:<someone@areallyfakedomain.fake>
S: 250 Ok
C: RCPT TO:<accounts@asecondfakedomain.fake>
S: 250 Ok
C: RCPT TO:<bossman@asecondfakedomain.fake>
S: 250 Ok
C: DATA
S: 354 End data with <CR><LF>.<CR><LF>
C: From: "Someone" <someone@areallyfakedomain.fake>
C: To: Accounts Dept <accounts@asecondfakedomain.fake>
C: Cc: bossman@asecondfakedomain.fake
C: Date: Tue, 15 January 2015 12:02:00 -0200
C: Subject: Test message
C:
C: Blah blah blah.
C: Here I am, sending you an email, for no apparent reason.
C: .
S: 250 Ok: queued as 12345
C: QUIT
S: 221 Bye
```

If you've been following this, you'll have noticed that we never had to log in to send this email. Authentication is not part of the SMTP protocol, but that wasn't very practical as computers and the internet started to become more widespread. These days every SMTP

server supports some form of authentication, however, once you have logged in there is nothing to stop you from putting whatever you want in the MAIL FROM. That means anyone with an email server can send an email pretending to be anyone else with no consequences. There are ways to detect when things like this happen, which we will be going over later in this module.

## POP3, IMAP & Exchange

These three protocols are responsible for syncing an email folder from the email server to your computer.

### POP3

The POP3 (Post Office Protocol 3) protocol is the oldest protocol that we are going to talk about. With the POP3 protocol, your email is stored on the email serv**until you sync with the email server**. After you have synced with the email server, then the email will be deleted from the email server. This is the same concept as picking up a letter from the post office. Once you collect it, the letter is no longer at the post office. POP3 was important because of limited space available on email servers at the time. These days it has mostly been superseded by IMAP and Exchange.

The downsides of POP3 are numerous in this modern age, including:

- You risk losing emails if the computer you download them to loses a hard drive and you don't have a backup.
- It doesn't play nice with having an email account on multiple devices because you'll end up with some emails on your phone and some on your computer (for example) and they won't be on both.

These are all reasons POP3 is not particularly useful these days, although it is still used by some people (for reasons that are beyond me).

### IMAP

The IMAP (Internet Message Access Protocol) protocol came after POP3, and it also solves most of the problems with POP3. It is the most common protocol in use today for accessing email on an email server. The messages remain on the server until the user specifically deletes them, and in addition, the IMAP server can track state on each email (whether it has been read, or unread for example).

IMAP also plays nicely with multiple devices connecting at once, and will even sync message state across devices (for example, if you read a message on your phone, and go back to your computer you will find the message has been marked as read in your inbox there also). It also supports server-side search, which means you can rely on the email server to perform a search for messages that meet certain criteria. You don't have to do the search on your computer.

### Microsoft Exchange

Microsoft Exchange isn't strictly a protocol, rather it is a server software. However, it is worth talking about here because it implements so many different protocols. Originally,

Exchange only supported the MAPI protocol, which was a proprietary protocol developed by Microsoft. These days, Microsoft Exchange also supports IMAP, POP3 and EAS (Exchange Active Sync). Generally speaking, its feature set is similar to that of any other server software that implements the IMAP protocol, if not slightly better. The downside is that it is restricted to Windows Server, so you can't run an exchange server on Linux.

## Email Spoofing, SPF & DKIM

### Email Spoofing

SMTP is an old protocol, and so there are some things that we need in the modern day that were never considered when the protocol was designed. The most serious problem is that of email spoofing (forging). Anyone with access to an SMTP server (even one they set up themselves) is capable of sending an email with any FROM address. Even though there is authentication on all modern SMTP servers, that only controls whether you are allowed to send emails through that SMTP server. Once you are allowed access, you can set any FROM address you like.

There have been several attempts to solve this problem over the years. The ones that are currently in use are SPF and DKIM.

### SPF

SPF stands for Sender Policy Framework. The idea is that in the DNS configuration for your domain, you put in a text record that is a list of all the IP addresses of mail servers allowed to send emails from your domain. When a mail server receives an email that says it is from your domain, it will look up the SPF record for your domain and compare the IP address of the mail server that delivered the email with the list of valid IP addresses in the SPF record. If the IP address is not on the list, then the receiving mail server knows the email is a forgery.

There is a glaring problem with this method, however. Many organisations no longer host their own email, they rather outsource it to a company like Google, so if you specify in your SPF record that Google is allowed to send emails from your domain, then anyone with a Gmail account can bypass your SPF protection.

### DKIM

DKIM, or Domain Keys Identified Mail, is similar to SPF, but it is one level more advanced. The idea with DKIM is that you put a public key (remember, asymmetrical cryptography has two keys: a public and private key) in your text record in your DNS settings for that domain. When your email server sends an email legitimately, it will sign that email with your private key. When the receiving server gets the email, it will validate that signature using the public key in your DNS records. If the signature is valid, then that email comes from a legitimate source, and if the email is not signed or the signature is not valid, then the email is a forgery.

This is better than SPF because most email hosts are able to have separate private keys for every domain they host email for. This bypasses the main problem with SPF, which is that anyone on the same server can impersonate you.

## How SPF Works

## How DKIM Works

In this section we walk through how DKIM (DomainKeys Identified Mail) works. When e-mail is sent, a private key held by the sending server is used to sign the message, and a hash generated for the header and a portion (or all) of the message body. The header of the message contains header values like: - d: The domain signing the message, e.g. sans.org - b: The signature of the message, produced using the private key of the sending server. Unique to the message. - bh: The hash of the message that can be validated.

When the message arrives at the recipient server, the headers can be extracted for verification. The server needs a way to verify this information out of band for the domain and to validate that the signed data was not provided by any old server pretending to be responsible for the domain. This is done by using an out of band check.

The system will query over DNS to get the public key to verify the signature. The domain key selector is a value that enables multiple DKIM entries to exist for a domain, for example for different senders like SES, your own mail server or O365. A query is made to:

```
selector._domainkey.domain
```

This returns a TXT record that contains a 'p' value. This is the public key that can be used to validate the signer of the message. Using this DNS check with the authoritative DNS server means that out of band verification of the message headers in DKIM is complete. Unless an attacker has compromised the sending server, a random mail server popping up on the Internet would struggle to achieve these header results. That being said, if an attacker controls DNS for your domain they would be able to inject their own DKIM values.

DKIM combined with SPF, and integrated to DMARC as a framework significantly reduces the chances of spoofing when sending and receiving e-mail.

# Networking 4

## Contents

In this module, we will be covering:

- DNS (Domain Name System)
- TLD (Top Level Domain)
- Authoritative Name Servers
- Forward & Reverse Lookup
- Recursive & Iterative Lookup
- DNS Records

## DNS

DNS or the Domain Name System is the glue that holds the internet together. DNS is responsible for translating a domain name, such as 'google.com', into an IP address, which computers can understand. This is important, because humans aren't very good at remembering long strings of numbers; we're much better at remembering words. This is doubly so for the IPv6 standard, which on a bad day can look like this: fd15:611d:d0dd:0209:xxxx:xxxx:xxxx:xxxx.

Every computer must have a DNS server configured in their network settings. Often your internet service provider will give you one, or even pre-setup your router to use it. Your router, in turn, will use DHCP (Dynamic Host Configuration Protocol) to tell every computer on your network to use that DNS server. In some cases, you may not have a router that supports DHCP, in which case you'll need to find a DNS server to use (Google has one, and so does OpenDNS) and configure your network settings for every device on the network. In some business environments, you may have a separate server that provides DHCP, in which case the DHCP server on the router should be disabled to avoid a clash.

Once you type a domain name into your browser, your computer will send a request to the DNS server in your network settings for that domain, and your DNS server will give your browser the matching IP address. Your browser can then happily send the HTTP request to that IP address on port 80 or 443 (remember, these are ports for HTTP).

## TLD

The DNS system is hierarchical; no one server has all the answers about every domain on the internet. The system relies on one server giving more information on the next DNS server to query, all the way down to the name server which does know the answer for that specific domain. This makes for a robust system which can deal with outages without bringing the entire internet down.

The Top Level Domain is the bit at the end of the domain name. So for 'google.com', the TLD is 'com'. For 'google.co.uk', the TLD would be 'co.uk'. The TLD indicates which DNS server to query first, so if you do a DNS lookup for google.com, you will query the name server(s) for 'com' first. That name server won't know the IP address for 'google.com', but it will know which name server you can ask to get your answer.

## Authoritative Name Servers

A name server that is authoritative for a domain is one that controls the mapping between the domain name and the IP address. For example:

Say we have a domain called 'thisisnotarealdomain.fake' and we want to map it to the IP address 192.168.0.6.

We go to the name server which is authoritative for that domain (the one that the .fake TLD name server directs us to, let's call it 192.168.0.100) and we edit the configuration of the name server to add the mapping thisisnotarealdomain.fake -> 192.168.0.6.

Every other name server on the internet must now query our authoritative name server to find which IP address 'thisisnotarealdomain.fake' points to. Our name server is authoritative for 'thisisnotarealdomain.fake' because it has the final say on where the domain points.

## Caching

We simplified the explanation above to state that every name server must query the authoritative name server to find out the IP address for that domain. This isn't strictly true because of caching. If a name server makes a query and discovers the IP address a domain points to, it can cache the result for a time. Future requests for that domain will use the saved value, to save from the name server having to make the request every time. Of course, it will still periodically erase the cache so that if the name server configuration is updated, it will refiect the new address in a reasonable time, but caching provides a measure of efficiency so that not every request to a domain has to be sent to the authoritative name server each and every time.

## Forward & Reverse Lookups

The most common usage of a DNS lookup is a forward lookup; in other words, translating a domain name into an IP address. It is also possible to do a reverse lookup; translating an IP address into a domain name. A reverse lookup is something of an oddity, because there is no authoritative name server for IP addresses.

The way we get around that is by converting the IP address to a domain name in the format: 1.0.168.192.in-addr.arpa (notice this is the reverse of the IP address 192.168.0.1). By performing a DNS query on this domain, we can look up any domains which are linked to that IP address. Of course, generally speaking, each domain must have reverse DNS configured by adding a particular DNS pointer record to the authoritative name server.

## Recursive & Iterative Lookups

DNS is a hierarchical protocol, which means you almost never get the answer you are looking for on the first query. Each query gives you more information until you have the complete answer. There are two ways that DNS servers handle this. With a recursive lookup, you ask the DNS server a question, and if the DNS server doesn't know, it will ask another DNS server, and so on, until it has the answer, and then it will pass on the answer to you. This method is not commonly seen anymore, because it uses up a lot more server resources than the alternative. Internet standards call for iterative lookups these days instead.

With an iterative lookup, you ask a DNS server a question. If it doesn't know the answer, it will respond with the IP address of a DNS server that does know. It is then your responsibility to ask the next DNS server in line the question, and then it will either answer, or send you to yet another DNS server to ask the question. This places more of a burden on the host asking the question, but it is much easier on the DNS servers.

## DNS Records

When you are configuring the authoritative DNS server for a particular domain, you will need to add DNS records for that domain. There are many different types of DNS records, so we'll only be covering the most commonly used types.

- **A Record:** The A record is the 'Address Mapping' record. This is the key record that maps a domain name to an IPv4 address.
- **AAAA Record:** The AAAA record is the 'IPv6 Address Mapping' record. The equivalent of an A record, but for IPv6 addresses.
- **CNAME Record:** The CNAME or 'Canonical Name' record is used for creating an alias of a domain name. For example, if you wanted your domain to redirect to google.com, you would use a CNAME.
- **MX Record:** The MX record or 'Mail Exchange' record specifies the mail server which is responsible for handling email for that domain. When an external user sends an email to your domain, their mail server will perform a DNS lookup for the 'MX' record to find the IP address of the mail server to send the email to.
- **NS Record:** The NS record or 'Name Server' record points to the authoritative name server for the domain in question. Usually, the NS records are configured separately on the domain registrar's systems to point to the DNS server of your choice.
- **PTR Record:** The PTR or 'Pointer' record is used for reverse DNS lookups. It ties an IP address to a domain name in the format: 1.0.168.192.in-addr.arpa PTR notarealdomain.fake
- **TXT Record:** The TXT or 'Text' record is used for storing any other textual data associated with the domain name. This is used a lot in SPF and DKIM, for mail servers where a list of IP addresses, which can be used for sending mail from that domain (in the case of SPF), or a public key (in the case of DKIM), is stored as a TXT record.

# Networking 5

## Contents

In this module, we will be covering:

- ICMP (Internet Control Message Protocol)
- DHCP (Dynamic Host Configuration Protocol)
- OSI Model
- TCP/IP Model
- Packet Headers
- ARP (Address Resolution Protocol)
- Denial of Service

## ICMP

ICMP stands for Internet Control Message Protocol. It is a protocol designed to transmit error messages and operational information between hosts on a network. While ICMP is technically in a class with TCP and UDP, it differs in that it does not communicate data and isn't typically used in end-user applications. For the most part, ICMP is used almost exclusively by network devices such as routers. There are a few exceptions, such as the ping application, which uses ICMP packets to get information about if a host is online or not.

Each ICMP packet consists of a 'type' and a 'code'. Used together they describe the packet's purpose. For example, the 'ping' program which is used to determine if a host on the network is available uses an ICMP type 8 code 0 packet. An ICMP packet with the type set to 8 and the code set to 0 is an ICMP echo request packet, in other words, once the host receives that packet, it will respond with an ICMP echo response packet, which is an ICMP Type 0, Code 0 packet. The response will allow the 'ping' program to determine that the host is alive, and the time it takes to receive the response is the latency.

Another example of an ICMP packet is the ICMP type 11 code 0 packet, or the ICMP time exceeded packet. This packet is usually sent by a gateway (router) to a system that sent a packet, where the TTL (the number of hops the packet can travel before it is considered lost and gets discarded) of that packet has expired.

There are many ICMP types and codes available. We won't cover all of them, but you'll be able to find them easily with a simple internet search.

## DHCP

DHCP stands for Dynamic Host Configuration Protocol. DHCP allows a DHCP server present on the network to assign network configuration settings to each host automatically when they join the network. This makes joining a network seamless to the end user while removing the possibility of IP address confiicts (because the DHCP server knows which IP addresses it has already assigned and can avoid assigning two computers the same address). Only one DHCP server should be used on any network, otherwise there is the possibility of confiicts where one DHCP server doesn't know the other has already assigned an IP address. Most home routers have a DHCP server built in, but for businesses they typically have a separate computer to use as a DHCP server.

It is not uncommon for DHCP to be configured with static IP addresses for specific devices which keep those devices at a predictable IP address on the network. From an attack perspective DHCP is also very interesting as controlling the networking details of clients opens up the possibility of intercepting their traffic.

## OSI Model

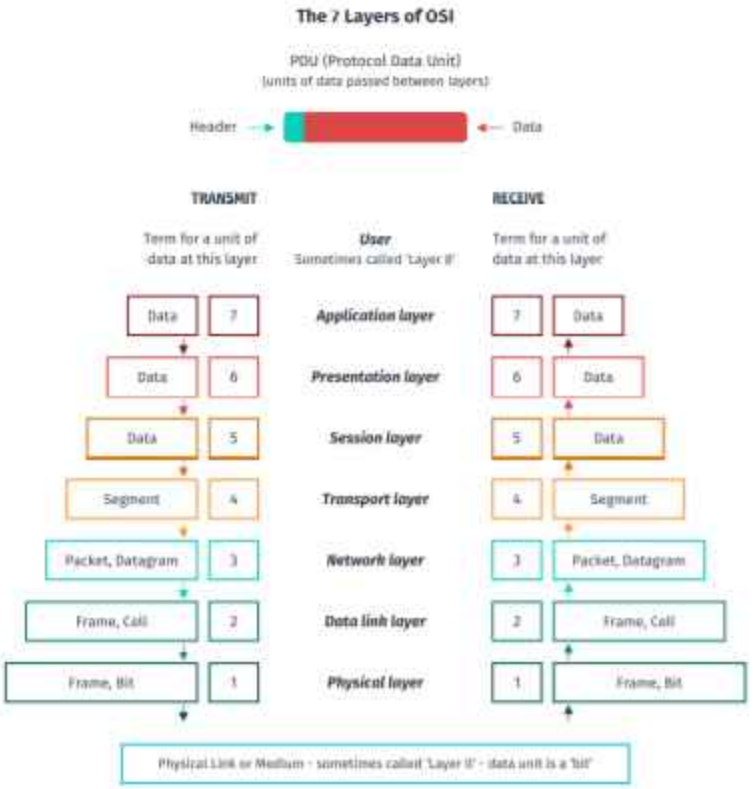
The OSI (Open Systems Interconnection) model is a way of describing the way computers communicate with each other over a network. The OSI model is a theoretical model, so not all protocols will fit neatly into the seven layers. Some will span layers, for example, TCP is a layer four protocol, but it also handles sessions, which is in layer five.

The OSI model consists of seven layers:

- Layer 7 - Application
- Layer 6 - Presentation
- Layer 5 - Session
- Layer 4 - Transport
- Layer 3 - Network
- Layer 2 - Data Link
- Layer 1 - Physical

The idea is that you start at the top and work your way down the model. Each layer transforms the data somehow until at the physical layer the data is converted into electrical signals, which are transmitted over the network. The computer receiving that information receives it at the physical layer and then works up to the application layer again until the user sees it in its original form on the receiving computer.

Here is an example of this in action:


The 7 Layers of OSI
PDU (Protocol Data Unit)
(units of data passed between layers)
Header
Data
TRANSMIT
RECEIVE
Term for a unit of data at this layer
User
Sometimes called 'Layer 8'
Term for a unit of data at this layer
Data
7
Application layer
7
Data
Data
6
Presentation layer
6
Data
Data
5
Session layer
5
Data
Segment
4
Transport layer
4
Segment
Packet, Datagram
3
Network layer
3
Packet, Datagram
Frame, Cell
2
Data link layer
2
Frame, Cell
Frame, Bit
1
Physical layer
1
Frame, Bit
Physical Link or Medium - sometimes called 'Layer 0' - data unit is a 'bit'

## Application Layer

An example of an application layer protocol is something like HTTP or FTP. For example, an HTTP GET request is an application layer protocol because the web browser will form the request necessary for displaying a page or submitting a form.

## Presentation Layer

An example of a presentation layer protocol is XML or JSON. The presentation layer deals with formatting data in a way that the intended recipient can understand. Encryption also happens in the presentation layer, if the data is meant to be encrypted. An example might be a file transfer, where the file must be converted to binary data before it can be sent.

## Session Layer

The session layer handles opening, closing and managing connections between computers. The session layer is a bit of an oddity because these days most people think TCP when they think connections. TCP is actually a transport layer protocol, which doesn't respect the OSI model, so it does**NOT** fall under the session layer.

## Transport Layer

The transport layer is responsible for end-to-end connections between computers on a network. The TCP and UDP protocols fall into the transport layer, and it is at this point that the TCP or UDP header gets added to the packet.

## Network Layer

The network layer is responsible for routing the packet over the internet. The IP (Internet Protocol) falls into the network layer. At this point, the IP header is added to the packet.

## Data Link Layer

The data link layer is responsible for encoding and decoding packets into bits. At this point the destination and source MAC address is added to the packet (in the form of the Ethernet frame, or wireless, or Bluetooth), indicating which network card on the network the packet is going to next.

## Physical Layer

The physical layer is responsible for converting the packet into electrical signals, which are sent over the network.

Of course, once the packet arrives at the destination, the reverse happens.

- The physical layer converts electrical signals into bits.
- The data link layer removes the Ethernet frame (for example).
- The network layer removes the IP header.
- The transport layer removes the TCP or UDP header.

And so on, up until the application layer where the data is received by the application in a form it can understand.

If you stick around long enough in security or forensics you may hear a joke about it being a layer 8 issue. Whilst layer 8 is not formally included in the OSI reference model, the idea is that layer 8 is the user. Therefore if you hear the phrase "layer 8 issue" you may well be hearing a reference to the user of said technology stack having made a mistake.

## TCP/IP Model

The TCP/IP model is another theoretical model designed to show how computers communicate over the network. The TCP/IP model has only four layers, but ultimately everything the OSI model covers is also covered by the TCP/IP model, it is merely the case that several layers have been joined into one in places.

The layers of the TCP/IP model are:

- Layer 4 - Application
- Layer 3 - Transport
- Layer 2 - Internet
- Layer 1 - Network Access

## Application Layer

The application layer is basically the same as the application layer in the OSI model, except it also includes the responsibilities of the presentation and session layers from the OSI model. Examples of protocols in the application layer are HTTP and FTP, basically any protocol that applications define and use.

## Transport Layer

The transport layer is a direct copy of the transport layer of the OSI model, in other words, the TCP and UDP protocols are implemented here.

## Internet Layer

The internet layer is effectively the network layer from the OSI model, responsible for routing traffic over the network.

## Network Access Layer

The network access layer is a combination of the data link and physical layers from the OSI model.

## Packet Headers

Starting with application-level data, we encapsulate that data into a packet ready for transmission over the network. The process of encapsulation occurs by adding headers to the data. Take for example an HTTP request made by a browser:

GET / HTTP/1.1

This is a request for the index page of a site made by a browser. To transmit this request to the site in question, the first thing that happens is a TCP header is added to the data. The TCP header consists of a source port, a destination port (80, or 443 usually), a sequence number, an acknowledgement number and anything else the TCP protocol implements. The original data, the GET / HTTP/1.1 is still there, but the TCP header now sits in front of it.

The next stage is to add the IP header. The IP header consists of information such as the version (4 or 6), the source IP address, the destination IP address, the size of the packet and anything else the IP protocol implements.

The next stage after that is to add the data link layer protocol (let's say we're using Ethernet to transmit the data in this case). The Ethernet frame (it isn't a header because it consists of a header AND a footer, so it frames the packet) consists of the source MAC address, destination MAC address and anything else the Ethernet protocol implements.

Finally, the packet is converted into a series of electrical impulses, which are transmitted over the network. On the receiving end, the packet works its way back up through the layers, first being converted from electrical impulses into data.

Then the Ethernet frame is removed, the IP header is removed, and the TCP header is removed until the application (the web server) receives the GET request. The web server will respond with the contents of the index page in HTML form, and that data will then be encapsulated into a packet and transmitted over the network in response.

## ARP

If you've been paying close attention, you may have noticed something a little funky. An Ethernet frame requires a source MAC address and a destination MAC address, but how do we know the destination MAC address?

Computer A

192.168.0.5

Computer B

192.168.0.10

Computer A wants to send some data to Computer B, but remember the Ethernet protocol requires a destination MAC address for Computer B. How does Computer A know what to write on the envelope (packet header)?

The answer is ARP or Address Resolution Protocol.

Computer A will send a broadcast request (a request to everyone on the local network) asking for the MAC address of 192.168.0.10. Computer B will respond with its MAC address. Computer A will then store that mapping in an ARP table which caches results (so it doesn't have to keep asking the same question over and over).

Keep in mind ARP is only for the local network. If you are sending data to the internet, then your computer should know that the IP address in question is not in your local network (because of the subnet mask). So your computer will send the data to the router and will need the router's MAC address (it will already know the IP of the router/gateway because it will be in the network settings on that computer). If your computer doesn't have it stored in the ARP table, then it will ask for the router's MAC address using ARP.

## DoS

Denial of Service or DoS attacks consume resources and prevent real customers from connecting. They come in many shapes and sizes -- for example, using just a sheer number of packets and huge bandwidth to saturate the internet connection of a server.

They can also be more application layer, for example finding a website request at /login.php that causes the server greater 'expense' in processing. The attacker then repeatedly calls and uses this thousands of times per second, overloading the system in processing terms.

## DDoS

Denial of service is problematic, but DDoS (Distributed Denial of Service) is significantly more painful. This is where attackers use a huge number of systems, for example a large bot network they control, and attack a system all at once. They can also 'take turns' and pulse. These can disrupt even the most well-connected businesses!

In some instances DDoS attacks will be protocol layer, such as just having a huge number of bot systems connecting over TCP to fiood the connection table. In other instances they may be application layer, such as having a huge number of bots turn up and interact over HTTP to post to a login form.

James Lyne shares some real-world examples of using networking in cyber security work.

## Building an SME Network

In this section we will build our very own SME network using common business grade hardware. The process is very similar across a multitude of vendors and over time these devices are becoming much simpler to configure. All the process we undertake here can be executed with a smart phone, or from a web browser. Older routers and integrated network devices like this required hundreds of hours of study of command line configuration before you would be able to even get packet routing working! As you can see from this walkthrough all of this can now be achieved with relative simplicity.

# Introduction to Servers

## Servers and Services

Servers are a crucial part of any network and a key location where lots of data is held, which makes them a tempting target for attackers. In the upcoming modules we will take a look at some common server types, their basic setup and installation procedure. We will also cover some basic server hardening and configuration, though you can find rich guides online to help you build secure servers.

Servers receive connections from client devices, which effectively means they talk to computers and not humans. It is a computer to computer interaction rather than a human to computer interaction.

The definition of a server from the Cambridge dictionary is "a central computer from which other computers get information."

The CIS benchmarks and guides provide fantastic resources for you to configure and secure common server types. They are an invaluable guide and are scaled from the more basic and unobtrusive configuration, to rigid and secure but likely to disrupt features. Take some time after the course to try applying them in your own lab environment!

## Server Hardware

A server is simply a computer which runs software that provides services. Servers can have specialised hardware requirements to perform their specific function.

For example:

- A Raspberry Pi used as a server is still a server.
- An old desktop PC with VMware ESXi installed on it is still a server.
- A dual CPU Xeon Blade Server that costs 10,000 USD is a server.

Dedicated servers (computers built with the intention of using them as servers) tend to be more powerful than your typical desktop computers. Besides having more RAM and storage, a server will likely have one or more specialised CPUs. If we remember building the computer, we used an i9 intel processor, a top of the range for consumer-level CPU. However, you will likely have ranges such as Intel Xeon or AMD EPYC (or possibly threadrippers) with servers. They may require specific motherboards that have specialised functionality, suchs as slots for multiple CPUs. A server may have two, four or more CPUs each having multiple cores, TeraBytes worth of RAM, several GPUs and hundreds of TeraBytes of storage.

Datacenters are places that are designed to house many servers. As you might expect having so many powerful computers in close proximity to one another generates a lot of heat and so datacenters are designed to provide cooling and connectivity for all those servers. In fact just walking into a datacenter will often require you to put on a warm jacket.

### How big servers are built and maintained

First came the computer that was just bigger, more hard drive space and more RAM, but bear in mind they were already pretty big. Fast forwards ten years, and they were all linked with cooling and daisy-chained together to maximise their capability. They may utilise parallel processing, breaking down a task into segments and running it on several CPUs simultaneously, then putting it back together to reduce the time it takes to execute.

Jump to the present day, and we mostly see blade servers. These are self-contained servers that are thin and can be slotted into a rack. They can still be connected together but the innovation here comes from the ability to quickly swap components. Perhaps a hard drive fails in a particular server. The monitoring system would tell the engineers which particular one and they would simply lift it out and clip a new one in place. So when one component fails it can be quickly replaced. This is necessary for continuous service. One important way of ensuring continuous service is redundancy which means having servers ready as a fallback when another server fails. This meant at times there were vast amounts of computing power that were not being utilised. Companies built intelligent software to manage these extra resources which gave rise to cloud computing. Companies realised that they could hire out unused processing power to customers. This

gave rise to dedicated cloud computing providers such as Amazon AWS and Microsoft Azure.

**Back to smaller scale**

Ok, let's scale it down slightly and ask yourself if you wanted to create your own server, what would you need.

Well, if you want to host your small website with not many visitors, then it is plausible to have a web server on your home computer, similar to the one we saw earlier. Another example that would require small efforts is a server on your local area network. Perhaps you have lots of pictures on your laptop hard drive. Well, it is possible to spin up a server for other devices connected to your network that will allow them to view the media. This will enable you to view the pictures on your laptop hard drive on your phone. This is a very basic idea of a server.

## Server Software

Another way we use the term server is related to the specific software running on a computer. For the server to do its job, there needs to be some software running that will sit and listen for connections and process the requests. This software will also create the necessary file structures based on whatever service you are running.

Take, for example, a game server. When you log in to your multiplayer game, it will connect to a server, and you may be on the same map with a hundred other players. Do you think this server would be running many different applications such as internet browsers or notepads? This kind of server will get updates as you use your input device, mouse or controller. The command is sent through the internet and updates what you have just done to everyone, usually positions and mathematical representations rather than graphically. It's almost like you are playing on a game console with split-screen multiplayer, but the screens are just far away.

Another example may be something like the syncing on your phone, tablet and laptop. There is a central server that is sending all the data and messages to each device. It is a similar principle to email, where you can insert as many email addresses to send an email to as you want. The server also allows you to sync all your devices, so they get added to the 'send' list. Don't worry too much about the workings of these; we cover them more extensively later on.

So the software on the server can be called a service or more casually a server, and one computer may be running several different services. Perhaps your home computer has a local media server and a web server, a DNS server and a proxy server for when you want to log into your computer from around the world. While this is fine for personal use modern day best practice for business use is to have one service running per server.

## What types of servers exist?

This list is not exhaustive; there are many varieties of servers, some of which we have already mentioned. Why are there so many? Well there are many things we can do on a computer? The principle remains the same, however. Something else to keep in mind is that even everyday objects now connect to servers, especially with 'Internet of Things' or IoT devices being prominent.

Note: IoT devices are classified as objects embedded with sensors and connected to a network. You may have heard them being called smart devices. All the everyday smart devices are a part of the Internet of Things. However, it does not just stop in your home.

For example, modern-day cars may be connected to a server, usually through 4g sim cards. Perhaps the pressure in your vehicle changes when it is switched off and locked. This indicates a smashed window, and this information can be sent to the car manufacturer and forwarded to the police.

Another use of IoT devices could be to check the moisture in a bridge. These days, many bridges have some form of IoT sensors to check for various attributes relating to stability and safety etc.

Here are a few common servers, but there are many more:

- Web server
- DNS server
- DHCP server
- Log Server
- Game server
- Print server
- Proxy server
- Streaming server

## Challenges of servers

We briefiy mentioned earlier some of the cost and environmental impacts of servers, but how about other challenges that may arise. Perhaps a military needs a server to be running with the best possible security, or a hospital needing a server running, for it to remain operational. Well, now we must architect a solution with security and/or redundancy in mind, respectively. This may be a nice thought experiment for yourself. Can you think of all the aspects you would need to cover to ensure the security for servers used by the military? Would you connect them to the internet? If you did, how would you secure this? When we think of servers in extreme cases, it is easy to appreciate what an exciting and monumental task this can be. A few things to consider for military servers, for example, would be:

- Will the servers be air-gapped (not connected to untrusted networks such as the internet).
- The physical security of the servers (making sure people can't simply walk up and power down the system or take a hard drive).
- Who do we trust to build the hardware components that go in the server (would they be tampered with before they get delivered or could components be swapped out while the server is in transit)?

All these precautions will depend on what the purpose of the server is.

Let's assume someone will get access. What then?

Overall, this requires a considerable effort to design well enough to satisfy the needs of the service. This should help you to appreciate the vast wealth of skills that needed to build a server. Not to mention the facilities they are contained in.

## How are they connected to the outside world?

This is an exciting topic and one that is critical to servers. After all, servers are processing connections from the outside world most of the time. Hence, we need to think about security every step of the way. First, we need to consider how we will contain and secure it physically. Should there be CCTV everywhere? If so, are we going to use on-site servers for that or the cloud?

In terms of networking, imagine if website hosting companies did not connect to the internet. That would not be very helpful now, would it? Servers that require internet connections need to have a very carefully laid out network infrastructure. Let's think back to the networking module. First of all, these servers would require a seriously powerful router to process all of the connections. Things get interesting here. You see, just for the networking alone, there would need to be several servers to run the servers. A good thought experiment might be to think of all the types of servers required to run a network.

Let's take an example. Perhaps a firm has a website, remote workers and an intranet. The intranet consists of several company document servers, each at different privilege levels. This means that some employees may not have access to particular documents or software. For example, software engineers do not need to access the accounting software. So there needs to be considerable thought when planning servers and the networks they are using, and who has what level of access.

Companies also require some layers of protection such as web application firewalls, intrusion detection systems, intrusion prevention systems, and a well designed internal network. In terms of how servers talk in the network stack, well, that will depend on what application it is running.

# Web Servers

## Introduction to Web Servers

Of all of the types of server, it seems most likely that a web server is the one you'll have heard of prior to this course. But do you actually know what a web server is, and what it does?

The primary purpose of a web server is to satisfy client requests on the World Wide Web, which translates to storing and processing web pages, and delivering them to clients, using the Hypertext Transfer Protocol to communicate. If youve ever opened a web page in a browser, the acronym HTTP will be familiar to you (although youll probably more commonly see HTTPS these days - more on that later in the course).

The worlds first web server was created in 1990 by Tim Berners-Lee, better known for being the inventor of WorldWideWeb itself - in fact, these two programs were written as part of the same project while Berners-Lee was employed by CERN, with the goal of easing the information between scientists by using a hypertext system.

This first web server ran on NeXTSTEP (a discontinued object-oriented, multitasking operating system based on the Mach kernel), and was later known as CERN httpd. Since then, these technologies spread rapidly, being ported to many different operating systems and coming into common use in scientific organisations, universities, and later industry. The World Wide Web Consortium (W3C) was constituted in 1994 to standardise and regulate further development across this group of technologies. Fast-forward to the present day, and web servers are integral to how we use the internet, and anything connected to it, in our daily lives. Internet of Things devices often run web servers, and they are growing second by second all around us!

A web server can be a piece of software, or hardware dedicated to running server software. As well as serving popular websites, web servers are often embedded in common home or office devices to help in the configuration of those devices. These include printers, webcams, and routers, particularly those serving only a local network, in schools, small offices or practically any organization. The web server can be used as part of a system for administering or monitoring devices through a web browser, removing the need to install additional software on the client computer.

The most used web servers on the World Wide Web are Apache and Nginx, both standalone products, and IIS, created by Microsoft. Google also has a small percentage of market share with its own Google Web Server (GWS), and then there are many smaller providers out there, each of which services tiny percentages of the worlds websites. This module includes a demonstration of setting up Nginx, configuring it to a basic level and then configuring a certificate and HTTPS so that we can connect to it securely.

Web servers are very common of course, and anyone can set one up, but doing it correctly without leaving doors open for attackers is a challenge! There are common mistakes, like overly broad user permissions used to fix issues, which can allow for compromise. We will review best practice for setup in this module.

## What are web servers?

Web servers are highly desirable targets to hackers for many reasons. Let's explore why they are such attractive targets.

Web servers are designed to take files or data and present them to a user over protocols such as HTTP. A simple function, but over time web servers have grown more complex and added support for new protocols with greater efficiency and security. For example, HTTP/1.1 is still widely used and has been around for a very long time. It is an ASCII based protocol, which we understand from our earlier Foundations modules. On the other hand web servers often now support HTTP/2.0 which enables binary based transfer. This is harder for a human to read from the wire, but is more efficient and high speed at transferring data - the primary useful use case of a web server.

A web server is just an application that listens to the network on a specific set of ports, and speaks a specific set of protocols. When clients connect to it they are able to speak this protocol and request resources, for example the infamous ' GET / ' which retrieves the base or default page at the root of a website. Remember however that web servers are just applications running on an operating system. At their core there is little difference between a word processor and a web server, although it is normal for web servers to require slightly higher permissions to bind to ports like :80. On most operating systems lower ports below 1024 require elevated permissions to bind. This presents a quandary as the web server typically wants to minimise it's permissions on the system to avoid allowing an attacker to compromise it further in the event it is hacked. This can result in some interesting permissions segregation of the web server in to a higher and lower privileged portion of the application.

In the most simple use case a web server will need permissions to read files from the web server directory, for example /var/www/html on many Linux systems. It might serve the file index.html and have only read permissions on it. In more advanced setups the web server might also receive instructions to change files, and then require permissions to write to specific files also. This has to be carefully controlled however as overly broad permissions can make compromises simpler for attackers.

Web servers over time have grown to support dynamic applications and web sites too, not just serving static files, for example PHP or other interpreted languages. When accessing a file such as 'index.php' the web server could recognise the suffix PHP and use a defined processor, or the PHP binary, to process it and produce an output. This output is then sent to the calling user in place of a simple static file. This is clever as it allows for dynamism like including a users name, or responding to data provided by the client.

This dynamic functionality also allows for the very common use case of a web server combined with a database. A database might be used to keep customer records or usernames and passwords. The database server may reside on the same system or may be a network based resource that the web server connects to. One of the most common setups is referred to as the 'LAMP stack' which stands for Linux, Apache, MySQL and PHP. In

this instance Apache is the web server which runs a PHP file that contains instructions to connect and query a MySQL database.

You can see, I am sure, why this makes web servers an interesting target for attackers. Let's review some high points on why they are interesting:

fi. They are exposed publicly or designed to be connected to, making them a good place to connect through a firewall.
fi. They can have permissions to access more files than they should, and potentially enable lateral movement.
fi. They are often connected with dynamic capabilities provided by PHP or other interpreters, which can allow an attacker to supply their own code or modify functionality.
fi. Other users might visit the page, which means that they can be compromised and used to target other users. This is a very popular malicious code distribution mechanism.
fi. They are often connected with databases. Stealing a long list of usernames and passwords, or emails, or credit card details (in the worst case) allows an attacker to get a treasure trove from one place!

Note that it is not just about compromising the web server. An attacker may be able to use this to get to the operating system, or other applications and slowly work their way back in to the network. A common problem is when developers leave useful code artifacts or code repositories keys on a web server believing they would never be found as they reside outside /var/www/html. Being able to connect to other workstations or systems nearby, or pivoting as we call it, is an attacker tactic we will cover more later in the course.

There is another sneaky reason why hackers may want to gain access to your webserver. If a third party hosts your web site, there is an excellent chance your website is not the only one on the server. Shared web hosting providers will often run multiple websites off of one web server. Perhaps a hacker can hack a website with little security to gain access to the server and then compromise other websites. Imagine being able to hit a hosting provider and compromise one system, but then get thousands of sites to serve up malicious code! That would be quite the force magnification pay off for the attacker!

Web server security largely comes down to patching your system and being careful with configuration. There are excellent guides for this and using mainstream practice, not being too clever with unusual ideas, is a really strong way to up your security. We will cover this more in the sections that follow.

## How do web servers work

Websites can be created using various programming languages; however, it's first important to note what is meant by the front-end and back-end and client-side.

### Client-Side

The client-side is the device that is being used to connect to the web site. Many things happen here, such as inputs to be uploaded, and each time you click on a new page, you request a download. Perhaps you click on the latest article of the day of the BBC news web site, the moment you do, the web page you are looking at is no longer live. Don't believe it? Well, let's do a quick check.

### Quick task

You may find it is easier to use a different browser such as Firefox or Chrome here. Also, be sure to use a desktop browser.

Note: If you are on Safari, you will need to enable the feature. Go to the top left of your screen where the apple symbol is. Next to it should be Safari if you are currently on the Safari browser. Click that and go to preferences. Look for the advanced section at the top of the new window. Make sure the 'Show Develop menu in menu bar' is ticked. Once you have done this, it should work the same.

Step 1: Browse to sans-foundations.com and be sure to open this is a new tab so you can continue to read the instructions. When it opens the home page, be sure to be logged in so you can view your name. As soon as you click on this article, your devices ask the server to download a file that is actually the web page.

Step 2: Whilst you are viewing the sans-foundations web site, find the title where it says 'Good afternoon<your name>' (may say good morning instead). Right-click on the title and find 'inspect element' and left-click it. This should open up a little box at the bottom of your browser. The blue highlighted line of code will be the instructions necessary to show this title; click on the little rightward facing arrow at the beginning of this to expand the code.

Step 3: now, this is the fun part. you will see something like `<h1>TheNameOfYourTitleHere</h1>`. This simply means whatever is inside these tags, display it with the biggest title format. In between the tags, where the title is written usually in plain black text, change the title to whatever you like. Perhaps change the name. NB: you may have to double click the title to allow you to type in the box.

Step 4: Hit return and check out what the title now says!

The point of this exercise was to help you appreciate that you did not just hack the website. Once the HTML is served to the browser, it can be modified in the browser

without you actually altering the site running on the server.

**How to read a URL**

Notice that when you open file explorer on a Windows machine, it will show you a path. You may see something like this: C:\home\documents\Sans_Foundations_Doc. It is essential to understand what this means.

The C: is telling you which hard drive slot to go to, although these are not fixed. On that hard drive, there is an entire file structure with folders and documents or apps etc. So the home folder has, among other things, a documents folder and so on. Have you ever seen the uncanny similarity between this and a web site URL?

Take this URL, for example.

https://en.wikipedia.org/wiki/Backslash

Now, because you are not simply accessing your motherboard, you cannot just put "C:" into a URL. Clearly, the website is not on your computer. So you need a little help from your friend, the internet. The first part, HTTPS, tells the device how to write and send the request to get the document you are asking for (document = webpage). The next part is en.wikipedia.org; well, this is kind of the equivalent to the "C:" part now. It is saying where is this computer. So behind these words should be an IP address. Perhaps Wikipedia also writes this article in a different language, which this webpage may store on a completely different device (or hard drive). The next part is the wiki/Backslash, which is now exactly like the home\documents\Sans_Foundations_Doc. The web page or HTML document that is called Backslash is located in that file path.

So you see, there is nothing more to the WorldWideWeb than just file structures and hyperlinks.

The client-side is simply the web browser and your device.

**Server Side**

Although this is not strictly the case, for now, let's assume that the front end is the same as the client-side, and the back-end is the same as the server-side. There is no exact definition of these terms, but in the profession, there are some nuances.

This is where most of the magic happens. The computation happens here, and it is usually written in some form of scripting language. Whenever you see scripting language, you can assume this means it does something rather than just be pretty text on a page. Common scripting languages for web sites are:

- JavaScript
- PHP
- Python

- JAVA
- Perl
- ASP
- Ruby
- Go

This is not a complete list by any stretch of the imagination. It simply gives you an idea of some languages that are used to code functionality on a web site. Note: they are not represented in any particular order.

## Analytics on web servers

Analytics and instrumentation is an essential part of websites and servers these days. But how do they work? Analytics tend to be calculated and logged remotely. This means there is another server elsewhere that is storing all this information. Data is collected about the system, application and users interactions to be used for this purpose. There are multiple types of analytics and instrumentation tools. Some collect data from the system or application automatically. In other cases programmers specifically instrument their programs with code. Programmers use something called an API (Application Programming Interface), which allows the programmers to write a few lines of code in their software that will call another application with a query and receive a response. Analytics are instrumental because they tell the website owner how many connections are being received and potentially where they originate.

Web analytics generally measures user activity, such as how long they are on the website, where they go, how long it takes to get there and whether they came from a link or not. It helps a business track if marketing campaigns were successful and also helps them understand the traffic fiow throughout the year. Imagine if a substantial online retail store had, say, one thousand connections a day. They may find themselves with a crashed website around black Friday or Christmas when that number could increase overnight. However, if the website were to run analytics, it would be possible to know roughly how many more customers to expect at peak times throughout the year based on previous years. Their infrastructure could then be configured to handle the requests and be fiexible during extraordinary times.

Business systems often have multiple components of infrastructure connected together to deliver an experience to a user or customer. Today analytics and instrumentation are used to measure all kinds of attributes between these systems, and then learning or anomaly detection mechanisms are used to identify if something unusual is happening. For example, you might on average find that the connection between the web server and the database takes 124ms. You might find on average you can respond to web users requests in 0.3s. However, for a period of time connections from users in a specific country register at 2s. This can be used to raise alarms to create action, or in some cases drive automated response in an elastic infrastructure.

## Aw, snap!

Sometimes you will see these messages when you try to load a web page. You may well be familiar with the '404' error message for example.

### Informational response

100 Continue - Everything is looking smooth. Continue as you were.

### Success

200 OK - The request has been completed with no errors. This is a good sign on weblogs. It also varies slightly depending on what kind of HTTP method was used (GET, POST, HEAD, TRACE)

202 Accepted - Received but have not yet done anything about it.

### Redirection

301 Moved Permanently - Usually, the new URL is given in the response

### Client errors

400 Bad Request - Invalid Syntax error

401 Unauthorized - You are not allowed into this site without logging in. Unauthenticated.

403 Forbidden - Same as 401, but the server knows the client's identity. This usually means you don't have enough privilege.

404 Not Found - Uh oh. Nobody knows where that URL is. Maybe you typed it in the wrong way. You could try doing this now.

408 Request Timeout - The server is shutting you down for idleness

### Server errors

500 Internal Server Error - The server has no idea what to do, so it took a nap instead. This typically happens due to bad code or failure to handle data or an exception. These are often worth investigating!

502 Bad Gateway - Issue with the servers connection.

503 Service Unavailable - Overloaded server. Most likely because everyone is trying to buy a PS5 on the website.

504 Gateway Timeout - Sorry, your servers are too slow. Probably a connection problem somewhere in the network.

Some of these codes are very useful for hackers to know. It can indicate whether there is a firewall and how things are configured on the web server.

You may not necessarily see a 200 code. For example, the page would just load, but they are still being received by the browser. Let's have a look at response headers to verify this.

**Response Headers**

Response headers are a form of HTTP header that do not carry information about the content but rather the status of the request you have sent. They come from the web server themselves. The client sends an HTTP request, and the web server replies with an HTTP response. Instead of showing you an example response header, let's take a look at a real one.

**Quick Task**

You may find it is easier to use a different browser such as Firefox or chrome here. Also, be sure to use a desktop browser.

Note: If you are on Safari, you will need to enable the feature. Go to the top left of your screen where the apple symbol is. Next to it should be Safari if you are currently on the Safari browser. Click that and go to preferences. Look for the advanced section at the top of the new window. Make sure the 'Show Develop menu in menu bar' is ticked. Once you have done this, it should work the same.

Step 1: Browse to sans-foundations.com and be sure to open this is a new tab so you can continue to read the instructions. When it opens the home page, right-click anywhere on the page and click inspect element.

Step 2: When the inspect feature opens at the bottom of your browser, search for the tab at the top called 'Network'. You may need to refresh to view all the requests.

Step 3: There should be a list of requests that you can click. Choose any one of them and select it. Once you do this, you should see a new box pop up, probably on the right-hand side of the inspect feature. In this new box, you should see the HTTP request and response headers. Browse around them to see what information is stored in them.

After you have looked at the response headers, you should have noted that the response codes are also shown.

If you would like to do a stretch exercise here, see if you can view these headers in the terminal or command line. You may need to use a search engine to find out exactly how to do this, or you can review the video above.

## Quick Task

Note: Browser extensions for privacy and some adblockers might prevent this task from working.

**Step 1:**

Go on to bing.com and search for what is my referer.

**Step 2:**

Scroll down until you see https://anonymiz.com/myreferer and click on this web site. At the top of your page, it tells you where you have just been.

**Step 3:**

Now copy this URL https://anonymiz.com/myreferer

**Step 4:**

In the URL bar, type in amazon.com

**Step 5:**

Then, from the Amazon home page, go back to the URL bar and paste in the nullreferer web address. What does it say this time? It does not tell you that you have come from amazon this way.

**Step 6:**

Open up a private browser and repeat these steps. What happens this time?

You should have seen that even in private browser mode, it still knows from which web site you are coming. It is essential to know what private browsing helps you with and how all these tracking mechanisms work. You may like to try this in several browsers private browser tabs. You may not be surprised to find out that they can still see your referer, even on privacy browsers such as the Tor browser. Another valuable task may be to think about why this is the case, and how do you stop them from being sent?

This is how comparing web sites pass you on to the provider. So when you click on the insurance quote, the insurance company knows what you have searched to get there and entitlement to benefits such as two-for-one movie tickets.

## Nginx and Let's Encrypt Demo

In this walkthrough, we take a look at installing a web server using nginx, and doing some basic security configuration of it with Let's Encrypt, to get the server up and running.

## Web Server Practice Checklist

Depending on which web server you are using and the exact setup of your application, best practices can vary. That being said, there are some common failures and problems that we see when security testing, or reviewing IT hygiene failures, that you can learn about in this section. Keeping an eye out for these can help you raise the bar of configuration and reduce the chance of attackers taking advantage of your systems. It is advisable to regularly check with vendors of the technology too -- Apache and Nginx have new configuration options, defaults and bug advisory along with releases that should be periodically checked.

### Check over your default configuration

The default for most modern web servers is actually pretty good, and we could just as easily write 'be wary of changing the defaults'. This was not always the case and the defaults were pretty bad, but often the package managers and default builds on most modern systems will do a good job of bringing in a lower privileged user like www-data and configuring appropriately.

That being said, scalability options like the number of workers in Apache are often set low. Take a moment to review the defaults, and validate your setup is good out of the gates.

Remember too that the default setup does not enable HTTPS, and so all communications will be unencrypted when exchanging information.

### Validate web server permissions

Most default setups today will create a separate user and group like www-data, which should have overall low privileges on the system. However, as a part of troubleshooting, we terrifyingly frequently see 'permissions sins'. One example of this is upgrading the permissions of this user by adding them to another group on the system that has real power. This happens because a permissions error was thrown, someone read online to give it more power, and it fixed the problem! That means any future attack on the web server could lead to attackers having real rights! Another example of this is where administrators do something like:

```
chmod 777 /var/www/html
```

This indeed fixes permissions errors because now everyone can do everything. We should see restrictive permissions set for the specific user, which should be low privilege overall. The Linux modules in the course aid with that setup.

### Disable the banner/versioning

Depending on your webserver, Apache or Nginx will output their version number and some basic configuration information. For example, in Apache this is called the 'signature'. This information can be valuable to attackers as it can leak that you are running a specific version that may be vulnerable to a specific exploit.

**Using Checklists**

Vendors publish specific information and checklists that you can use to secure their webservers, as well as looking at resources like the CIS benchmarks to further restrict them. It doesn't take a lot of time just to double check the configuration and validate you have good hygiene. In fact, a lot less time will be spent checking configurations, than handling an incident later.

# Database Servers

## Introduction to Database Servers

A database server is a computer system that provides other computers (or programs) with services related to accessing and retrieving data from a database. As software, it is the back-end portion of a database application, following the client-server model. Database servers are used to store and manage the databases stored on the server, and to provide data access for authorised users. Database management systems, or DBMSs, often provide this server functionality. We will be using MySQL, a common DBMS, later in the module.

Different types of users will access a database server in different ways, depending on what they're trying to achieve. The "front end" displays requested data, and will run on the user's computer. The "back end" handles tasks such as data analysis and storage, and runs on the server itself.

The initial conception of the Database Server started out with the introduction of Data Structure Diagrams (DSDs) by Charles Bachman in 1969. These diagrams provided a means to graphically represent relationships between data entities, which formed the basis of Codd's Relational Model for database management in the 1970s - the main principle of which is that users of a database shouldn't need to bother themselves with its inner workings - that most databases still use today.

Further research and development in this area in the 1970s focused on this model, culminating in Peter Chen's proposal of the Entity-relationship model, an abstract data model defining a data or information structure that can be implemented in a database. This model was recognised as being more applicable to the "real world" than its predecessors, and consequently became the most frequently used model to describe relational databases.

So, what does that mean for us? At the time of this writing, there are over 300 recognised DBMSs in existence, largely operating on the same basic foundations as those being researched in the 1970s. The demo in this module will focus on MySQL, a freely available open source Relational Database Management System that uses Structured Query Language (SQL). SQL is the most commonly used language for adding, accessing and managing content in a database, and its popularity can be attributed to reliability, speed of processing, and fiexibility.

Whilst SQL is known for being simple to use, there are many things to consider when setting up a database server effectively and securely. This module will guide you through the configuration, and give you the chance to try it out for yourself.

## Logic and databases

Logic is required to design databases to ensure that all statements are complete. We need to do this for many reasons, but mostly so there is no redundancy, making them efficient to use.

The goal is to create a database that is well structured and accurately represents the data which it stores. Primarily, databases are used by a company to store information about their business crucial for their day to day dealings. This is the reason why the database needs to be a good representation of the company.

Not only will all the information be logged in these tables, but the database models the relationships between all the data. For the company to become more efficient and to potentially grow, the databases need to make it easy to understand relationships between data. This means they are crucial in business development and analytics. Of course, there would need to be some form of engineer sorting through all the data in the first place.

An understanding of logic helps us structure databases better, helps us write more efficient query statements and think through the relationships. Good database design can make for extensible business systems and easy to extract data. Bad database design can function but make every task very difficult, and future upgrade paths painful.

## PHPMyAdmin & Adminer functions

These tools provide a graphical frontend to manage the contents of a MySQL database. For example, you may want to back up your database and with these tools this is easily done.

Firstly note that they are written in PHP, which means they integrate nicely with a web server.

It provides a friendly user interface to make tasks simple. It will also allow you to:

- Run SQL commands
- Export the tables into different file types such as a pdf
- Search the database
- Create, copy and drop tables
- Maintain the server
- Administer multiple servers
- Addminer allows the use of CSS skins

This list is not exhaustive. The database admin can manipulate and control the entire database through one of these tools. It should be noted that these tools should never be used on a production database or if they must be used you should only leave them up for as long as you need them. An exploit in one of these tools could lead to the compromise of the database it connects to.

## SQL Server Demo

SQL servers are the most widespread form of database server, and their relational structure and intuitive language make them compelling choices for many developers. In this walkthrough, we step through the installation and configuration of MySQL on a Linux system. We will undertake the following steps:

- Installation of MySQL
- Basic security configuration
- Create a simple database, table and query the data
- Load a backup of a substantial reference database
- Query and modify the database, exploring relationships
- Configure a user with specific rights for a database
- Configure a graphical database management tool to connect
- Export a backup of selective data and tables

The procedure will vary across versions and platforms, but many of these concepts remain unchanged over many years, and will be valuable whether you are in offensive security and finding flaws, or defensive security and locking down unauthorised paths to data.

## SQL Server Setup Considerations

There are, of course, a variety of different SQL servers out there, and specific best practice and security configuration will depend on which you are using. That being said, a lot of the guidance is standard, and you can find wonderful setup guides, like the CIS guides, to help you lockdown appropriately. Let us review a few key considerations to get you started:

fi. Validate the IP connectivity rules. This is a particularly common mistake for SQL servers running in cloud instances, like on AWS. It is set to make them accessible to the open Internet, allowing port 3306 (for MySQL) to be connected to by anyone in the world. Granted, password protection should prevent anything bad from happening, but a vulnerability or configuration error could allow for huge database leaks. It is best that the access is locked down to those that need it, e.g. the web server it might be powering.

fi. Remember to run the `mysql_secure_installation` as we covered in our demonstration. This will step you through removing the test database, anonymous accounts and enforcing credentials on the root user. This improves your security instantly.

fi. Validate that MySQL is running with an appropriately privileged and restricted user. Most default setup packages on modern Linux will do this for you, but it pays to check, as you do not want your database running as root. A specific lower privileged mysql user is a really common and useful practice.

fi. Consider removing the mysql history file. Typically located at `~/.mysql_history` this file helpfully logs what you have typed in to the mysql cli. However, it could expose credentials or configuration to users of the system!

fi. Ask yourself if you need remote logins at all? Is your database running with a local web application that can connect over 127.0.0.1? Do you need it bound to 0.0.0.0 and accessible more broadly? If not you can reduce your attack surface area generically with this configuration change.

fi. Conduct MySQL verb restrictions. I know this one sounds odd but here is a good example - limit the use of `SHOW DATABASES`. If an attacker has less information to work with it is harder for them to exploit, and even more so for automated tools. You know your database names for your applications, and have them configured in connection strings. Attackers may not and this can make it much harder for them to identify them depending on where they attack.

fi. Update. Update. Update. This will help you eliminate security vulnerabilities but remember that each patch might come with changes to how SQL works and the language specifics, which means you might also need to make changes to your

application - or update web frameworks. A good test environment where you can validate changes really helps here.

fi. Go check the Linux file permissions! Check that my.cnf is accessible from the root user only or your administrator user via sudo. Make sure that the MySQL data directory (typically `/usr/local/mysql/data`) is accessible for the mysql user but not other users on the system.

fi. Consider use of a managed SQL platform that does a lot of this for you. Amazon RDS or services like Aurora can automatically apply updates, restrict access to the backend system and simplify operational and security ownership. That does come at a cost, and you still need to configure credentials and the service appropriately - but it can reduce the work of server ownership significantly!

fifi. It is not allabout security. Have you considered the size of your database? Have you allocated only 5GB to the virtual volume, which will shortly be exhausted by your use case and cause the system to completely implode? Make sure you have adequately sized your system to the requirements, and configured monitoring so you know if disk space is getting low!

MySQL and CIS both have excellent guides of configuration you should consider, but apply what you have learned so far. Keep permissions tight. Users and roles specific and software updated. It really does make a difference!

## DNS Servers

## Introduction to DNS Servers

DNS, which stands for domain name system, is the phonebook of the internet, and a DNS server is responsible for translating typed domain names into numeric IP address - basically, they translate a language that humans can understand and remember into a language that computers can understand and process.

Abstracting DNS reduces to two types, authoritative servers and client-facing servers (for example, Google's DNS that recursively resolves requests by pivoting around the DNS infrastructure).

DNS is needed because humans cannot remember IPv4 octet addresses. This gets even more complex in IPv6, given the longer addresses. Therefore, a browser allows a user to input a human-readable domain name such as microsoft.com. There is a need to resolve this domain name and translate it into numbers for the computer to read.

We do not remember peoples phone numbers, so we store their names and look up the numbers. We do not remember people by their numbers and look up their names. Although in DNS, this is called reverse lookup, and there are many reasons why you would want to do this. It may be an excellent task for yourself to think of a few of these reasons.

There are four types of DNS servers, which, in a typical DNS lookup, work in harmony to deliver the IP address.

### Recursive resolver / DNS recursor

This is the server type that is responsible for receiving queries from the client machines via applications, such as internet browsers - this includes the input of a user typing a domain into the URL bar.

### Root nameserver

The root nameserver is the first port of call for the resolver to query. It responds to the resolver by redirecting it to the TLD nameserver that stores the information for its domains. There are 13 root servers distributed strategically around the world, acting as an index, or reference, for locating the IP address for a site host.

### TLD (top level domain) nameserver

The resolver then queries the provided TLD server, which will respond with the IP address of the domain's authoritative nameserver. TLD nameservers differentiate websites that end in .com, .net, and .org.

### Authoritative nameserver

The authoritative nameserver will retrieve the specific IP address of the origin server for the provided web domain name, which the resolver will pass back to the client.

Once a query has been passed through all four server types, the client can initiate a query directly to the origin server of the provided web address, that has been provided, which will respond by sending data to be displayed in the web browser.

It's worth noting that this is the process that takes place the first time that a particular DNS query is made. The resolver will then store the origin server's IP address in its cache for a certain time period (or until the cache is deliberately cleared), so that any further requests for the same domain don't have to be sent through the same lookup process.

In this module, you'll get the chance to walk through a simple DNS configuration, setting it up from start to finish, so you can see how this all works in practice.

## Theory vs Practise

When a user inputs the domain name into their browser, for example, microsoft.com, there is something called a recursive resolver that will query around the global DNS servers to determine where this website address is. The recursive resolver will go to the hierarchy of name servers, the first one being root, and they will work backwards and pass the resolver onto the next level down to the address. So in the example of microsoft.com, the root server will either know the ".com" or know which sub server will know. The root server then gives a list of servers that are the next level down in the hierarchy called top-level domain servers. These will then know perhaps the "microsoft" part of the URL is, but maybe not exactly where the specified resource is in that domain.

It then passes the resolver down to the next level called the bottom level domain server. This bottom level server will know what the exact address is for the resource requested, and it will send it to the recursive resolver, which gives the IP address back to the user. In theory, the resolver should only require three rounds of challenge responses; root, middle and bottom.

In practice, the recursive resolver still queries each level of the domain hierarchy and is satisfied by the end of it. However, the crucial difference here is that when the root server passes the records of the servers below, they will append the resolver to a handful of records, perhaps five. The recursive resolver then sends all the top-level domains a query simultaneously, rather than one at a time. It is already apparent that this can lead to generating lots of traffic. Furthermore, when the TLDs respond, they also give a handful of records for the bottom level domains, further exacerbating the problem.

The servers return a handful of addresses rather than just one because users want the internet to be fast, so the resolver sends queries all at the same time to get back responses as quickly as possible. However, there are other reasons, such as IP addresses are constantly changing, and new things are continually being added. There may also be many people using the internet simultaneously requesting from one specific server by coincidence, so it may take longer to respond.

Here are the two crucial points. Firstly, higher-level servers send to more than one sub-server. This means that the higher-level DNS server will send the addresses of several lower-level DNS servers. Secondly, The recursive resolver will send all the server's queries at the same time. This gives rise to the so-called 'amplification factor'. It is easy to appreciate how it would be possible to attack this system effectively.

It is important to remember here that theory and practice are usually quite far apart. When something like this happens to the global DNS system, the consequences are pretty drastic. So as you can imagine, root DNS servers need to protect themselves well as well as all other DNS servers. Luckily, the root servers themselves are decentralised and distributed. It isn't just 13 root servers; although we only see 13, they are actually clusters of servers worldwide.

# What can a DNS server do for you personally?

When you change your DNS server from the default, which is usually your ISP, all that happens is your computer will now send requests via your ISP to the new DNS server. However, this can give you some benefits. Your ISP may block access to specific sites using DNS. It is one of the most common forms of content blocking. If your ISP uses this type of content blocking then one way to bypass this is by changing your DNS. Change to another DNS server and you will get 'real' answers instead.

Another major benefit of changing your DNS servers from the default is speed - often your ISPs DNS servers are slower than other major providers such as Cloudfiare, OpenDNS or Google DNS.

So does this mean your ISP cannot see what you are doing? Of course not. They are still the ones that will be routing your traffic and by default, DNS traffic is unencrypted. It takes more than just changing your DNS to give you anonymity.

On the Internet there are a number of providers such as Cloudfiare which provide DNS infrastructure for you to use that tries to apply special rules or filtering. Some enterprises do this too and redirect you away from known dangerous resources. Imagine you try to access `www.areallybadsitetobeafraidof.fake` which is known to contain malicious code? Well, if you are using one of these DNS providers that site could have an entry for a 'jail page' that warns you about the site instead of actually taking you to the IP address of the attacker. In this sense DNS can be customized to protect you.

You can also fiip this logic on it's head however! Think about how an attacker could redirect you to other places if they controlled DNS. In fact, I've often said 'if you control DNS you win' in penetration tests. Someone could access a bank website or trusted corporate page and instead find themselves at a site packed with exploits. It is not quite this simple as there are other controls like HTTPS and HSTS that might stand in the way. However, these do not apply unilaterally and the owner of your DNS infrastructure controls basically everywhere you go!

## Walkthrough DNS Setup

Here, we're taking a look at a simple DNS configuration, and walking through the process of setting it up from start to finish.

## Are you secure?

By default, DNS is not encrypted, which means anyone in a position on the network where your traffic passes through can intercept the request and send you to the wrong website.

For example, you might be connected to the Wi-Fi at your local coffee shop. Others on the network might be able to 'sniff' the network traffic of the wireless network, see your DNS requests, and learn about where you are visiting. This might also leak useful information about software you are running when it updates or phones home. Just because they can sniff it does not necessarily mean however that they can redirect you to whichever website they feel like.

You might also connect to that coffee shop Wi-Fi and a devious party might control the Wi-Fi router. In this instance your traffic passes through the router and your DNS is provided by it. That means that when you want to know where to go to find importanttrustedsite.com it would ask the devious Wi-Fi router. They could send you somewhere else instead. That is why connecting to random wireless networks can be problematic, you do not know whether you can trust them to send you to authentic places when you ask!

Typically this threat is localised as the attacker actually needs to be in the path of communications. More on that later in the course, but for now, it should be noted there are some forms of DNS that use encryption. This is not a panacea but it can help. Let's explore them in the following sections.

## DNS over HTTPS vs TLS

By default, DNS is sent in plaintext, which allows it to be monitored by anyone in the path of transit. There are different reasons why people want to use encryption, but it is essentially a privacy problem that leads to a security problem. It is also a global problem, given that governments tend to use DNS to censor the internet. Attackers can also use this for many reasons, but predominantly, it can be used as a man-in-the-middle attack or simply to gather information about a person or organisation.

The basics of securing DNS is to encrypt it first. Without encryption, it is like sending a letter through the post that has no envelope. Encryption is a way to put an envelope around the letter to stop it from being read by 'the network'. However, this is not an exact analogy because people who intercept a letter can easily open the envelope and replace it with a new one or use other tricks like steaming the glue to open it easily, then resealing it. This would be hacking the post, by the way. Hacking does not only apply to electronics.

This is where encrypted DNS comes in.

### DNS over TLS

DNS over TLS, or DoT is a way to encrypt DNS using TLS! Ha, the name is actually very descriptive, unlike many terms in security! In DoT the DNS sits on top of UDP, which is simply wrapped with encryption via TLS. This provides a mechanism to hide the contents of the query and also makes tampering detectable, as the encryption will be corrupted through modification.

### DNS over HTTPS

DNS over HTTPS is an alternative to DoT. It is known as DoH, and you can probably guess why! In this instance DNS queries are sent over HTTP or HTTP/2 instead of over UDP. This provides the same features as DoT and makes sure an attacker can't see inside the requests or responses, as well as protecting them from tampering. This has become an increasingly default protocol for many browsers and users.

### Wait, isn't HTTPS just using TLS?

Well, kind of. It is true that in the end both DoT and DoH both use TLS to encrypt. One of the big differences here is the protocols involved and the port numbers. DoH uses port 443 which is a standard port used by most websites for secure web connections. That means it is very often available. DoT on the other hand uses port 853, so you get a dedicated port for visibility and network filtering, but it may be more restricted. They both perform a very similar functional role and aid us with the issue we are describing.

### What about DNSSEC?

DNSSEC is actually more related to DNS root server lookups, and how authoritative name servers correspond with DNS resolvers. Whilst the name sounds like a full DNS security suite, DNSSEC is more focused on problems like DNS cache poisoning. It is not used to encrypt communications or solve the problem of interception or monitoring on a cafe wifi connection. We will discuss it more in the next section.

**Which is better?**

It all comes down to firewalls. DoT uses port 853, and to a network administrator, it is self-evident what this is, an encrypted DNS request. The plus side of this is that the network admin can monitor DNS requests to identify malicious traffic. Although some companies may want to see all the DNS requests in plain text, so in that case they can block port 853. However, if your concern as a user is privacy, it is arguably better to use DoH because all requests are sent using regular HTTPS traffic. If the network administrator wants to block these requests, they would have to block all traffic on port 443. Therefore, users will blend in with regular website traffic fiow, and the network administrators will have much less control. There is also an extension to DoH called oblivious DoH which would effectively route the traffic through a proxy server, so the DNS servers do not know who sent the request. Hence, more privacy for the user.

Note: Although the DNS requests are encrypted in transit, you still need to trust the DNS server because they still know what domains you are requesting. If you query an evil attacker DNS server you will have a very secure connection to the attacker as they feed you bad data and send you who knows where!

## What is DNSSEC

DNS security issues do not stop at the cafe scenario we described previously. There are also issues that have arisen due to the fact that DNS was designed without all of our modern uses cases in mind. With the criticality of DNS in directing network traffic, cyber criminals have found ways to take DNS offiine, trick DNS servers into caching and serving bad entries, or hijacking domains. The DNS Security Extensions (DNSSEC) are a security protocol focused on these issues.

Domain Name Systems Security Extensions verifies the DNS servers' identity in communication with the DNS resolver. It does this by allowing them to sign the DNS responses digitally. If this is implemented at every level of DNS, then an attacker can't turn up in the process and try to convince you it is authoritative for something it does not own. The signature can be used to validate that the response comes from the right server and that it has not been tampered with.

Consistent deployment is key here, and each DNS level must sign to protect the resolver as it communicates to a different server. This creates a parent-child chain of trust throughout the infrastructure. Here is an example:

fi. You lookup SANS.org, and know how to validate the root
fi. A root DNS server signs a key for the .org name server
fi. The .org name server would sign a key for the authoritative name server

Each level must participate in this way, otherwise a fall back to traditional DNS is likely and that opens up the possibility of attack.

As you can see this trust depends heavily on the root, and therefore at the top root level signing, humans are there to verify it. This is known as the DNSSEC root signing ceremony to sign the DNSKEY RRset. This makes it verifiable and trustworthy.

Note: This is a simplified version of how this process works. It is well documented and if you are curious beyond the concepts you can go read all about this. One of the benefits of this design is that it is transparently shared for all to read and challenge.

DNSSEC is often implemented with backwards compatibility, because achieving total deployment and support on such a widely used service is extremely difficult. It is also intended to be combined with DoH or DoT and in concert provide strong security for this crucial service.

# Log Servers

## Log Servers

Being able to log events of a variety of severity levels on systems and across the network is very important to maintaining good IT hygiene, but also cyber security incident response. Imagine this nightmare scenario. An attacker breaks in to the network and finds a common username and password of a service account on a selection of systems. We now need to identify where that account was used. We know a rough time frame of the breach and now need to evaluate hundreds of systems to identify if they were there, and what they did. Central logging enables us to immediately start an investigation and source potential impacted systems. Without this we could spend days trying to find out the scope of the investigation before we even begin!

A further dimension is that logging on local systems is inherently less trustworthy. An attacker that compromises a system could modify the logs to hide their presence or actions. If log events are immediately sent from a system to a central log server this is much more difficult to do.

This central logging capability extends into the cloud too, with a variety of products known as logging as a service (LaaS) platforms becoming more and more popular. These LaaS platforms provide pre-built ingestion capabilities, analysis and reports so you have to do less plumbing, and can focus on analysing of the logs instead. It is well worth taking a look at some of the major players in this space and the latest capabilities they are touting!

Combined with the virtualisation modules, setting up your own logging server with Rsyslog is a wonderful exercise for you to validate your understanding. From here we can also talk about commercial products as well as SIM, SEM and SIEM and how they are enabling teams to respond to data in their networks about incidents and events.

## Basic Log Server Setup

In this walk through we setup a basic Rsyslog server to receive log information from the local system, and the broader network. There are a myriad of options to configure how you collect data, and how you structure/store it. We demonstrate here configuration of a network Rsyslog server and have a client send information to the remote logging server. We also demonstrate building a database to enable easier query of the data.

Tools like Rsyslog enable very high performance ingestion of alerts, at over 1M records per second. You can ingest the records from a variety of sources, filtering by application type and severity. You can then store the data in all manor of databases or services, even piping it out to Logging as a Service (LaaS) providers to ease your analysis!

A fun note on syntax and the old vs new way to declare templates. The $template statement is the legacy way to declare a template. It works well, and if you have formed a habit like me, you keep typing it. The new way is to type template(parameters).

Combined with the virtual machine portion of this course, this is a great opportunity to build a small network with logging as a challenge!

## SIM vs SEM vs SIEM

First of all, let's clear up some confusion about acronyms that you may come across. You may have come across SIM, SEM, and SIEM. SIM means Security Information Management, SEM means Security Event Management. Combine the two together, and we get SIEM Security Information and Event Management. So what role do they play? Let's break their value down a little further.

Simply put, they refer to a type of tool that will store and process logs or event data. The quick difference is that SIM deals with storing log data for long term analysis, whereas SEM is concerned with live-action processing. Combined and you get one neat tool that deals with both, SIEM.

### SIM

SIMs are tools or platforms used to collect and store all the security data that is logged within an organisation. There is a considerable amount of data ranging across the entire IT platform. Depending on your technology choices and how well configured the collection process is, they would collect data from things such as:

- Web servers (if in house)
- Firewalls
- IDS and IPS logs
- Router information
- Terminal commands
- Changes to an employees computer
- Antivirus logs
- DNS servers
- Authentication servers
- Database access

Basically, all the information from any software and how it's being used, including changes to the operating system. This could get very big very quickly! It is therefore not uncommon for collection to be a little more selective. All of these stored logs amounts to far too much data for any human to process. Enter SEM.

### SEM

These tools provide real-time analysis of all the SIM data with the added benefit of sending alerts about any security anomalies. Imagine combing and sorting through all the data listed above and then parsing it. This would then output to a console so that a human gets an alert to be investigated. This can be used to trip alerts, but also provide bigger picture insights in to the network and trends. The console tends to have displays such as charts and pictorial representation of the data to make it easy for the network administrators to see what is happening. Log data can provide an invaluable insight into what attackers are doing on the network. The ability to analyse this data quickly as

opposed to in 30 days, could make the difference between stopping an attacker quickly, or giving them significant dwell time in your network!

**SIEM**

Rather than having two separate tools that do practically the same thing, why not combine them into one platform? This is precisely what a SIEM does and has become quite the default. These SIEM systems automate much of this process and come pre-packaged with rules and fiows designed to make processing data and reacting to it much easier.

This integrated platform is useful to security, but can also be very beneficial when auditors come and check how compliant the company has been. This is for things like Information Security management systems and ISO 27000 certificates. Demonstrating control over your logs and understanding of events in your network is key to such processes.

As mentioned before you can process a huge volume of data with these systems. Simply imagine every endpoint streaming continuous log data about a given software update or configuration change! Retention policies to limit what matters and filtering rules that focus on the data of consequence is key, or data can become an absolutely gigantic monolith.

In most businesses there are pretty typical things you want to log, so there are default setups to arrange collection for common technology or platforms. However some configuration is often needed. SIEM tools do often come with a set of rules or triggers, but customising them for your environment is typically required. SIEM platforms are not just installed in the network, they can be cloud based too! In this setup you can benefit from management by a vendor, and continual software updates or features, but there can be a trade off of handing your event data to a third party.

These platforms include more and more clever techniques to react to data, and baselining with machine learning or heuristic mechanisms is finding more suspicious entries than ever before. In short, this means that if a hacker wants to get through the network, they will have to look like regular traffic.

A great SIEM setup will help organisations react to breaches, but also understand what happened and investigate retrospectively.

## Free tools vs commercial

There are many vendors out there, and if you want to take a look at a very popular product in the commercial space take a look at Splunk. There are other projects which are open source such as Apache Metron that evolved out of the Cisco OpenSOC platform. Metron uses other Apache projects such as Kafka, and analysis via tools like Apache Hadoop. It has a very extensible architecture but does not have the easy out of the box coverage of a commercial tool.

Some platforms are available both free and as premium paid products. SIEMonster is a popular example of this. Different products have different strengths and weaknesses and different capabilities, so it is not a case of a clear best. There are very strong open source options available to you, but of course a commercial product has the benefit of support, which given the importance of this data is a path most companies elect to take.

### ELK vs SIEM

This is a great moment to touch on ELK vs SIEM. ELK stack combines Elasticsearch, Logstash and Kibana. It is a very powerful stack with massive capabilities on the processing and querying side. This platform can grab logs, process them and store them. There is also a query capability and you can build dashboards. At the outset you can use ELK with some tooling to get towards an SIEM solution. However, out of the box you will find alerting and correlation capabilities missing, which make it fall short of the broader SIEM definition that the commercial tools offer. Whilst it may not be a full SIEM that does not mean it is not extremely useful.

One example of the utility of this platform is SOF-ELK. This "big data analytics" platform is optimised towards security operations and forensic investigator roles. It was developed by Phil Hagen originally for a SANS class but has grown in to broader use. It is well worth a look to see the power of data collection and query from the stack!

## Security vs Privacy

So far we've been talking about collecting all this data and the benefits it has for finding attackers and raising the alarm on potential attacks. But infinite collection of data, aside from the size problem, also has drawbacks.

The battle is simply a trade-off in modern-day security. Heuristics based security is extremely popular because it works very well. Think of this as a form of behavioral analysis. Imagine you let someone into your house to watch you all the time for security reasons. Not only would they be allowed to watch everything you do, but they would also need to write notes and figure out all your patterns. When you come home and do something that is out of the ordinary, the security guard would know something is wrong straight away. Perhaps you bought your laptop five years ago and have been using it every day during standard working hours, and then you switch it off.

Let's also assume you may be a typical user on the laptop, and you do not tend to open up command prompt or terminal. One day your computer wants to run a program that opens a command prompt during the early hours of the night. This behavioral based software would block this software from running straight away since this seems like an extreme anomaly. With 'AI' (this is really more marketing from vendors, it is essentially a profiling and learning model!) software that learns your behavior and watches everything on the network for your security, it is impossible to have privacy. It is important to note that many models are not trained on the computer but require the data to be sent to a remote server to be processed. This is an obvious cause for concern with regards to privacy.

Whether you are using a more clever behavioral profiling approach, or just conventional log collection, there is a balance of how much oversight and intrusion users might accept on their devices. In the most strict trade environments this might be acceptable (though there are other issues with the logs potentially containing sensitive data that needs to be purged!), but what about if a user brings their own computer? Can you really go logging all their website visits legally or ethically?

This problem can also extend in to tidying up data based on the rights of users. For example you might collect web server logs of interactions with your application. Superb! Then a user comes along and invokes their right to be forgotten under GDPR and you must purge entries about them. This is easy in a SQL database as you can identify them and remove the rows (at least in a well designed database). However, what about all the entries about them in the log files? Is there an easy way to identify them? What if the log collector accidentally grabbed a load of data about them and stored it in some other event? It could get really difficult to comply with the law.

This can lead to a direct trade off of security vs privacy. More monitoring can lead to better attacker prevention, but can also leave you open to legal or ethical constraints. It is a balance, and one that has to be owned and worked on by security teams.

## Log server best practices

Logs are invaluable to investigating attacks, but also to compliance standards or forensic investigations. There are lots of types of log servers and collectors, so truly generalising is hard, but there are some golden rules to think about in any implementation. Naturally, you will want to review the specific best practice from a vendor or tool you chose as well.

fi. Do not start by logging everything! It is tempting to just start collecting as much as possible but this can make it very difficult to size your data collection and tune your rules. You may also identify how you want to segment your data early on, or build multiple roles for different reviewers. Build up, not collect everything and reduce back down.

fi. Consider compliance as well as security early on. You may need to change your setup and data you collect or DO NOT collect in accordance with them. What about GDPR, or HIPAA? You should know what impacts your organisation up front and change your rules.

fi. Consider collection efficiency and safety. You do not want to get in to accidentally logging data that contains passwords or other PII, that would generally introduce additional risk for your systems and the users data. Make sure you are collecting logs where this data will not be present, or specifically exclude these events.

fi. Consider key business and security events. Authorisation success and failure attempts are particularly important, and do you have them being collected at the major service boundaries that matter? What about changes in user rights, or errors from key applications? Consider particularly high privileged users in terms of data access or rights and how they are logged. Thinking about the attack surface area can help you build a good checklist for your logging systems too.

fi. Challenge yourself on the retention strategy and period. How long do you really need all this data? Can you keep just one category for an extended period, and archive less important data after a period of time to save storage? Are you under a legal obligation to keep it for longer?

fi. Run a test! Try and create a mini incident (though ideally make sure people know it is coming) and see if it turns up in your log process. Does an alert get generated? Are the right people notified in the way expected? A quick run of something like the EICAR test virus is one way to do this.

fi. Consider data transport security and encryption at rest. Does the data contain sensitive information? More than likely you don't want observers on the network to be able to casually read your log entries and events! How is the data transported safely?

It is also important to validate your log server is operational as expected. They should be regularly interacted with or they can sit in a corner slowly filling up with cruft. A malicious

event hidden in a log server surrounded by a million other events with no alerting is not much use, though at least more useful for retrospective incident analysis than NO logs!

# Email Servers

## Introduction to Email Servers

How does an email message make its way from a sender to a recipient in a matter of seconds, when both parties may be on opposite sides of the world? Introducing the email server.

An email server, also known simply as a mail server, is a computer system that sends and receives email using standard email protocols. Generally speaking, the SMTP (Simple Mail Transfer Protocol) deals with outgoing email messages, and the IMAP (Internet Message Access Protocol) and POP3 (Post Office Protocol) deal with incoming email. All these connections are dealt with by these protocols behind the scenes when you use an email client or webmail interface.

Many businesses and individuals use web-based email clients, which don't usually require the separate configuration of an email server, but businesses often choose to invest in their own servers, for reasons of space or security.

We all probably use email servers multiple times most days, and it's easy to do so without needing to know much about the processes involved. In this module, we'll walk through setting up your own email server, as it's highly useful to have a good knowledge of the inner workings when it comes to understanding how email servers can be compromised in the event of a cyber attack.

An understanding of email servers is of course useful given we all use email, but it is often the case that email is used as a delivery mechanism in attacks! You might depend on this understanding later in your career when it comes to analysing an attack, or re-creating an attack tactic!

A further dimension to email servers is that they are very often provided via cloud services, such as Microsoft Office 365. These SaaS models for email can provide lots of capabilities without having to manage hardware, but also come with an absolutely monstrous set of policy capabilities. What you gain in simple running and reliability, as well as functionality, can however be compromised in the ease of forensics and investigation if you need to extract specific data to understand an incident. This is a classic trade off of cloud services and SaaS versus on-site hosting. We would recommend students go and set up a Microsoft Office365 instance and experiment with the policy settings. They have very reasonable evaluation accounts and it can be an insightful guide in to server management in the cloud.

## SMTP, IMAP, POP3 and others

What are all these acronyms, and do they make a difference? Each of them are email protocols and you have likely interacted with them already today, but they serve different functions in the e-mail process. Depending on your host and the client or server you use, you might also prefer one versus the other. Let us briefiy just outline the role of each.

### What does SMTP do?

Simple Mail Transfer Protocol is pretty much the industry standard for sending emails. Notice the word sending! The important part of SMTP is the 'T' for Transfer. It is the protocol you use if you want to send an email and not to retrieve it. It is also what is most commonly used when transferring e-mail between e-mail servers. We say most commonly as sometimes alternatives are used within an organisational boundary, but when it comes to transmitting across the Internet SMTP is the default.

### What is IMAP?

This stands for Internet Message Access Protocol and is used for retrieving emails. IMAP allows you to read the message from the email server itself, versus SMTP which is used to transfer or send emails. When reading messages on the server via IMAP it is common to download the messages on to your device and synchronise state. One of the benefits of this in the default configuration is that multiple e-mail clients can synchronise with a remote server, and therefore your e-mail appears the same and up to date in multiple locations.

IMAP is great to retrieve e-mail, but you need SMTP to send it. They work together in most e-mail configurations and when you setup your e-mail you more often than not are providing IMAP and SMTP server details.

### What is POP3 for then?

POP 3 stands for Post Office Protocol version 3. It has a similar function to IMAP and enables you to retrieves your email. POP3 however is not built as much around the idea of synchronisation, so instead you tend to connect to a server and download the contents of the mailbox and then remove it from the server. If you were to access your email from your laptop and download all the emails, you could not sign in to your phone and retrieve them since they are deleted after download.

### Exchange, MAPI and HTTP

MAPI or the Messaging API, a proprietary Microsoft protocol, has a lot of similarities with IMAP. It is designed to enable the state synchronisation with a remote Exchange server so that a number of Outlook clients can be up to date with the state of their mailbox data. However in addition to this there are also provisions for contacts, calendar

synchronisation and other such features. It is built this way so that enterprises can provide the rich workplace management features that extend beyond e-mail. Do note that lots of e-mail providers also provide e-mail access capabilities over HTTP and there are native mobile apps which also implement their own synchronisation protocols. We will not dissect them all here but they often are an alternative transport for IMAP like capabilities. Thankfully these are very well documented online, and can be fun to take to with a network tool to observe how they work!

**Which one is better?**

It is not so much that one is better than the other, but that they have different features and you need to pick the right one for the job! IMAP and SMTP are a very common pairing, more so than POP3 these days for the synchronisation features. That being said Exchange and web based clients are also extremely popular these days. It is important to know which protocol provides which capabilities, as in a later forensic scenario you might be trying to figure out who sent an e-mail or when someone receive some malware to their mailbox, and on which device!

## What happens when you click send

Millions of emails are sent every second. So, you open your e-mail client, write an e-mail and then click send. What happens? Let us talk through a simple exchange, but keep in mind there are more complex versions of this when encryption or other more advanced sending features come in to play.

More than likely you are using an e-mail client, such as Outlook, or perhaps Mac Mail. This client will understand the protocol and steps that need to be taken to send this e-mail.

Firstly, the e-mail contents needs to be packaged up ready to send, and encoded. We will cover this in greater depth in a later section. The e-mail also needs to be packaged with information to enable it's correct routing and to be decoded the other side. There is something called an email header. That describes information about the email, such as who it is coming from and where it is going. It also includes information about the sending mail client, or mail contents. These headers are key to processing a message but also used as part of authenticity checks with spam filters (more on that later). These headers can also contain information about the origins of the message, for example where in your local network it is being sent from. Keep in mind, the email is sent to many places before it reaches the destination so having some information on the path it took can be helpful for troubleshooting.

An important step in sending an e-mail is finding the recipient. If you provide an e-mail such as supertestaccount@sans.org, the domain sans.org can be queried to find where the responsible server is to handle e-mail. These are called MX records. Let's take a look at one by asking DNS nicely for the mail servers:

```
nslookup
set type=mx
sans.org

Server:         192.168.0.1
Address:        192.168.0.1#53

Non-authoritative answer:
sans.org        mail exchanger = 10 mxa-002c1802.gslb.pphosted.com.
sans.org        mail exchanger = 10 mxb-002c1802.gslb.pphosted.com.
```

These entries tell us where to send the message ultimately.

OK so let's step through this. Your e-mail client connects to your provider to send the e-mail. It provides a packaged up version of the e-mail with these headers, and a destination e-mail. Depending on the configuration your provider server may reject this as it doesn't want to send e-mail for the whole Internet, and will usually rely on authentication to validate you are one of its users. Assuming this passes, your providers mail server will then be able to use the above information to connect to the mail server of

the intended recipient. The e-mail is transferred, and then the recipient's mail server will need to get it to the mailbox of the specific user. In effect:

E-mail client --> Your sending server --> Recipient mail server --> Recipient mailbox

There are several protocols at play here that we will cover in more depth in the following sections. Note however that each of these stages involves numerous protocols to validate how to send information, and to do it securely. DNS is absolutely crucial to this otherwise e-mails could not be routed to the right place and we would have e-mails fiying all over the Internet being intercepted!

**Summary**

Email messages go through a series of transfers on their journey from sender to recipient:

- Compose email, hit send - your email client connects to the sending server on your domain.
- Email client shares the relevant information (your email address, recipient's email address, and the message content, including any attachments) with your sending server.
- Your sending server processes the email address of the recipient, paying particular attention to whether or not it is on the same domain as that of the sender, as this affects what happens next.
- If the two email addresses are on the same domain, then no routing between servers is needed; the message goes straight to the domain's incoming mail server (IMAP or POP3) - this is the end of the journey for this email, and the message can be read by the recipient.
- If the recipient is on a different domain to the sender, the sending server identifies where the recipients mail server is using DNS.
- The message is usually then routed along a series of servers until it reaches the recipients mail server.
- The incoming message is scanned by the recipient's mail server. If it passes filters and other checks, and the recipient mail server validates the e-mail is for one of its users it will be forwarded to the mailbox of the user for them to download.

## Spam Filters

An attacker sends a spam message and when it reaches the client's server, it will more than likely pass through some security filtering services, such as a spam filter. Spam has many forms, but in general, it is either adverts from companies or cyber criminals trying to get you to click links with malicious code behind it, or perhaps just a good old credential harvester that tries to trick you in to handing over data. For example it may be a scam that tries to get you to reveal credit card information, send money or simply hand over your passwords or account details.

So how do spam filters work?

There is some magic behind this. Not literally, but many companies do not like the spam filters mechanisms to get out because they don't want spam creators to know how to avoid them. That being said, there are some prominent and well-known techniques.

At a high level spam filters may: * Examine the IP reputation data or the sender information * Look at headers for suspect fields or odd values that don't look like real clients * Share data between many users to find attacks that have been reported * Scan the contents of e-mails to find common hooks or attacks * Examine attachments, links and other parts of the e-mail to see if it identifies a suspicious next step

Scanning the e-mail contents for suspicious links or text is a fairly obvious tactic that a spam filter will use. Scammers are very cunning and will often write emails that try to evade these filters with just the right balance of seeming 'human'.

However, if spam filters are trying to find authentic e-mails why do spam messages often look so weird? Why do they have stereotypical bad errors in them which make us chuckle? Well, simply because they don't want time wasters. Some scammers write the emails so that only a tiny minority will fall for it - the people who fall for something so overt are much more likely to fall for the secondary phase of their scam. After all, they don't need many people to send them money to make it worthwhile.

### Spot the sender? IP reputation?

Spam messages tend to be sent from BotNets which are effectively massive networks of computers that have been hacked. These might be desktop computers, or more exotic and interesting devices such as Internet connected fridges, doorbells and more. If it has a processor and is connected to the internet, it can be part of a BotNet. The MIRAI bot was an example of this and makes for interesting historical reading. Using lots of legitimate devices that have been compromised is actually a tactic to defeat detection in of itself. Lots of spam filters look for suspicious IP ranges on the Internet that should not be sending e-mail, for example if a home user pool of IP addresses for an ISP suddenly turns in to an e-mail server it is suspicious. It may not block the e-mails but score them more aggressively. Hijacking the computer to send a burst of spam messages makes this reputation check harder. If the attacker has a larger number of systems they can also send

the spam in small bursts and avoid poisoning the reputation of the compromised device for much longer. All that being said, IP reputation is a very effective mechanism in modern spam filters.

**Evade content filters with encoding and variability**

Another crafty tactic used in spam emails is that they are sometimes encoded or use unusual characters or symbols to avoid matches. Spam filters will be looking for particular keywords or strings of text that might be suspicious, such as 'Bank XYZ: Reset your password'. By changing this to an unusual variation that a human might presume to be a typo or error they could avoid this automated filter. For example 'BAnk XYZ: Re3et your password'. Sometimes these mails have emojis and some backslashes that look like no one would click, but remember in millions of spams it doesn't take many to make money for the attacker. Here is an example with emojis:

Hi Th3re âœŒï¸ðŸ'€ðŸ™ˆðŸ˜œ

This h@s been a wonderful day. ðŸ'ƒ

I.really wish YOU have been ðŸµðŸŽªðŸ¤¹â€  â™‚ï¸ðŸ¥)

Wh3r3 Eva, you go... I h0pe it "s up <>

Yours sincerely

H@kr 101 xxx â¤ï¸â¤ï¸â¤ï¸ðŸ"žâ˜Žï¸ðŸ"Ÿ

In some cases these e-mails are not actually about getting users to hand over money or credentials. Some of these spam emails are checking to see if these symbols bypass the spam filters. Its reconnaissance for spam email creators, V3rY Cunn1ng@ ðŸ˜).

**What about faking a sender?**

Spoofing tactics are also used, such as manipulating headers to try and appear to have come from authoritative senders. Manipulating the headers to look like a specific sender could increase the click rate massively! If the e-mail appears to come from your bank, it is much more trustworthy!

So how do email clients stop this from happening? They use authentication mechanisms (such as SPF, DKIM and DMARC) on the headers to make sure they know where the header and email just came from. These can be validated to see if the e-mail came from an expected source. Spam filters also check if the mail links actually go to where they say they will.

**Does it work?**

Spam filters find and stop millions of messages every day! They are not infallible but they have huge value in preventing e-mail noise and allowing scammers free reign. Although scammers are catching on to a few of these tricks, it's a very worthwhile cat and mouse game. At the core of these filters however is a set of simple behaviours:

- Spam filters tend to get constant updates on blacklists of known bad senders
- Spam filters will get many updates a day on rules to identify text patterns associated with known campaigns
- Spam filters will feed suspicious samples in to a lab so that systems or humans can spot errors and update detection

The fiaw? What if the campaign is low volume and from a good reputation account? This kind of low volume targeted spam is much more difficult to detect, and often depends on the user or detection of the payload that follows. Spam filters really struggle with this using the tactics above!

**When spam is delivered**

If a message makes it all the way through the spam filters, the email ends up waiting for retrieval in your mailbox. The recipient logs in to check it and the spam is transferred to your device with a protocol like IMAP. You may then click the e-mail and depending on the configuration of your device load images, which come from the attackers server and tell them you viewed it! Perhaps you will click the link and hand over data too!

**Is it always too late once it is delivered?**

If a message makes it to your mailbox, that does not mean that the spam filter job is over. Lots of e-mail systems these days can actually identify spam campaigns *after* they have passed in to some user mailboxes. Based on configuration they can actually go and remove or quarantine these e-mails even though they have gone through the filter, to prevent more users from interacting with it. Of course, this doesn't help you if you already fell for the spam, but it is another nice step in allowing global spam filter updates to limit the damage caused by cyber criminals.

## Response codes

An SMTP response code is made up of 3 digits, and each individual number means a different thing. The first digit tells you what class of status it is. So, it may accept the command or have a temporary error and so on. The second digit will explain a bit more about what kind of problem is encountered. So, perhaps if there was a syntax error or a connection problem. The third digit is then the specific detail about what the code is. Let us take a look at an SMTP conversation and where the codes are used:

```
Server: 220 smtp.sans.org ESMTP Postfix
Client: HELO relay.sans.org
Server: 250 smtp.sans.org, I am glad to meet you
Client: MAIL FROM:<bob@sans.org>
Server: 250 Ok
Client: RCPT TO:<alice@sans.org>
Server: 250 Ok
Client: RCPT TO:<jimbo@sans.org>
Server: 250 Ok
Client: DATA
Server: 354 End data with <CR><LF>.<CR><LF>
Client: From: "Bob Le Hacker" <bob@sans.org>
Client: To: Alice Le Hacker <alice@sans.org>
Client: Cc: jimbo@sans.org
Client: Date: Tue, 1 Jan 2099 10:00:12 -0400
Client: Subject: Test message
Client:
Client: Hey hey
Client: How are you?
Client: TTFN, Bob
Client: .
Server: 250 Ok: queued as 19472
Client: QUIT
Server: 221 Bye
```

Note the numbers being used in response to client data, e.g.

```
Server: 250 Ok
```

This denotes that the requested action has been completed, and is the most common response. The following codes are some examples but not an extensive list:

220 SMTP Service ready. This means the server can process the following command.

250 Requested action taken and completed. This is the most prevalent response.

421 The service is not available, and the connection will be closed. It probably means your destination server is not reachable.

450 The requested command failed because the user's mailbox was unavailable. Try again later.

451 The command has been aborted due to an error from the recipient's server.

452 The command has been aborted because the server has insufficient system storage. Maybe you have overloaded the server by sending too many messages at once.

500 The server could not recognise the command due to a syntax error. This could be caused by filtering issues with your security software or similar.

501 A syntax error was encountered in command arguments. Caused mainly by invalid email addresses.

503 The server has encountered a bad sequence of commands. Usually an authentication error. The commands are not executing in the correct order.

550 The requested command failed because the user's mailbox was unavailable. Again, this can indicate no email address or SPAM.

551 The recipient is not local to the server. The server then gives a forward address to try. This is commonly used as a strategy for spam prevention.

552 The action was aborted due to exceeded storage allocation. The recipient's email was full. This could have been an attack, or simply you are sending files that are too big.

554 Delivery error: Sorry, your message cannot be delivered. This mailbox is disabled. Either blacklisted your IP or thinks the email is SPAM.

These can all be very helpful for diagnoses but unfortunately can also help cyber criminals identify mailboxes to spam, so some servers limit their response code range.

**Enhanced Status Codes**

This change was brought in to fix some of the uncontrolled growth of use cases, as denoted by the quote from the RFC3463 "SMTP suffers some scars from history, most notably the unfortunate damage to the reply code extension mechanism by uncontrolled use."

This organises messages in to a structure:

Class . Subject . Detail

Class can be for example "2", "4" or "5" The subject is 1 to 3 digits The detail can be 1 to 3 digits

These are documented rigorously online, but for example a class of 2 denotes a positive or successful message. A class of 5 on the other hand is a permanent error, where 4 is transient or temporary.

The subject field could be X.2.XXX for the mailbox status or X.4.XXX for network and routed related responses. Let us take a look at an example of an SMTP conversation using enhanced error codes:

```
Server: 220 testmail.sans.org SMTP service ready
Client: EHLO testrelay.sans.org
Server: 250-testmail.sans.org says hello
Server: 250 ENHANCEDSTATUSCODES
Client: MAIL FROM:<test-user-jl@testrelay.sans.org>
Server: 250 2.1.0 Originator <test-user-jl@testrelay.sans.org> ok
Client: RCPT TO:<test-user-jl2@testmail.sans.org>
Server: 250 2.1.5 Recipient <test-user-jl2@testmail.sans.org> ok
Client: RCPT TO:<veryfakeuser@testmail.sans.org>
Server: 550 5.1.1 Mailbox "veryfakeuser" does not exist
Client: RCPT TO:<remoteuser@helical-levity.com>
Server: 551-5.7.1 Forwarding to remote hosts disabled
Server: 551 5.7.1 Select another host to act as your forwarder
Client: DATA
Server: 354 Send message, ending in CRLF.CRLF.
 ...
Client: .
Server: 250 2.6.0 Message accepted
Client: QUIT
Server: 221 2.0.0 Goodbye
```

Let us examine one response code:

```
Server: 550 5.1.1 Mailbox "veryfakeuser" does not exist
```

550 tells us the action was not taken. The 5.1.1 tells us that a permanent issue has been raised (the 5). The first 1 tells us this is an addressing subject. The final 1 tells us the user is not found or the recipient is rejected. We can look this up in the standard or just search for it. Thankfully, we often get nice text that is quite descriptive in these conversations too!

There you go, SMTP is actually a very human readable protocol compared to many. This would be a lot tougher if it used a binary mechanism for example!

## MIME

An interesting fact about email is that the protocol is text characters only. Of course, you are thinking, but I always send pictures through email.

Well, technically, you don't!

So what is happening behind the scenes? MIME, or the Multipurpose Internet Mail Extension, provides a mechanism to be able to transfer non ASCII data as well as suggesting to receiving clients whether it should be displayed inline or as an attachment. In order to achieve this we need a set of headers to describe the content and also encoding mechanisms to get the data to transmit over the protocol. I suppose in some ways we had to hack the email protocol that was not designed for this use case and build in these features. MIME provides a set of capabilities to allow you to send HTML, images, audio and video in messages. In later times it has also become key to supporting some security use cases too.

### How does this work?

If you are looking at a MIME message you will know, as it tends to declare the version and use of MIME overtly. There are other tell tale signs too. Firstly, it has MIME headers such as the Content-Type header which might stipulate text/plain, or text/html. The more interesting is multipart, which enables multiple 'parts' of messages to exist with different encoding standards being used. This is useful if you have a message with an embedded image, but also a ZIP file attached. Another common header is the Content-Transfer-Encoding header which stipulates how the data will be encoded.

Since email only supports text to include files as an attachment they must be encoded. Typically MIME attachments such as images are encoded in base64 which produces a string of text like so: `VGhpcyBpcyBhIHJlZCBoZXJyaW5nLCB3aHkgYXJlIHlvdSByZWFkaW5nIHRoaXM/`. The email client then recognises this is an attachment and decodes it back into a file which you can download through the email client. There are other schemes used such as 7bit, 8bit, printed quotable or binary. It depends on the e-mail client and transfer mechanism. You can read all about these in the RFC for MIME, and thanks to being so widely used their extensions and use cases are also very well documented.

### What could go wrong?

One of the problems with MIME is that it has quite a few different features and headers. Unfortunately over time some clients have handled the data differently to a given server, despite pretty clear guidance in the standard. This can lead to occasions where a security scanner fails to parse the MIME and allows it through, but the client then renders the malicious content. This is becoming less common but is a constant reminder that

protocols for exchanging data and consistency across multiple applications interacting with a standard are not simple to achieve!

## Basic Email Server Setup

In this video we walk through the basic configuration of a SMTP server, using postfix. Postfix is one of many MTAs (Mail Transfer Agents) that has an SMTP server designed to send and receive e-mail. Others are sendmail, Exim and Qmail (to name just a few). These servers can be configured in a variety of modes - for example receiving e-mail and simply forwarding it on to some other MTA - a kind of upstream forwarder. They can also be local only, or configured to work just within a specific IP range in the local network.

In this instance we configure a functioning SMTP server that works with the network, but that sends email between local users on the Linux system. This is the basis of more advanced configuration where we can layer on webmail services that can be accessed with a browser. We can also bring in POP3 and IMAP which are designed to enable clients to get access to their e-mail, once Postfix has put it in the right place!

In this video you will also find a useful walk through of the SMTP protocol and the mail command. We will get more practice at this in the networking portion of the course but it is useful to see how SMTP was built up on trust. When these protocols were created there were far fewer Internet users and the notion a server would 'lie' about identity was unfathomable. That is sadly not the world we live in today.

Thankfully, numerous controls exist to thwart this type of behaviour. They have been layered on top of good old SMTP. These will be explored in greater depth later in the course.

# Synchronisation Servers

## Introduction to synchronisation servers

This isn't as specific as some servers we have discussed, because there are so many types of synchronisation servers. However, the concept is pretty simple. Synchronisation servers enable us to synchronise data between one or more locations.

fi. How directional are they? Do they synchronise in one direction like a backup or archive? Do they synchronise changes between one and many devices, like a file system?
fi. What type of data is being synchronised? Is it files on a file system, or entries in a database? This could be a replica of entries in a table, or email data.
fi. What is used to identify changes to replicate the data?
fi. How does the protocol efficiently transfer information? Does the system avoid transmitting all of the data again if just a small portion changes? Does it have a delta capability?
fi. Is the synchronisation continuous or triggered on a schedule, such as a cron?

Perhaps this service also spans multiple device types, such as moving data between your laptop device and your phone. This is an increasingly common capability for photos, files and all manner of other data.

Perhaps you have been working on a Microsoft Word document online where several people can edit at the same time with live updates? It is hard to generalise the capabilities here given each of these implementations work in specific ways, but the most important take away of this module is to understand the principle and common mechanisms used by 'sync' servers and services. We will later discuss ways in which they can go wrong, and some of the security features are often not considered.

Take a moment and think about all the places data appears to 'synchronise' between multiple users, devices, or platforms and how the system might be sharing this information efficiently. Is it a hub and spoke model where there is a central server? Or perhaps distributed and a clever protocol identifies the changes that need to be distributed? By this time in the course, you can likely start to piece together a picture of how this all works, and spending some time drawing out a theory of the mechanisms at play will benefit your ability to think architecturally. Have a go, then proceed to the next section to learn about one reference implementation.

## How do they work?

So, let's say you are collaborating with your team on a Word document. Please think of this as all the individuals simply connecting to one central device using something like remote desktop protocol and opening the document simultaneously. Each time they edit the document, the changes are saved instantly. It's a little bit more complicated than this, but this is a general principle. The sync server is the centrally located computer that everyone works on and gets uploaded to each person's screen.

The linked device example is similar in that there is one central server processing all the accounts and uploading any traffic to each device on that account. But let's take a look at this further.

First, we need to look at public key infrastructure and asymmetric cryptography quickly.

### Asymmetric cryptography

With symmetric cryptography, you lock and unlock the message with the same key. Whereas with asymmetric cryptography, you have two keys, a public and a private. You can lock the message with whichever one you would like, but you must unlock it with the opposite key. We will cover this much more in the encryption module, but now this is most important.

Let's say Alice and Bob want to talk to each other. There are a few instances that can happen.

1) Bob encrypts his message with Alice's public key. Since Alice is the only person with the private key (remember when you lock with one, you must unlock with the other), then Bob knows Alice is the only person who can unlock and read that message. He also knows that the reader will be Alice and not someone pretending to be Alice.

2) Bob encrypts with his private key. This would allow Alice to know that the message definitely came from Bob. Everyone knows Bob's public key, and since Bob is the only one that could have locked that message, then it must be Bob sending it.

3) Bob can encrypt with his public key. This means that the only person that can unlock this message is Bob. There are a couple of uses for this, by the way, but in this scenario, we will not need it.

### Public key infrastructure

When the public and private keys are generated, the public key must get out to the world to see. It does this by going on relevant servers usually, although you may also see a PGP key at the bottom of some emails. Nevertheless, the distribution for these public keys is stored on the same synchronisation servers for this particular use.

So, the public key infrastructure is the service where all the public keys are stored. If you wanted to message a friend, you would have to go to some PKI to get their key to encrypt the message and send it to them. The important take from all of this is that depending on how the server controls this. It is possible for the server itself to snoop on the messages or processes happening. The fiaw is in knowing exactly whose private, and public key are being used. If the server controls this, then there is a possibility they could forward the message to themselves.