Tools of the Trade: Linux and SQL

Module 1 - Introduction to Operating Systems

Introduction to operating systems

- Operating system is the interface between the computer hardware and the user.
- The operating system, or the OS as it's commonly called, is responsible for making the computer run as efficiently as possible while also making it easy to use.
- Hardware may be another new term. Hardware refers to the physical components of a computer.
- The OS interface that we now rely on every day is something that early computers didn't have. In the 1950s the biggest challenge with early computers was the amount of time it took to run a computer program.
- At the time, computers could not run multiple programs simultaneously.
- Instead, people had to wait for a program to finish running, reset the computer, and load up the new program.
- Another reason why operating systems are important is that they help humans and computers communicate with each other.
- Computers communicate in a language called binary, which consists of 0s and 1s.
- The OS provides an interface to bridge this communication gap between the user and the computer, allowing you to interact with the computer in complex ways.
- Operating systems are critical for the use of computers.
- Likewise, OS security is also critical for the security of a computer.
- This involves securing files, data access, and user authentication to help protect and prevent threats such as viruses, worms, and malware. Knowing how operating systems work is essential for completing different security related tasks.
- For example, as a security analyst, you may be responsible for configuring and maintaining the security of a system by managing access.
- You may also be responsible for managing and configuring firewalls, setting security policies, enabling virus protection, and performing auditing, accounting, and logging to detect unusual behavior.

Common operating systems

The following operating systems are useful to know in the security industry: Windows, macOS®, Linux, ChromeOS, Android, and iOS.

Windows and macOS

Windows and macOS are both common operating systems. The Windows operating system was introduced in 1985, and macOS was introduced in 1984. Both operating systems are used in personal and enterprise computers.

Windows is a closed-source operating system, which means the source code is not shared freely with the public. macOS is partially open source. It has some open-source components, such as macOS's kernel. macOS also has some closed-source components.

Linux

The first version of Linux was released in 1991, and other major releases followed in the early 1990s. Linux is a completely open-source operating system, which means that anyone can access Linux and its source code. The open-source nature of Linux allows developers in the Linux community to collaborate.

Linux is particularly important to the security industry. There are some distributions that are specifically designed for security. Later in this course, you'll learn about Linux and its importance to the security industry.

ChromeOS

ChromeOS launched in 2011. It's partially open source and is derived from Chromium OS, which is completely open source. ChromeOS is frequently used in the education field.

Android and iOS

Android and iOS are both mobile operating systems. Unlike the other operating systems mentioned, mobile operating systems are typically used in mobile devices, such as phones, tablets, and watches. Android was introduced for public use in 2008, and iOS was introduced in 2007. Android is open source, and iOS is partially open source.

Operating systems and vulnerabilities

Security issues are inevitable with all operating systems. An important part of protecting an operating system is keeping the system and all of its components up to date.

Legacy operating systems

A **legacy operating system** is an operating system that is outdated but still being used. Some organizations continue to use legacy operating systems because software they rely on is not compatible with newer operating systems. This can be more common in industries that use a lot of equipment that requires embedded software—software that's placed inside components of the equipment.

Legacy operating systems can be vulnerable to security issues because they're no longer supported or updated. This means that legacy operating systems might be vulnerable to new threats.

Other vulnerabilities

Even when operating systems are kept up to date, they can still become vulnerable to attack. Below are several resources that include information on operating systems and their vulnerabilities.

- Microsoft Security Response Center (MSRC): A list of known vulnerabilities affecting Microsoft products and services
- Apple Security Updates: A list of security updates and information for Apple® operating systems, including macOS and iOS, and other products

- Common Vulnerabilities and Exposures (CVE) Report for Ubuntu: A list of known vulnerabilities affecting Ubuntu, which is a specific distribution of Linux
- Google Cloud Security Bulletin: A list of known vulnerabilities affecting Google Cloud products and services

Keeping an operating system up to date is one key way to help the system stay secure. Because it can be difficult to keep all systems updated at all times, it's important for security analysts to be knowledgeable about legacy operating systems and the risks they can create.

Inside the operating system

- Think about when someone drives a car. They push the gas pedal and the car moves forward. They don't need to pay attention to all the mechanics that allow the car to move. Just like a car can't work without its engine, a computer can't work without its operating system.
- The job of an OS is to help other computer programs run efficiently. The OS does this by taking care of all the messy details related to controlling the computer's hardware, so you don't have to.
- First, let's see what happens when you turn on the computer. When you press the power button, you're interacting with the hardware. This boots the computer and brings up the operating system. Booting the computer means that a special microchip called a BIOS is activated. On many computers built after 2007, the chip was replaced by the UEFI. Both BIOS and UEFI contain booting instructions that are responsible for loading a special program called the bootloader. Then, the bootloader is responsible for starting the operating system. Just like that, your computer is on.
- As a security analyst, understanding these processes can be helpful for you. Vulnerabilities can occur in something like a booting process. Often, the BIOS is not scanned by the antivirus software, so it can be vulnerable to malware infection. Now, that you learned how to boot the operating system, let's look at how you and all users communicate with the system to complete a task.
- The process starts with you, the user. And to complete tasks, you use applications on your computer. An application is a program that performs a specific task. When you do this, the application sends your request to the operating system. From there, the operating system interprets this request and directs it to the appropriate component of the computer's hardware.
- In the previous video, we learned that the hardware consists of all the physical components of the computer. The hardware will also send information back to the operating system. And this in turn is sent back to the application.
- Let's give a simple overview of how this works when you want to use the calculator on your computer. You use your mouse to click on the calculator application on your computer. When you type in the number you want to calculate, the application communicates with the operating system. Your operating system then sends a calculation to a component of the hardware, the central processing unit, or CPU. Once the hardware does the work of determining the final number, it sends the answer back to your operating system. Then, it can be displayed in your calculator application.

- Understanding this process is helpful when investigating security events. Security analysts should be able to trace back through this process flow to analyze where a security event could have occurred.

Operating systems are a critical component of a computer. They make connections between applications and hardware to allow users to perform tasks. In this reading, you'll explore this complex process further and consider it using a new analogy and a new example.

Booting the computer

When you boot, or turn on, your computer, either a BIOS or UEFI microchip is activated. The **Basic Input/Output System (BIOS)** is a microchip that contains loading instructions for the computer and is prevalent in older systems. The **Unified Extensible Firmware Interface (UEFI)** is a microchip that contains loading instructions for the computer and replaces BIOS on more modern systems. The BIOS and UEFI chips both perform the same function for booting the computer. BIOS was the standard chip until 2007, when UEFI chips increased in use. Now, most new computers include a UEFI chip. UEFI provides enhanced security features.

The BIOS or UEFI microchips contain a variety of loading instructions for the computer to follow. For example, one of the loading instructions is to verify the health of the computer's hardware. The last instruction from the BIOS or UEFI activates the bootloader. The **bootloader** is a software program that boots the operating system. Once the operating system has finished booting, your computer is ready for use.

Completing a task

As previously discussed, operating systems help us use computers more efficiently. Once a computer has gone through the booting process, completing a task on a computer is a four-part process.

User

The first part of the process is the user. The user initiates the process by having something they want to accomplish on the computer. Right now, you're a user! You've initiated the process of accessing this reading.

Application

The application is the software program that users interact with to complete a task. For example, if you want to calculate something, you would use the calculator application. If you want to write a report, you would use a word processing application. This is the second part of the process.

Operating system

The operating system receives the user's request from the application. It's the operating system's job to interpret the request and direct its flow. In order to complete the task, the operating system sends it on to applicable components of the hardware.

Hardware

The hardware is where all the processing is done to complete the tasks initiated by the user. For example, when a user wants to calculate a number, the CPU figures out the answer. As another example, when a user wants to save a file, another component of the hardware, the hard drive, handles this task.
After the work is done by the hardware, it sends the output back through the operating system to the application so that it can display the results to the user.

The OS at work behind the scenes

Consider once again how a computer is similar to a car. There are processes that someone won't directly observe when operating a car, but they do feel it move forward when they press the gas pedal. It's the same with a computer. Important work happens inside a computer that you don't experience directly. This work involves the operating system.
You can explore this through another analogy. The process of using an operating system is also similar to ordering at a restaurant. At a restaurant you place an order and get your food, but you don't see what's happening in the kitchen when the cooks prepare the food.
Ordering food is similar to using an application on a computer. When you order your food, you make a specific request like "a small soup, very hot." When you use an application, you also make specific requests like "print three double-sided copies of this document."
You can compare the food you receive to what happens when the hardware sends output. You receive the food that you ordered. You receive the document that you wanted to print.
Finally, the kitchen is like the OS. You don't know what happens in the kitchen, but it's critical in interpreting the request and ensuring you receive what you ordered. Similarly, though the work of the OS is not directly transparent to you, it's critical in completing your tasks.

An example: Downloading a file from an internet browser

Previously, you explored how operating systems, applications, and hardware work together by examining a task involving a calculation. You can expand this understanding by exploring how the OS completes another task, downloading a file from an internet browser:

- First, the user decides they want to download a file that they found online, so they click on a download button near the file in the internet browser application.
- Then, the internet browser communicates this action to the OS.
- The OS sends the request to download the file to the appropriate hardware for processing.
- The hardware begins downloading the file, and the OS sends this information to the internet browser application. The internet browser then informs the user when the file has been downloaded.

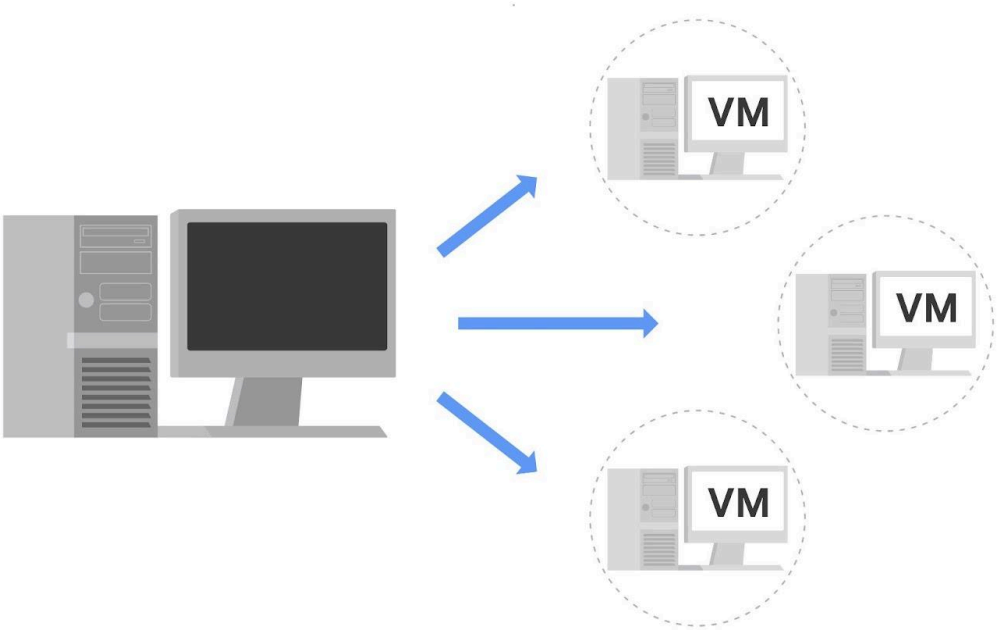
Resource allocation via the OS

- Not only does the OS interact with other parts of your computer, but it's also responsible for managing the resources of the system. This is a big task that requires a lot of balance to make sure all the resources of the computer are used efficiently. Think of this like the concept of energy. A person needs energy to complete different tasks.

- Some tasks need more energy, while others require less. For example, going for a run requires more energy than watching TV. A computer's OS also needs to make sure that it has enough energy to function correctly for certain tasks. Running an antivirus scan on your computer will use more energy than using the calculator application.
- Imagine your computer is an orchestra. Many different instruments like violins, drums, and trumpets are all part of the orchestra. An orchestra also has a conductor to direct the flow of the music. In a computer, the OS is the conductor. The OS handles resource and memory management to ensure the limited capacity of the computer system is used where it's needed most. A variety of programs, tasks, and processes are constantly competing for the resources of the central processing unit, or CPU. They all have their own reasons why they need memory, storage, and input/output bandwidth. The OS is responsible for ensuring that each program is allocating and de-allocating resources. All this occurs in your computer at the same time so that your system functions efficiently.
- Much of this is hidden from you as a user.
- But, your task manager will list all of the tasks that are being processed, along with their memory and CPU usage.
- As an analyst, it's helpful to know where a system's resources are used. Understanding usage of resources can help you respond to an incident and troubleshoot applications in the system.
- For example, if a computer is running slowly, an analyst might discover it's allocating resources to malware.

What is a virtual machine?

A **virtual machine (VM)** is a virtual version of a physical computer. Virtual machines are one example of virtualization. Virtualization is the process of using software to create virtual representations of various physical machines. The term “virtual” refers to machines that don’t exist physically, but operate like they do because their software simulates physical hardware. Virtual systems don’t use dedicated physical hardware. Instead, they use software-defined versions of the physical hardware. This means that a single virtual machine has a virtual CPU, virtual storage, and other virtual hardware. Virtual systems are just code.



You can run multiple virtual machines using the physical hardware of a single computer. This involves dividing the resources of the host computer to be shared across all physical and virtual components. For example, **Random Access Memory (RAM)** is a hardware component used for short-term memory. If a computer has 16GB of RAM, it can host three virtual machines so that the physical computer and virtual machines each have 4GB of RAM. Also, each of these virtual machines would have their own operating system and function similarly to a typical computer.

Benefits of virtual machines

Security professionals commonly use virtualization and virtual machines. Virtualization can increase security for many tasks and can also increase efficiency.

Security

One benefit is that virtualization can provide an isolated environment, or a sandbox, on the physical host machine. When a computer has multiple virtual machines, these virtual machines are “guests” of the computer. Specifically, they are isolated from the host computer and other guest virtual machines. This provides a layer of security, because virtual machines can be kept separate from the other systems. For example, if an individual virtual machine becomes infected with malware, it can be dealt with more securely because it’s isolated from the other machines. A security professional could also intentionally place malware on a virtual machine to examine it in a more secure environment.

Note: Although using virtual machines is useful when investigating potentially infected machines or running malware in a constrained environment, there are still some risks. For example, a malicious program can escape virtualization and access the host machine. This is why you should never completely trust virtualized systems.

Efficiency

Using virtual machines can also be an efficient and convenient way to perform security tasks. You can open multiple virtual machines at once and switch easily between them. This allows you to streamline security tasks, such as testing and exploring various applications.

You can compare the efficiency of a virtual machine to a city bus. A single city bus has a lot of room and is an efficient way to transport many people simultaneously. If city buses didn't exist, then everyone on the bus would have to drive their own cars. This uses more gas, cars, and other resources than riding the city bus.
Similar to how many people can ride one bus, many virtual machines can be hosted on the same physical machine. That way, separate physical machines aren't needed to perform certain tasks.

Managing virtual machines

Virtual machines can be managed with a software called a hypervisor. Hypervisors help users manage multiple virtual machines and connect the virtual and physical hardware. Hypervisors also help with allocating the shared resources of the physical host machine to one or more virtual machines.
One hypervisor that is useful for you to be familiar with is the Kernel-based Virtual Machine (KVM). KVM is an open-source hypervisor that is supported by most major Linux distributions. It is built into the Linux kernel, which means it can be used to create virtual machines on any machine running a Linux operating system without the need for additional software.

Other forms of virtualization

In addition to virtual machines, there are other forms of virtualization. Some of these virtualization technologies do not use operating systems. For example, multiple virtual servers can be created from a single physical server. Virtual networks can also be created to more efficiently use the hardware of a physical network.

GUI versus CLI

Let's discuss how users and operating systems communicate with each other. So far, you've learned that a computer has an operating system, hardware, and applications. Remember, the operating system communicates with the hardware to execute tasks.

The user communicates with the operating system via an interface. A user interface is a program that allows a user to control the functions of the operating system. Two user interfaces that we'll discuss are the graphical user interface, or GUI, and the command-line interface, or CLI. Let's cover these interfaces in more detail.

A GUI is a user interface that uses icons on the screen to manage different tasks on the computer. Most operating systems can be used with a graphical user interface. If you've used a personal computer or a cell phone, you have experienced operating a GUI. Most GUIs include these components: a start menu with program groups, a task bar for launching programs, and a desktop with icons and shortcuts. All these components help you communicate with the OS to execute tasks. In addition to clicking on icons, when you use a GUI, you can also search for files or applications from the start menu. You just have to remember the icon or name of the program to activate an application.

Now let's discuss the command-line interface. In comparison, the command-line interface, or CLI, is a text-based user interface that uses commands to interact with the computer. These commands communicate with the operating system and execute tasks like opening programs. The command-line interface is a much different structure than the graphical user interface. When you use the CLI, you'll immediately notice a difference. There are no icons or graphics on the screen.

The command-line interface looks similar to lines of code using certain text languages. A CLI is more flexible and more powerful than a GUI. Think about using a CLI like creating whatever meal you'd like from ingredients bought at a grocery store. This gives you a lot of control and customization about what you're going to eat.

In comparison, using a GUI is more like ordering food from a restaurant. You can only order what's on the menu. If you want both a noodle dish and pizza, but the first restaurant you go to only has pizza, you'll have to go to another restaurant to order the noodles. With a graphical user interface, you must do one task at a time. But the command-line interface allows for customization, which lets you complete multiple tasks simultaneously. For example, imagine you have a folder with hundreds of files of different file types, and you need to move only the JPEG files to a new folder. Think about how slow and tedious this would be as you use a GUI to find each JPEG file in this folder and move it into the new one. On the other hand, the CLI would allow you to streamline this process and move them all at once.

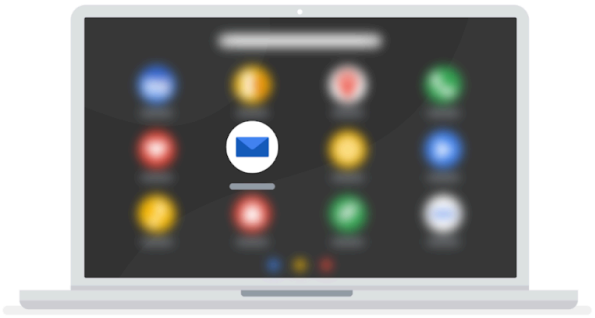
As you can see, there are very big differences in these two types of user interfaces. As a security analyst, some of your work may involve the command-line interface. When analyzing logs or authenticating and authorizing users, security analysts commonly use a CLI in their everyday work. In this video, we discussed two types of user interfaces. You learned that you already have experience using a graphical user interface, as most personal computers and cell phones use a GUI. You were introduced to the command-line interface.

CLI vs. GUI

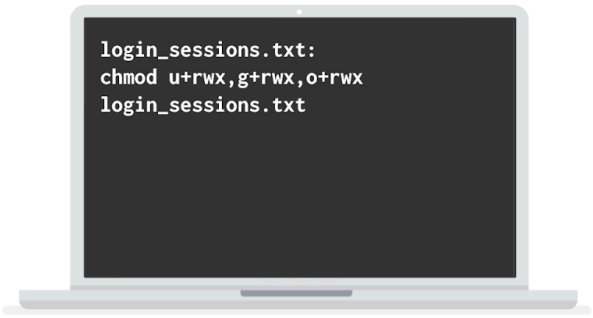
A **graphical user interface (GUI)** is a user interface that uses icons on the screen to manage different tasks on the computer. A **command-line interface (CLI)** is a text-based user interface that uses commands to interact with the computer.

Display

One notable difference between these two interfaces is how they appear on the screen. A GUI has graphics and icons, such as the icons on your desktop or taskbar for launching programs. In contrast, a CLI only has text. It looks similar to lines of code.



Graphical user interface
(GUI)



Command-line interface
(CLI)

Function

These two interfaces also differ in how they function. A GUI is an interface that only allows you to make one request at a time. However, a CLI allows you to make multiple requests at a time.

Advantages of a CLI in cybersecurity

The choice between using a GUI or CLI is partly based on personal preference, but security analysts should be able to use both interfaces. Using a CLI can provide certain advantages.

Efficiency

Some prefer the CLI because it can be used more quickly when you know how to manage this interface. For a new user, a GUI might be more efficient because they're easier for beginners to navigate.

Because a CLI can accept multiple requests at one time, it's more powerful when you need to perform multiple tasks efficiently. For example, if you had to create multiple new files in your system, you could quickly perform this task in a CLI. If you were using a GUI, this could take much longer, because you have to repeat the same steps for each new file.

History file

For security analysts, using the Linux CLI is helpful because it records a history file of all the commands and actions in the CLI. If you were using a GUI, your actions are not necessarily saved in a history file.

For example, you might be in a situation where you're responding to an incident using a playbook. The playbook's instructions require you to run a series of different commands. If you used a CLI, you'd be able to go back to the history and ensure all of the commands were correctly used. This could be helpful if there were issues using the playbook and you had to review the steps you performed in the command line.

Additionally, if you suspect an attacker has compromised your system, you might be able to trace their actions using the history file.

Terms and definitions from Course 4, Module 1

- **Application:** A program that performs a specific task
- **Basic Input/Output System (BIOS):** A microchip that contains loading instructions for the computer and is prevalent in older systems
- **Bootloader:** A software program that boots the operating system
- **Command-line interface (CLI):** A text-based user interface that uses commands to interact with the computer
- **Graphical user interface (GUI):** A user interface that uses icons on the screen to manage different tasks on the computer
- **Hardware:** The physical components of a computer
- **Legacy operating system:** An operating system that is outdated but still being used
- **Operating system (OS)**: The interface between computer hardware and the user
- **Random Access Memory (RAM):** A hardware component used for short-term memory
- **Unified Extensible Firmware Interface (UEFI):** A microchip that contains loading instructions for the computer and replaces BIOS on more modern systems
- **User interface:** A program that allows the user to control the functions of the operating system
- **Virtual machine (VM)**: A virtual version of a physical computer

Module 2 - The Linux Operating System

Introduction to Linux

Linux is an open-source operating system. It was created in two parts. In the early 1990s, two different people were working separately on projects to improve computer engineering. The first person was Linus Torvalds. At the time, the UNIX operating system was already in use. He wanted to improve it and make it open source and accessible to anyone. What was revolutionary was his introduction of the Linux kernel.

Around the same time, Richard Stallman started working on GNU. GNU was also an operating system based on UNIX. Stallman shared Torvalds' goal of creating software that was free and open to anyone. After working on GNU for a few years, the missing element for the software was a kernel. Together, Torvalds' and Stallman's innovations made what is commonly referred to as Linux.

Linux is open source, meaning anyone can have access to the operating system and the source code. Linux and many of the programs that come with Linux are licensed under the terms of the GNU Public License, which allow you to use, share, and modify them freely. Thanks to Linux's open-source philosophy as well as a strong feature set, an entire community of developers has adopted this operating system. These developers are able to collaborate on projects and advance computing together. As a security analyst, you'll discover that Linux is used at different organizations. More specifically, Linux is used in many security programs. Another unique feature about Linux is the different distributions, or varieties, that have been developed. Because of the large community contribution, there are over 600 distributions of Linux. Later you'll learn more about distributions.

Finally, let's take a look at how you would use Linux in an entry-level security position. As a security analyst, you'll use many tools and programs in everyday work. You might examine different types of logs to identify what's going on in the system. For example, you might find yourself looking at an error log when investigating an issue. Another place where you will use Linux is to verify access and authorization in an identity and access management system. In security, managing access is key in order to ensure a secure system. We'll take a closer look into access and authorization later.

Finally, as an analyst, you might find yourself working with specific distributions designed for a particular task. For example, you might use a distribution that has a digital forensic tool to investigate what happened in an event alert. You might also use a distribution that's for pen testing in offensive security to look for vulnerabilities in the system. Distributions are created to fit the needs of their users.

Linux architecture

Just like buildings, operating systems also have an architecture and are made up of discrete components that work together to form the whole.

The components of Linux include the user, applications, the shell, the Filesystem Hierarchy Standard, the kernel, and the hardware.

First, you are the user. The user is the person interacting with the computer. In Linux, you're the first element to the architecture of the operating system. You're initiating the tasks or commands that the OS is going to execute. Linux is a multi-user system. This means that more than one user can use the system's resources at the same time.

The second element of the architecture is the applications within a system. An application is a program that performs a specific task, such as a word processor or a calculator. You might hear the word "applications" and "programs" used interchangeably. As an example, one popular Linux application that we'll learn more about later is Nano. Nano is a text editor. This simple application

helps you keep notes on the screen. Linux applications are commonly distributed through package managers. We'll learn more about this process later.

The next component in the architecture of Linux is the shell. This is an important element because it is how you will communicate with the system. The shell is a command line interpreter. It processes commands and outputs the results. This might sound familiar. Previously, we learned about the two types of user interfaces: the GUI and the CLI. You can think of the shell as a CLI.

Another element of the architecture of Linux is the Filesystem Hierarchy Standard, or FHS. It's the component of the Linux OS that organizes data. An easy way for you to think about the FHS is to think about it as a filing cabinet of data. The FHS is how data is stored in a system. It's a way to organize data so that it can be found when the data is accessed by the system.

That brings us to the kernel. The kernel is a component of the Linux OS that manages processes and memory. The kernel communicates with the hardware to execute the commands sent by the shell. The kernel uses drivers to enable applications to execute tasks. The Linux kernel helps ensure that the system allocates resources more efficiently and makes the system work faster.

Finally, the last component of the architecture is the hardware. Hardware refers to the physical components of a computer. You can compare this to software applications which can be downloaded into a system. The hardware in your computer are things like the CPU, mouse, and keyboard.

User

The **user** is the person interacting with a computer. They initiate and manage computer tasks. Linux is a multi-user system, which means that multiple users can use the same resources at the same time.

Applications

An **application** is a program that performs a specific task. There are many different applications on your computer. Some applications typically come pre-installed on your computer, such as calculators or calendars. Other applications might have to be installed, such as some web browsers or email clients. In Linux, you'll often use a package manager to install applications. A **package manager** is a tool that helps users install, manage, and remove packages or applications. A **package** is a piece of software that can be combined with other packages to form an application.

Shell

The **shell** is the command-line interpreter. Everything entered into the shell is text based. The shell allows users to give commands to the kernel and receive responses from it. You can think of the shell as a translator between you and your computer. The shell translates the commands you enter so that the computer can perform the tasks you want.

Filesystem Hierarchy Standard (FHS)

The **Filesystem Hierarchy Standard (FHS)** is the component of the Linux OS that organizes data. It specifies the location where data is stored in the operating system.

A **directory** is a file that organizes where other files are stored. Directories are sometimes called “folders,” and they can contain files or other directories. The FHS defines how directories, directory contents, and other storage is organized so the operating system knows where to find specific data.

Kernel

The **kernel** is the component of the Linux OS that manages processes and memory. It communicates with the applications to route commands. The Linux kernel is unique to the Linux OS and is critical for allocating resources in the system. The kernel controls all major functions of the hardware, which can help get tasks expedited more efficiently.

Hardware

The **hardware** is the physical components of a computer. You might be familiar with some hardware components, such as hard drives or CPUs. Hardware is categorized as either peripheral or internal.

Peripheral devices

Peripheral devices are hardware components that are attached and controlled by the computer system. They are not core components needed to run the computer system. Peripheral devices can be added or removed freely. Examples of peripheral devices include monitors, printers, the keyboard, and the mouse.

Internal hardware

Internal hardware are the components required to run the computer. Internal hardware includes a main circuit board and all components attached to it. This main circuit board is also called the motherboard. Internal hardware includes the following:

- The **Central Processing Unit (CPU)** is a computer’s main processor, which is used to perform general computing tasks on a computer. The CPU executes the instructions provided by programs, which enables these programs to run.
- **Random Access Memory (RAM)** is a hardware component used for short-term memory. It’s where data is stored temporarily as you perform tasks on your computer. For example, if you’re writing a report on your computer, the data needed for this is stored in RAM. After you’ve finished writing the report and closed down that program, this data is deleted from RAM. Information in RAM cannot be accessed once the computer has been turned off. The CPU takes the data from RAM to run programs.
- The **hard drive** is a hardware component used for long-term memory. It’s where programs and files are stored for the computer to access later. Information on the hard drive can be accessed even after a computer has been turned off and on again. A computer can have multiple hard drives.

Linux Distributions

Linux is a very customizable operating system. Unlike other operating systems, there are different versions available for you to use. These different versions of Linux are called distributions. You might also hear them called distros or flavors of Linux. It's essential for you to understand the

distribution that you're using so you know what tools and apps are available to you. For example, Debian is a distro that has different tools than the Ubuntu distribution.

Let's use an analogy to describe Linux distributions. Think of the OS as a vehicle. First, we'll start with its engine—that would be the kernel. Just as the engine makes a vehicle run, the kernel is the most important component of the Linux OS. Because the Linux kernel is open source, anyone can take the kernel and modify it to build a new distribution. This is comparable to a vehicle manufacturer taking an engine and creating different types of vehicles: trucks, cars, vans, convertibles, busses, airplanes, and so on. These different types of vehicles can be compared to different Linux distributions. A bus is used to transport lots of people. A truck is used to transport a large number of goods across vast distances. An aircraft transports passengers or goods by air.

Just as each vehicle serves its own purpose, different distributions are used for different reasons. Additionally, vehicles all have different components which distinguish them from each other. Aircrafts have control panels with buttons and knobs. Regular cars have four tires, but trucks can have more. Similarly, different Linux distributions contain different preinstalled programs, user interfaces, and much more. A lot of this is based on what the Linux user needs, but some distros are also chosen based on preference—the same way a sports car might be chosen as a vehicle.

The advantage of using Linux as an OS is that you can customize it. Distributions include the Linux kernel, utilities, a package management system, and an installer. We learned earlier that Linux is open source, and anyone can contribute to adding to the source code. That is how new distributions are created.

All distros are derived from another distro, but there are a few that are considered parent distributions. Red Hat® is the parent of CentOS, and Slackware® is the parent of SUSE®. Both Ubuntu and KALI LINUX™ are derived from Debian.

KALI LINUX™

KALI LINUX™ is a trademark of Offensive Security and is Debian derived. This open-source distro was made specifically with penetration testing and digital forensics in mind. There are many tools pre-installed into KALI LINUX™. It's important to note that KALI LINUX™ should be used on a virtual machine. This prevents damage to your system in the event its tools are used improperly. An additional benefit is that using a virtual machine gives you the ability to revert to a previous state.

As security professionals advance in their careers, some specialize in penetration testing. A penetration test is a simulated attack that helps identify vulnerabilities in systems, networks, websites, applications, and processes. KALI LINUX™ has numerous tools that are useful during penetration testing. Let's look at a few examples.

To begin, Metasploit can be used to look for and exploit vulnerabilities on machines. Burp Suite is another tool that helps to test for weaknesses in web applications. And finally, John the Ripper is a tool used to guess passwords. As a security analyst, your work might involve digital forensics. Digital forensics is the process of collecting and analyzing data to determine what has happened after an attack. For example, you might take an investigative look at data related to network activity. KALI LINUX™ is also a useful distribution for security professionals who are involved in digital forensic work. It has a large number of tools that can be used for this. As one example, tcpdump is a command-line packet analyzer. It's used to capture network traffic. Another tool commonly used in the security profession is Wireshark. It has a graphical user interface that can be used to analyze live and captured network traffic. And as a final example, Autopsy is a forensic tool used to analyze hard drives and smartphones. These are just a few tools included with KALI LINUX™. This distribution has many tools used to conduct pen testing and digital forensics.

More Linux distributions

KALI LINUX ™

KALI LINUX ™ is an open-source distribution of Linux that is widely used in the security industry. This is because KALI LINUX ™, which is Debian-based, is pre-installed with many useful tools for penetration testing and digital forensics. A penetration test is a simulated attack that helps identify vulnerabilities in systems, networks, websites, applications, and processes. Digital forensics is the practice of collecting and analyzing data to determine what has happened after an attack. These are key activities in the security industry.
However, KALI LINUX ™ is not the only Linux distribution that is used in cybersecurity.

Ubuntu

Ubuntu is an open-source, user-friendly distribution that is widely used in security and other industries. It has both a command-line interface (CLI) and a graphical user interface (GUI). Ubuntu is also Debian-derived and includes common applications by default. Users can also download many more applications from a package manager, including security-focused tools. Because of its wide use, Ubuntu has an especially large number of community resources to support users.
Ubuntu is also widely used for cloud computing. As organizations migrate to cloud servers, cybersecurity work may more regularly involve Ubuntu derivatives.

Parrot

Parrot is an open-source distribution that is commonly used for security. Similar to KALI LINUX ™, Parrot comes with pre-installed tools related to penetration testing and digital forensics. Like both KALI LINUX ™ and Ubuntu, it is based on Debian.
Parrot is also considered to be a user-friendly Linux distribution. This is because it has a GUI that many find easy to navigate. This is in addition to Parrot's CLI.

Red Hat® Enterprise Linux®

Red Hat Enterprise Linux is a subscription-based distribution of Linux built for enterprise use. Red Hat is not free, which is a major difference from the previously mentioned distributions. Because it's built and supported for enterprise use, Red Hat also offers a dedicated support team for customers to call about issues.

CentOS

CentOS is an open-source distribution that is closely related to Red Hat. It uses source code published by Red Hat to provide a similar platform. However, CentOS does not offer the same enterprise support that Red Hat provides and is supported through the community.

Introduction to package managers

A package is a piece of software that can be combined with other packages to form an application. Some packages may be large enough to form applications on their own. Packages contain the files necessary for an application to be installed. These files include dependencies, which are supplemental files used to run an application.
Package managers can help resolve any issues with dependencies and perform other management tasks. A package manager is a tool that helps users install, manage, and remove packages or applications. Linux uses multiple package managers.
Note: It's important to use the most recent version of a package when possible. The most recent version has the most up-to-date bug fixes and security patches. These help keep your system more secure.

Types of package managers

Many commonly used Linux distributions are derived from the same parent distribution. For example, KALI LINUX ™, Ubuntu, and Parrot all come from Debian. CentOS comes from Red Hat. This knowledge is useful when installing applications because certain package managers work with certain distributions. For example, the Red Hat Package Manager (RPM) can be used for Linux distributions derived from Red Hat, and package managers such as dpkg can be used for Linux distributions derived from Debian.
Different package managers typically use different file extensions. For example, Red Hat Package Manager (RPM) has files which use the .rpm file extension, such as Package-Version-Release_Architecture.rpm. Package managers for Debian-derived Linux distributions, such as dpkg, have files which use the .deb file extension, such as Package_Version-Release_Architecture.deb.

Package management tools

In addition to package managers like RPM and dpkg, there are also package management tools that allow you to easily work with packages through the shell. Package management tools are sometimes utilized instead of package managers because they allow users to more easily perform basic tasks, such as installing a new package. Two notable tools are the Advanced Package Tool (APT) and Yellowdog Updater Modified (YUM).

Advanced Package Tool (APT)

APT is a tool used with Debian-derived distributions. It is run from the command-line interface to manage, search, and install packages.

Yellowdog Updater Modified (YUM)

YUM is a tool used with Red Hat-derived distributions. It is run from the command-line interface to manage, search, and install packages. YUM works with .rpm files.

How to use Qwiklabs

Launching Qwiklabs

When you select a lab, you start from a Coursera page. You will need to click **Launch App** on that page. After you click **Launch App**, a new tab will open with a Qwiklabs page that contains instructions for that particular lab.

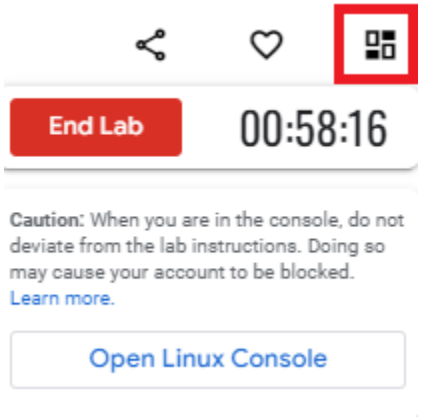
Start Lab button

On the Qwiklabs page, you must click **Start Lab** to open a temporary terminal. The instructions for the lab will move to the right side of the screen.

Read the instructions and complete all the tasks in the lab by entering commands in the terminal.
Note: It may take a moment for the terminal to start.

Lab control dialog box

After you click **Start Lab**, the lab control dialog box opens. It contains the **End Lab** button, the **timer**, and the **Open Linux Console** button.
You can hide or unhide the dialog box by clicking the following icon in the red box:

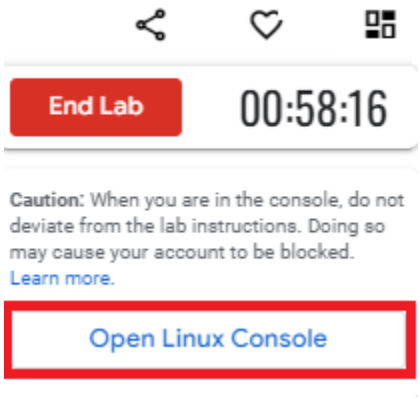


The timer

The **timer** starts when the terminal has loaded. The timer keeps track of the amount of time you have left to complete a lab. The timer counts down until it reaches 00:00:00. When it does, your temporary terminal and resources are deleted.
You will have ample time to complete the labs. But, stay focused on completing the tasks to ensure you use your time well.

Open Linux Console button

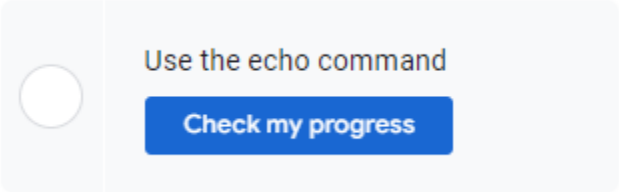
When you click the button to **Open Linux Console**, the terminal opens in a new browser window:



Use this feature if you want a full-screen view of the terminal. You can close this window at any time. Closing the window does not end your lab, and you can continue working in the terminal in the original tab.

Check progress

You can check your progress by clicking **Check my progress** at the end of each task.



If you haven't yet completed a task, you'll receive hints on what you must do to complete it. You can click **Check my progress** whenever you want to check the completion status of a task or receive a hint.

Using copy/paste commands

The first time you try to use copy or paste keyboard shortcuts (such as **CTRL + C**), you'll receive a pop-up requesting permission to use your device's clipboard: "**googlecoursera.qwiklabs.com wants to see text and images copied to the clipboard.**" Please click **Allow** if you would like to be able to use these shortcuts in the Qwiklabs platform. If you choose not to allow Qwiklabs access to your clipboard, you cannot use keyboard shortcuts but you can still complete the lab.

Code block

Certain steps may include a code block. Click the copy button to copy the code provided and then paste it into the terminal.

To paste code or other text content that you have copied from the instructions into the terminal, activate the terminal by clicking anywhere inside it. The terminal is active when the cursor in the terminal changes from a static empty outline to a flashing solid block.

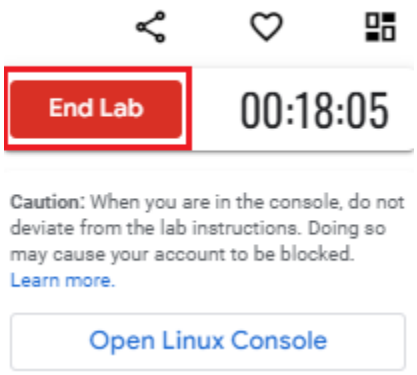
```
analyst@14bc4618e5ba:~$ ls reports
Q1patches.txt  Q2patches.txt
analyst@14bc4618e5ba:~$ pwd
/home/analyst
analyst@14bc4618e5ba:~$
```

Once the terminal is active, use the keyboard shortcut **CTRL + V** (hold down the **CTRL** key and press the **V** key) to insert the copied text into the terminal at the location of the flashing cursor.

Scrolling

In certain situations, you may want to scroll within the terminal window. To do so, use the scroll wheel on your mouse or the touchpad of your computer.

End Lab button



Finally, click **End Lab** when you've completed the tasks in the lab.
Note: Don't click **End Lab** until you're finished; you'll lose access to the work you've done throughout the lab.

Tracking progress on Coursera

If you complete a lab but your progress hasn't been tracked on Coursera, you may need to refresh the page for your progress to be registered. Once you complete the lab and refresh the page, the green check mark should appear.

Helpful navigation tips and keyboard shortcuts

The following contains a list of navigation tips and keyboard shortcuts you may find useful when completing your Linux labs. Your cursor must be in the terminal window to use these navigation tips and keyboard shortcuts.

- **CTRL + C**: Terminates a command that is currently running; from the instructions portion of Qwiklabs, you can use **CTRL + C** to copy, but within the terminal, it will only terminate a command and if one isn't running, it will display **^C** at the prompt
- **CTRL + V**: Pastes text
- **clear**: Clears the terminal screen; this can also be done by entering **CTRL + L**
- **CTRL + A**: Sets your cursor at the beginning of a command
- **CTRL + E**: Sets your cursor at the end of a command
- **Left arrow key**: Moves left within a command
- **Right arrow key**: Moves right within a command

- **Up arrow key**: Provides the last command you entered into the command line; can be entered multiple times to go through multiple commands from the command history
- **Down arrow key**: Provides the next command in the command history; must be after using the **up arrow** key
- **Tab key**: Provides available suggestions for completing your text

Browser compatibility

Make sure your internet browser is updated regularly. Qwiklabs and Jupyter Notebooks require the latest version of Google Chrome, Firefox, or Microsoft Edge. If your browser is outdated or you are using a browser that is not supported by Qwiklabs or Jupyter Notebooks, you may encounter a problem. If your browser is up to date and you are using one of the browsers listed above and still encountering problems try restarting your browser or clearing your browser's cache and cookies. You can also use incognito mode which prevents your browser from storing cookies and other temporary data.
Note: The Qwiklabs user interface works best with Google Chrome.

Internet connection

Qwiklabs and Jupyter Notebooks require a stable internet connection. If you are experiencing problems starting or completing Qwiklabs or Jupyter Notebooks, your internet connection may be slow or unreliable. Some signs of an unstable internet connection may be freezing labs, difficulty connecting to virtual machines, or the inability to type or enter commands within the lab environment.
Pro Tip: If you are unable to complete a Qwiklab or Jupyter Notebooks lab on one device, try using another device.

Troubleshooting steps

To summarize, here are the troubleshooting steps to try if you encounter a problem with Qwiklabs or Jupyter Notebooks.

1. Make sure you are using the latest version of a supported browser: Google Chrome, Firefox, or Microsoft Edge.
2. Restart your browser and clear your browser's cache and cookies. You can also use incognito mode.
3. Check your internet connection and make sure it is stable. You can try restarting your router and modem to regain a stable connection.
4. Try restarting Qwiklabs or Jupyter Notebooks again.
5. **For Qwiklabs only:** If problems persist or you receive a message stating that you have exceeded the quota for a Qwiklab, submit this form to Qwiklabs support for assistance.

Introduction to the shell

Linux shell. This part of the Linux architecture is where the action will happen for you as a security analyst. We introduced the shell with other components of the Linux OS earlier, but let's take a deeper look at what the shell is and what it does.

The shell is the command-line interpreter. That means it helps you communicate with the operating system through the command line. Previously, we discussed a command-line interface. This is essentially the shell. The shell provides the command-line interface for you to interact with the OS. To tell the OS what to do, you enter commands into this interface. A command is an instruction telling the computer to do something. The shell communicates with the kernel to execute these commands.

Earlier, we discussed how the operating system helps humans and computers speak with each other. The shell is the part of the OS that allows you to do this. Think of this as a very helpful language interpreter between you and your system. Since you do not speak computer language or binary, you can't directly communicate with your system. This is where the shell comes in to help you. Your OS doesn't need the shell for most of its work, but it is an interface between you and what your system can offer. It allows you to perform math, run tests, and execute applications. More importantly, it allows you to combine these operations and connect applications to each other to perform complex and automated tasks.

Communicate through a shell

As you explored previously, the **shell** is the command-line interpreter. You can think of a shell as a translator between you and the computer system. Shells allow you to give commands to the computer and receive responses from it. When you enter a command into a shell, the shell executes many internal processes to interpret your command, send it to the kernel, and return your results.

Types of shells

The many different types of Linux shells include the following:

- Bourne-Again Shell (bash)
- C Shell (csh)
- Korn Shell (ksh)
- Enhanced C shell (tcsh)
- Z Shell (zsh)

All Linux shells use common Linux commands, but they can differ in other features. For example, ksh and bash use the dollar sign (**$**) to indicate where users type in their commands. Other shells, such as zsh, use the percent sign (**%**) for this purpose.

Bash

Bash is the default shell in most Linux distributions. It's considered a user-friendly shell. You can use bash for basic Linux commands as well as larger projects.
Bash is also the most popular shell in the cybersecurity profession. You'll use bash throughout this course as you learn and practice Linux commands.

Input and output in the shell

Communicating with a computer is like having a conversation with your friend. One person asks a question and the other person answers with a response. If you don't know the answer, you can just say you don't know the answer. When you communicate with the shell, the commands in the shell can take input, give output, or give error messages.

Let's explore standard input, standard output, and error messages in more detail. Standard input consists of information received by the OS via the command line. This is like you asking your

friend a question during a conversation. The information is input from your keyboard to the shell. If the shell can interpret your request, it asks the kernel for the resources it needs to execute the related task.

Let's take a look at this through echo, a Linux command that outputs a specified string of text. String data is data consisting of an ordered sequence of characters. In our example, we'll just have it output the string of: hello. So, as input, we'll type: echo hello into the shell. Later, when we press enter, we'll get the output. But before we do that, let's first discuss the concept of output in more detail.

Standard output is the information returned by the OS through the shell. In the same way that your friend gives an answer to your question, output is a computer's response to the command you input. Output is what you receive. Let's pick up where we left off in our example and send the input of: echo hello to the OS by pressing enter. Immediately, the shell returns the output of: hello.

Finally, standard error contains error messages returned by the OS through the shell. Just like your friend might indicate that they can't answer a question, the system responds with an error message if they can't respond to your command. Sometimes this might occur when we misspell a command or the system doesn't know the response to the command. Other times, it might happen because we don't have the appropriate permissions to perform a command.

We'll explore another example that demonstrates standard error. Let's input: eco hello into the shell. Notice I intentionally misspelled echo as e-c-o. When we press enter, an error message appears.

Terms and definitions from Course 4, Module 2

- **Application:** A program that performs a specific task
- **Bash:** The default shell in most Linux distributions
- **CentOS:** An open-source distribution that is closely related to Red Hat
- **Central Processing Unit (CPU):** A computer's main processor, which is used to perform general computing tasks on a computer
- **Command:** An instruction telling the computer to do something
- **Digital forensics:** The practice of collecting and analyzing data to determine what has happened after an attack
- **Directory:** A file that organizes where other files are stored
- **Distributions:** The different versions of Linux
- **File path:** The location of a file or directory
- **Filesystem Hierarchy Standard (FHS):** The component of the Linux OS that organizes data
- **Graphical user interface (GUI):** A user interface that uses icons on the screen to manage different tasks on the computer
- **Hard drive:** A hardware component used for long-term memory
- **Hardware**: The physical components of a computer
- **Internal hardware:** The components required to run the computer
- **Kali Linux** ™: An open-source distribution of Linux that is widely used in the security industry
- **Kernel:** The component of the Linux OS that manages processes and memory
- **Linux:** An open source operating system
- **Package:** A piece of software that can be combined with other packages to form an application
- **Package manager:** A tool that helps users install, manage, and remove packages or applications
- **Parrot:** An open-source distribution that is commonly used for security
- **Penetration test (pen test):** A simulated attack that helps identify vulnerabilities in systems, networks, websites, applications, and processes
- **Peripheral devices:** Hardware components that are attached and controlled by the computer system
- **Random Access Memory (RAM):** A hardware component used for short-term memory
- **Red Hat® Enterprise Linux®** (also referred to simply as Red Hat in this course)**:** A subscription-based distribution of Linux built for enterprise use

- **Shell:** The command-line interpreter
- **Standard error:** An error message returned by the OS through the shell
- **Standard input:** Information received by the OS via the command line
- **Standard output:** Information returned by the OS through the shell
- **String data:** Data consisting of an ordered sequence of characters
- **Ubuntu:** An open-source, user-friendly distribution that is widely used in security and other industries
- **User:** The person interacting with a computer

Module 3 - Linux Commands in the bash shell

Linux commands via the Bash shell

As a security analyst, you will work with server logs and you'll need to know how to navigate, manage and analyze files remotely without a graphical user interface. In addition, you'll need to know how to verify and configure users and group access. You'll also need to give authorization and set file permissions. That means that developing skills with the command line is essential for your work as a security analyst. When we learned about the Linux architecture, we learned that the shell is one of the main components of an operating system. We also learned that there are different shells. In this section, we'll utilize the Bash shell.

Bash is the default shell in most Linux distributions. For the most part, the key Linux commands that you'll be learning in this section are the same across shells. Now that you know what shell you'll be using, let's go into how to write in Bash. As we discussed in a previous section, communicating with your OS is like a conversation. You type in commands, and the OS responds with an answer to your command. A command is an instruction telling the computer to do something.

We'll try out a command in Bash. Notice a dollar sign before the cursor. This is your prompt to enter a new command. Some commands might tell the computer to find something like a specific file. Others might tell it to launch a program. Or, it might be to output a specific string of text. In the last section, when we discussed input and output, we explored how the echo command did this.

Let's input the echo command again. You may notice that the command we just input is not complete. If we're going to use the echo command to output a specific string of texts, we need to specify what the string of text is. This is what arguments are for. An argument is specific information needed by a command. Some commands take multiple arguments. So now let's complete the echo command with an argument. We're learning some pretty technical stuff, so how about we output the words: "You are doing great!" We'll add this argument, and then we'll press enter to get the output.

Core commands for navigation and reading files

Now, I want you to imagine a tree. What did you notice first about the tree? Would you say the trunk or the branches? These might definitely get your attention, but what about its roots? Everything about a tree starts in the roots. Something similar happens when we think about the Linux file system.

Previously, we learned about the components of the Linux architecture. The Filesystem Hierarchy Standard, or FHS, is the component of the Linux OS that organizes data. This file system is a very important part of Linux because everything we do in Linux is considered a file somewhere

in the system's directory. The FHS is a hierarchical system, and just like with a tree, everything grows and branches out from the root. The root directory is the highest-level directory in Linux. It's designated by a single slash. Subdirectories branch off from the root directory. The subdirectories branch out further and further away from the root directory. When describing the directory structure in Linux, slashes are used when tracing back through these branches to the root. For example, here, the first slash indicates the root directory. Then it branches out a level into the home subdirectory. Another slash indicates it is branching out again. This time it's to the analyst subdirectory that is located within home. When working in security, it is essential that you learn to navigate a file system to locate and analyze logs, such as log files. You'll analyze these log files for application usage and authentication.

With that background, we're now ready to learn the commands commonly used for navigating the file system. First, pwd prints the working directory onto the screen. When you use this command, the output tells you which directory you're currently in. Next, ls displays the names of files and directories in the current working directory. And finally, cd navigates between directories. This is the command you'll use when you want to change directories.

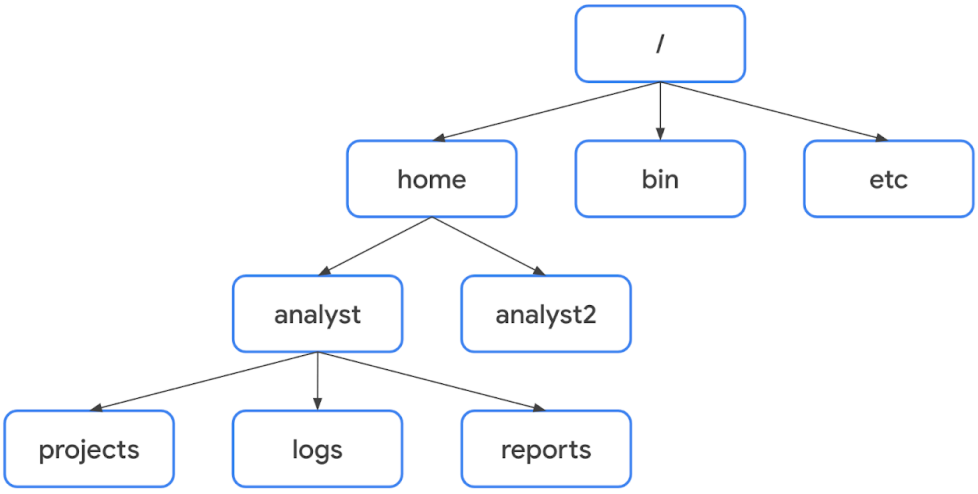
Let's use these commands in Bash. First, we'll type the command pwd to display the current location and then press enter. The output is the path to the analyst directory where we're currently working. Next, let's input ls to display the files and directories within the analyst directory. The output is the name of four directories: logs, old reports, projects, and reports, and one file named updates.txt. So let's say we now want to go into the logs directory to check for unauthorized access. We'll input: cd logs to change directories. We won't get any output on the screen from the cd command, but if we enter pwd again, its output indicates that the working directory is logs. Logs is a subdirectory of the analyst directory.

As a security analyst, you'll also need to know how to read file content in Linux. For example, you may need to read files that contain configuration settings to identify potential vulnerabilities. Or, you might look at user access reports while investigating unauthorized access. When reading file content, there are some commands that will help you. First, cat displays the content of a file. This is useful, but sometimes you won't want the full contents of a large file. In these cases, you can use the head command. It displays just the beginning of a file, by default ten lines.

Let's try out these commands. Imagine that we want to read the contents of access.txt, and we're already in the working directory where it's located. First, we input the cat command and then follow it with the name of the file, access.txt. And Bash returns the full contents of this file. Let's compare that to the head command. When we input the head command followed by our file name, only the first 10 lines of this file are displayed.

Filesystem Hierarchy Standard (FHS)

Previously, you learned that the **Filesystem Hierarchy Standard (FHS)** is the component of Linux that organizes data. The FHS is important because it defines how directories, directory contents, and other storage is organized in the operating system.
This diagram illustrates the hierarchy of relationships under the FHS:



Under the FHS, a file's location can be described by a file path. A **file path** is the location of a file or directory. In the file path, the different levels of the hierarchy are separated by a forward slash (**/**).

Root directory

The **root directory** is the highest-level directory in Linux, and it's always represented with a forward slash (**/**). All subdirectories branch off the root directory. Subdirectories can continue branching out to as many levels as necessary.

Standard FHS directories

Directly below the root directory, you'll find standard FHS directories. In the diagram, **home**, **bin**, and **etc** are standard FHS directories. Here are a few examples of what standard directories contain:

- **/home**: Each user in the system gets their own home directory.
- **/bin**: This directory stands for "binary" and contains binary files and other executables. Executables are files that contain a series of commands a computer needs to follow to run programs and perform other functions.
- **/etc**: This directory stores the system's configuration files.
- **/tmp**: This directory stores many temporary files. The **/tmp** directory is commonly used by attackers because anyone in the system can modify data in these files.
- **/mnt**: This directory stands for "mount" and stores media, such as USB drives and hard drives.

Pro Tip: You can use the **man hier** command to learn more about the FHS and its standard directories.

User-specific subdirectories

Under **home** are subdirectories for specific users. In the diagram, these users are **analyst** and **analyst2**. Each user has their own personal subdirectories, such as **projects**, **logs**, or **reports**.
Note: When the path leads to a subdirectory below the user's home directory, the user's home directory can be represented as the tilde (**~**). For example, **/home/analyst/logs** can also be represented as **~/logs**.
You can navigate to specific subdirectories using their absolute or relative file paths. The **absolute file path** is the full file path, which starts from the root. For example, **/home/analyst/projects** is an absolute file path. The **relative file path** is the file path that starts from a user's current directory.

Note: Relative file paths can use a dot (**.**) to represent the current directory, or two dots (**..**) to represent the parent of the current directory. An example of a relative file path could be **../projects**.

Key commands for navigating the file system

The following Linux commands can be used to navigate the file system: **pwd**, **ls**, and **cd**.

pwd

The **pwd** command prints the working directory to the screen. Or in other words, it returns the directory that you're currently in.
The output gives you the absolute path to this directory. For example, if you're in your **home** directory and your username is **analyst**, entering **pwd** returns **/home/analyst**.
Pro Tip: To learn what your username is, use the **whoami** command. The **whoami** command returns the username of the current user. For example, if your username is **analyst**, entering **whoami** returns **analyst**.

ls

The **ls** command displays the names of the files and directories in the current working directory. For example, in the video, **ls** returned directories such as **logs**, and a file called **updates.txt**.
Note: If you want to return the contents of a directory that's not your current working directory, you can add an argument after **ls** with the absolute or relative file path to the desired directory. For example, if you're in the **/home/analyst** directory but want to list the contents of its **projects** subdirectory, you can enter **ls /home/analyst/projects** or just **ls projects**.

cd

The **cd** command navigates between directories. When you need to change directories, you should use this command.
To navigate to a subdirectory of the current directory, you can add an argument after **cd** with the subdirectory name. For example, if you're in the **/home/analyst** directory and want to navigate to its **projects** subdirectory, you can enter **cd projects**.
You can also navigate to any specific directory by entering the absolute file path. For example, if you're in **/home/analyst/projects**, entering **cd /home/analyst/logs** changes your current directory to **/home/analyst/logs**.
Pro Tip: You can use the relative file path and enter **cd ..** to go up one level in the file structure. For example, if the current directory is **/home/analyst/projects**, entering **cd ..** would change your working directory to **/home/analyst**.

Common commands for reading file content

The following Linux commands are useful for reading file content: **cat**, **head**, **tail**, and **less**.

cat

The **cat** command displays the content of a file. For example, entering **cat updates.txt** returns everything in the **updates.txt** file.

head

The **head** command displays just the beginning of a file, by default 10 lines. The **head** command can be useful when you want to know the basic contents of a file but don't need the full contents. Entering **head updates.txt** returns only the first 10 lines of the **updates.txt** file.
Pro Tip: If you want to change the number of lines returned by **head**, you can specify the number of lines by including **-n**. For example, if you only want to display the first five lines of the **updates.txt** file, enter **head -n 5 updates.txt**.

tail

The **tail** command does the opposite of **head**. This command can be used to display just the end of a file, by default 10 lines. Entering **tail updates.txt** returns only the last 10 lines of the **updates.txt** file.
Pro Tip: You can use **tail** to read the most recent information in a log file.

less

The **less** command returns the content of a file one page at a time. For example, entering **less updates.txt** changes the terminal window to display the contents of **updates.txt** one page at a time. This allows you to easily move forward and backward through the content.
Once you've accessed your content with the **less** command, you can use several keyboard controls to move through the file:

- **Space bar**: Move forward one page
- **b**: Move back one page
- **Down arrow**: Move forward one line
- **Up arrow**: Move back one line
- **q**: Quit and return to the previous terminal window

Manage file content in Bash

Find what you need with Linux

As a security analyst, your work will likely involve filtering for the information you need. Filtering means searching your system for specific information that can help you solve complex problems. For example, imagine that your team determines a piece of malware contains a string of characters. You might be tasked with finding other files with the same string to determine if those files contain the same malware. Later, we'll learn more about how you can use SQL to filter a database, but Linux is a good place to start basic filtering.

First, we'll start with grep. The grep command searches a specified file and returns all lines in the file containing a specified string. Here's an example of this. Let's say we have a file called updates.txt, and we're currently looking for lines that contain the word: OS. If the file is large, it would take a long time to visually scan for this. Instead, after navigating to the directory that contains updates.txt, we'll type the command: grep OS updates.txt into the shell. Notice how the grep command is followed by two arguments. The first argument is the string we're searching for; in this case, OS. The second argument is the name of the file we're searching through, updates.txt. When we press enter, Bash returns all lines containing the word OS.

Now let's talk about piping. Piping is a Linux command that can be used for a variety of purposes. In a moment, we'll focus on how it can be used for filtering. But first, let's talk about the general idea of piping. The piping command sends a standard output of one command as standard input into another command for further processing. It's represented by the vertical bar character. In our context, we can refer to this as the pipe character. Take a moment and imagine a physical pipe. Physical pipes have two ends. On one end, for example, water might enter the pipe from a hot water tank. Then, it travels through the pipe and comes out on the other end in a sink. Similarly, in Linux, piping also involves redirection. Output from one command is sent through the pipe and then is used on the other side of the pipe. Earlier in this video, I explained how grep can be used to filter for strings of characters within a file. Grep can also be incorporated after a pipe.

Let's focus on this example. The first command, ls, instructs the operating system to output the file and directory contents of their reports subdirectory. But because the command is followed by the pipe, the output isn't returned to the screen. Instead, it's sent to the next command. As we just learned, grep searches for a specified string of characters; in this case, it's users. But where is it searching? Since grep follows a pipe, the output of the previous command indicates where to search. In this case, that output is a list of files and directories within the reports subdirectory. It will return all files and directories that contain the word: users.

Let's explore this in Bash. So we can better understand how the filter works, let's first output everything in the reports directory. If we were already in the directory, we would just need to input ls. But since we're not, we'll also specify the path to this directory. When we press enter, the output indicates there are seven files in the reports directory. Because we want to return only the files that contain the word users, we'll combine this ls command with piping and the grep command. As the output demonstrates, Linux has been instructed to return only files that contain the word users. The two files that don't contain this string no longer appear.

Filter content in Linux

Filtering for information

You previously explored how filtering for information is an important skill for security analysts. **Filtering** is selecting data that match a certain condition. For example, if you had a virus in your system that only affected the **.txt** files, you could use filtering to find these files quickly. Filtering allows you to search based on specific criteria, such as file extension or a string of text.

grep

The **grep** command searches a specified file and returns all lines in the file containing a specified string. The **grep** command commonly takes two arguments: a specific string to search for and a specific file to search through.
For example, entering **grep OS updates.txt** returns all lines containing **OS** in the **updates.txt** file. In this example, **OS** is the specific string to search for, and **updates.txt** is the specific file to search through.

Piping

The pipe command is accessed using the pipe character (**|**). **Piping** sends the standard output of one command as standard input to another command for further processing. As a reminder, **standard output** is information returned by the OS through the shell, and **standard input** is information received by the OS via the command line.
The pipe character (**|**) is located in various places on a keyboard. On many keyboards, it's located on the same key as the backslash character (****). On some keyboards, the **|** can look different and have a small space through the middle of the line. If you can't find the **|**, search online for its location on your particular keyboard.
When used with **grep**, the pipe can help you find directories and files containing a specific word in their names. For example, **ls /home/analyst/reports | grep users** returns the file and directory names in the **reports** directory that contain **users**. Before the pipe, **ls** indicates to list the names of the files and directories in **reports**. Then, it sends this output to the command after the pipe. In this case, **grep users** returns all of the file or directory names containing **users** from the input it received.
Note: Piping is a general form of redirection in Linux and can be used for multiple tasks other than filtering. You can think of piping as a general tool that you can use whenever you want the output of one command to become the input of another command.

find

The **find** command searches for directories and files that meet specified criteria. There's a wide range of criteria that can be specified with **find**. For example, you can search for files and directories that

- Contain a specific string in the name,
- Are a certain file size, or
- Were last modified within a certain time frame.

When using **find**, the first argument after **find** indicates where to start searching. For example, entering **find /home/analyst/projects** searches for everything starting at the **projects** directory.
After this first argument, you need to indicate your criteria for the search. If you don't include a specific search criteria with your second argument, your search will likely return a lot of directories and files.
Specifying criteria involves options. **Options** modify the behavior of a command and commonly begin with a hyphen (**-**).

-name and -iname

One key criteria analysts might use with **find** is to find file or directory names that contain a specific string. The specific string you're searching for must be entered in quotes after the **-name** or **-iname** options. The difference between these two options is that **-name** is case-sensitive, and **-iname** is not.
For example, you might want to find all files in the **projects** directory that contain the word "log" in the file name. To do this, you'd enter **find /home/analyst/projects -name "*log*"**. You could also enter **find /home/analyst/projects -iname "*log*"**.
In these examples, the output would be all files in the **projects** directory that contain **log** surrounded by zero or more characters. The **"*log*"** portion of the command is the search criteria that indicates to search for the string "log". When **-name** is the option, files with names

that include **Log** or **LOG**, for example, wouldn't be returned because this option is case-sensitive. However, they would be returned when **-iname** is the option.
Note: An asterisk (*****) is used as a wildcard to represent zero or more unknown characters.

-mtime

Security analysts might also use **find** to find files or directories last modified within a certain time frame. The **-mtime** option can be used for this search. For example, entering **find /home/analyst/projects -mtime -3** returns all files and directories in the **projects** directory that have been modified within the past three days.
The **-mtime** option search is based on days, so entering **-mtime +1** indicates all files or directories last modified more than one day ago, and entering **-mtime -1** indicates all files or directories last modified less than one day ago.
Note: The option **-mmin** can be used instead of **-mtime** if you want to base the search on minutes rather than days.

Create and modify directories and files

The subdirectories are the branches of the tree. They're all connected from the same root but can grow to make a complex tree. In this video, we'll create directories and files and learn how to modify them.

When it comes to working with data in security, organization is key. If we know where information is located, it makes it easier to detect issues and keep information safe. In a previous video, we've already discussed navigating between directories, but let's take a moment to examine directories more closely. It's possible you're familiar with the concept of folders for organizing information. In Linux, we have directories. Directories help organize files and subdirectories. For example, within a directory for reports, an analyst may need to create two subdirectories: one for drafts and one for final reports.

Now that we know why we need directories, let's take a look at some essential Linux commands for managing directories and files. First, let's take note of commands for creating and removing directories. The mkdir command creates a new directory. In contrast, rmdir removes or deletes a directory. A helpful feature of this command is its built-in warning that lets you know a directory is not empty. This saves you from accidentally deleting files. Next, you'll use other commands for creating and removing files. The touch command creates a new file, and then the rm command removes or deletes a file. And last, we have our commands for copying and moving files or directories. The mv command moves a file or directory to new location, and cp copies a file or directory into a new location.

Now, we're ready to try out these commands. First, let's use the pwd command, and then let's display the names of the files and directories in the analyst directory with the ls command. Imagine that we no longer need the old reports directory that appears among the file contents. Let's take a look at how to remove it. We input the rmdir command and follow it with the name of the directory we want to remove: oldreports. We can use the ls command to confirm that old reports has been deleted and no longer appear among the contents.

Now, let's make another change. We want a new directory for drafts of reports. We need to use the command: mkdir and specify a name for this directory: drafts. If we input ls again, we'll notice the new directory drafts included among the contents of the analyst directory. Let's change into this new directory by entering: cd drafts. If we run ls, it doesn't return any output, indicating

that this directory is currently empty. But next, we'll add some files to it. Let's say we want to draft new reports on recently installed email and OS patches. To create these files, we input: touch email_patches.txt and then: touch OS_patches.txt.

Running ls indicates that these files are now in the drafts directory. What if we realize that we only need a new report on OS patches and we want to delete the email patches report? To do this, we input the rm command and specify the file to delete as: email_patches.txt. Running ls confirms that it's been deleted. Now, let's focus on our commands for moving and copying. We realized that we have a file called email policy in the reports folder that is currently in draft format. We want to move it into the newly created drafts folder. To do this, we need to change into the directory that currently has that file.

Running ls in that directory indicates that it contains several files, including email_policy.txt. Then to move that file, we'll enter the mv command followed by two arguments. The first argument after mv identifies the file to be moved. The second argument indicates where to move it. If we change directories into drafts and then display its contents, we'll notice that the email policy file has been moved to this directory. We'll change back into reports. Displaying the file contents confirms that email_policy is no longer there.

Okay, one more thing. vulnerabilities.txt is a file that we want to keep in the reports directory. But since it affects an upcoming project, we also want to copy it into the project's directory. Since we're already in the directory that has this file, we'll use the cp command to copy it into the projects directory. Notice that the first argument indicates which file to copy, and the second argument provides the path to the directory that it will be copied into. When we press Enter, this copies the vulnerabilities file into the projects directory while also leaving the original within reports. Isn't it cool what we can do with these commands?

Now, let's focus on one more concept related to modifying files. In addition to using commands, you can also use applications to help you edit files. As a security analyst, file editors are often necessary for your daily tasks, like writing or editing reports. A popular file editor is nano. It's good for beginners. You can access this tool through the nano command. Let's get familiar with nano together. We'll add a title to our new draft report: OS_patches.txt. First, we change into the directory containing that file, then we input nano followed by the name of the file we want to edit: OS_patches.txt. This brings up the nano file editor with that file open. For now, we'll just enter the title OS Patches by typing this into the editor. We need to save this before returning to the command line, and to do so, we press Ctrl+O and then enter to save it with the current file name. Then to exit, we press Ctrl+X.

Manage directories and files

Creating and modifying directories

mkdir

The **mkdir** command creates a new directory. Like all of the commands presented in this reading, you can either provide the new directory as the absolute file path, which starts from the root, or as a relative file path, which starts from your current directory.
For example, if you want to create a new directory called **network** in your **/home/analyst/logs** directory, you can enter **mkdir /home/analyst/logs/network** to create this new directory. If you're already in the **/home/analyst/logs** directory, you can also create this new directory by entering **mkdir network**.
Pro Tip: You can use the **ls** command to confirm the new directory was added.

rmdir

The **rmdir** command removes, or deletes, a directory. For example, entering **rmdir /home/analyst/logs/network** would remove this empty directory from the file system.
Note: The **rmdir** command cannot delete directories with files or subdirectories inside. For example, entering **rmdir /home/analyst** returns an error message.

Creating and modifying files

touch and rm

The **touch** command creates a new file. This file won't have any content inside. If your current directory is **/home/analyst/reports**, entering **touch permissions.txt** creates a new file in the **reports** subdirectory called **permissions.txt**.
The **rm** command removes, or deletes, a file. This command should be used carefully because it's not easy to recover files deleted with **rm**. To remove the permissions file you just created, enter **rm permissions.txt**.
Pro Tip: You can verify that **permissions.txt** was successfully created or removed by entering **ls**.

mv and cp

You can also use **mv** and **cp** when working with files. The **mv** command moves a file or directory to a new location, and the **cp** command copies a file or directory into a new location. The first argument after **mv** or **cp** is the file or directory you want to move or copy, and the second argument is the location you want to move or copy it to.
To move **permissions.txt** into the **logs** subdirectory, enter **mv permissions.txt /home/analyst/logs**. Moving a file removes the file from its original location. However, copying a file doesn't remove it from its original location. To copy **permissions.txt** into the **logs** subdirectory while also keeping it in its original location, enter **cp permissions.txt /home/analyst/logs**.
Note: The **mv** command can also be used to rename files. To rename a file, pass the new name in as the second argument instead of the new location. For example, entering **mv permissions.txt perm.txt** renames the **permissions.txt** file to **perm.txt**.

nano text editor

nano is a command-line file editor that is available by default in many Linux distributions. Many beginners find it easy to use, and it's widely used in the security profession. You can perform multiple basic tasks in nano, such as creating new files and modifying file contents.
To open an existing file in nano from the directory that contains it, enter **nano** followed by the file name. For example, entering **nano permissions.txt** from the **/home/analyst/reports** directory opens a new nano editing window with the **permissions.txt** file open for editing. You can also provide the absolute file path to the file if you're not in the directory that contains it.
You can also create a new file in nano by entering **nano** followed by a new file name. For example, entering **nano authorized_users.txt** from the **/home/analyst/reports** directory creates the **authorized_users.txt** file within that directory and opens it in a new nano editing window.

Since there isn't an auto-saving feature in nano, it's important to save your work before exiting. To save a file in nano, use the keyboard shortcut **Ctrl + O**. You'll be prompted to confirm the file name before saving. To exit out of nano, use the keyboard shortcut **Ctrl + X**.
Note: Vim and Emacs are also popular command-line text editors.

Standard output redirection

There's an additional way you can write to files. Previously, you learned about standard input and standard output. **Standard input** is information received by the OS via the command line, and **standard output** is information returned by the OS through the shell.
You've also learned about piping. **Piping** sends the standard output of one command as standard input to another command for further processing. It uses the pipe character (**|**).
In addition to the pipe (**|**), you can also use the right angle bracket (**>**) and double right angle bracket (**>>**) operators to redirect standard output.
When used with **echo**, the **>** and **>>** operators can be used to send the output of **echo** to a specified file rather than the screen. The difference between the two is that **>** overwrites your existing file, and **>>** adds your content to the end of the existing file instead of overwriting it. The **>** operator should be used carefully, because it's not easy to recover overwritten files.
When you're inside the directory containing the **permissions.txt** file, entering **echo "last updated date" >> permissions.txt** adds the string "last updated date" to the file contents. Entering **echo "time" > permissions.txt** after this command overwrites the entire file contents of **permissions.txt** with the string "time".
Note: Both the **>** and **>>** operators will create a new file if one doesn't already exist with your specified name.

Authenticate and authorize users

File permissions and ownership

Permissions are the type of access granted for a file or directory. Permissions are related to authorization. Authorization is the concept of granting access to specific resources in a system. Authorization allows you to limit access to specified files or directories. A good rule to follow is that data access is on a need-to-know basis. You can imagine the security risk it would impose if anyone could access or modify anything they wanted to on a system.

There are three types of permissions in Linux that an authorized user can have. The first type of permission is read. On a file, read permissions means contents on the file can be read. On a directory, this permission means you can read all files in that directory. Next are write permissions. Write permissions on a file allow modifications of contents of the file. On a directory, write permissions indicate that new files can be created in that directory. Finally, there are also execute permissions. Execute permissions on files mean that the file can be executed if it's an executable file. Execute permissions on directories allow users to enter into a directory and access its files.

Permissions are granted for three different types of owners. The first type is the user. The user is the owner of the file. When you create a file, you become the owner of the file, but the ownership can be changed. Group is the next type. Every user is a part of a certain group. A group consists of several users, and this is one way to manage a multi-user environment. Finally, there is another. Others can be considered all other users on the system. Basically, anyone else with access

to the system belongs to this group. In Linux, file permissions are represented with a 10-character string. For a directory with full permissions for the user group, this string would be: drwxrwxrwx.

Let's examine what this means more closely. The first character indicates the file type. As shown in this example, d is used to indicate it is a directory. If this character contains a hyphen instead, it would be a regular file. The second, third, and fourth characters indicate the permissions for the user. In this example, r indicates the user has read permissions, w indicates the user has write permissions, and x indicates the user has execute permissions. If one of these permissions was missing, there would be a hyphen instead of the letter. In the same way, the fifth, sixth, and seventh characters indicate permissions for the next owner type group. As it shows here, the type group also has read, write, and execute permissions. There are no hyphens to indicate that any of these permissions haven't been granted. Finally, the eighth through tenth characters indicate permissions for the last owner type: other. They also have read, write, and execute permissions in this example.

Ensuring files and directories are set with their appropriate access permissions is critical to protecting sensitive files and maintaining the overall security of a system. For example, payroll departments handle sensitive information. If someone outside of the payroll group could read this file, this would be a privacy concern. Another example is when the user, the group, and other can all write to a file. This type of file is considered a world-writable file. World-writable files can pose significant security risks.

So how do we check permissions? First, we need to understand what options are. Options modify the behavior of the command. The options for a command can be a single letter or a full word. Checking permissions involves adding options to the ls command. First, ls -l displays permissions to files and directories. You might also want to display hidden files and identify their permissions. Hidden files, which begin with a period before their name, don't normally appear when you use ls to display file contents. Entering ls -a displays hidden files. Then you can combine these two options to do both. Entering ls -la displays permissions to files and directories, including hidden files.

Let's get into Bash and try out these options. Right now, we're in the project subdirectory. First, let's use the ls command to display its contents. The output displays the files in this directory, but we don't know anything about their permissions. By using ls -l instead, we get expanded information on these files. Let's do this. The file names are now on the right side of each row. The first piece of information in each row shows the permissions in the format that we discussed earlier. Since these are all files and not directories, notice how the first character is a hyphen. Let's focus on one specific file: project1.txt. The second through fourth characters of its permissions show us the user has both read and write permissions but lacks execute permissions. In both the fifth through seventh characters and eighth through tenth characters, the sequence is r--. This means group and others have only read privileges. After the permissions, ls -l first displays the username. Here, that's us, analysts. Next comes the group name; in our case, the security group. Now let's use ls -a The output includes two more files—hidden files with the names: .hidden1.txt and .hidden2.txt Finally, we can also use ls -la to show the permissions for all files, including these hidden files.

Change permissions

When working as a security analyst, there may be many reasons to change permissions for a user. A user may have changed departments or been assigned to a different work group. A user might simply no longer be working on a project that requires certain permissions. These changes are necessary in order to protect system files from being accidentally or deliberately altered or deleted.

Let's explore a related command that helps control this access. In this video, we'll learn about chmod. chmod changes permissions on files and directories. The command chmod stands for change mode.

There are two modes for changing permissions, but we'll focus on symbolic. The best way to learn about how chmod works is through an example. I know this has a lot of details, but we'll break this down. Also, please keep in mind that, like many Linux commands, you don't have to memorize the information and can always find a reference.

With chmod, you need to identify which file or directory you want to adjust permissions for. This is the final argument, in this case, a file named: access.txt. The first argument, added directly after the chmod command, indicates how to change permissions. Right now, this might seem hard to interpret, but soon we'll understand why this is called symbolic mode.

Previously, we learned about the three types of owners: user, group, and other. To identify these with chmod, we use u to represent the user, g to represent the group, and o to represent others. In this particular example, g indicates we will make some changes to group permissions, and o to permissions for others. These owner types are separated by a comma in this argument.

But do we want to add or take away permissions? Well, for this, we use mathematical operators. So, the plus sign after g means we want to add permissions for the group. The minus sign after o means we want to take them away from others. And the last question is: what kind of changes? We've already learned that r represents read permissions, w represents write permissions, and x represents execute permissions. So in this case, the w indicates that we're adding write permissions to the group, and r indicates that we are taking away read permissions from others. This is still very complex. But now that we've broken it down, perhaps it doesn't seem quite so much like a foreign language. And remember, you don't have to memorize this all.

Let's give this new command a try. We'll start out in the logs sub-directory. If we use the ls -l command, it will output the permissions for the file. It shows the permissions for the only file in this directory: access.txt. Previously, we learned how to read these permissions. The second through fourth characters indicate that the user has read and write permissions. The fifth through seventh characters show the group only has read permissions. And the eighth to tenth characters show that others only have read permissions. We need to adjust these permissions.

We want to ensure analysts in the security group have write permission, but take away read permissions from the owner-type other, so we add write permissions for one group and remove read permissions for others. Let's run ls -l again. This shows a change in the permissions for access.txt. Notice how in the middle segment of permissions for the group, w has been added to give write permissions. And another change is that the r has been removed in the last segment, indicating that read permissions for others have been removed. As mentioned earlier, these hyphens indicate a lack of permissions. Now, others are lacking all permissions.

Permission commands

Reading permissions

In Linux, permissions are represented with a 10-character string. Permissions include:

- **read**: for files, this is the ability to read the file contents; for directories, this is the ability to read all contents in the directory including both files and subdirectories
- **write**: for files, this is the ability to make modifications on the file contents; for directories, this is the ability to create new files in the directory
- **execute**: for files, this is the ability to execute the file if it’s a program; for directories, this is the ability to enter the directory and access its files

These permissions are given to these types of owners:

- **user**: the owner of the file
- **group**: a larger group that the owner is a part of
- **other**: all other users on the system

Each character in the 10-character string conveys different information about these permissions. The following table describes the purpose of each character:

Character	Example	Meaning
1st	**d**rwxrwxrwx	file type • **d** for directory • **-** for a regular file
2nd	d**r**wxrwxrwx	read permissions for the user • **r** if the user has read permissions • **-** if the user lacks read permissions
3rd	dr**w**xrwxrwx	write permissions for the user • **w** if the user has write permissions • **-** if the user lacks write permissions
4th	drw**x**rwxrwx	execute permissions for the user • **x** if the user has execute permissions • **-** if the user lacks execute permissions
5th	drwx**r**wxrwx	read permissions for the group • **r** if the group has read permissions • **-** if the group lacks read permissions
6th	drwxr**w**xrwx	write permissions for the group • **w** if the group has write permissions • **-** if the group lacks write permissions
7th	drwxrw**x**rwx	execute permissions for the group • **x** if the group has execute permissions • **-** if the group lacks execute permissions
8th	drwxrwx**r**wx	read permissions for other • **r** if the other owner type has read permissions • **-** if the other owner type lacks read permissions

9th	drwxrwxr**w**x	write permissions for other • **w** if the other owner type has write permissions • **-** if the other owner type lacks write permissions
10th	drwxrwxrw**x**	execute permissions for other • **x** if the other owner type has execute permissions • **-** if the other owner type lacks execute permissions

Exploring existing permissions

You can use the **ls** command to investigate who has permissions on files and directories. Previously, you learned that **ls** displays the names of files in directories in the current working directory.

There are additional options you can add to the **ls** command to make your command more specific. Some of these options provide details about permissions. Here are a few important **ls** options for security analysts:

- **ls -a**: Displays hidden files. Hidden files start with a period (**.**) at the beginning.
- **ls -l**: Displays permissions to files and directories. Also displays other additional information, including owner name, group, file size, and the time of last modification.
- **ls -la**: Displays permissions to files and directories, including hidden files. This is a combination of the other two options.

Changing permissions

The **principle of least privilege** is the concept of granting only the minimal access and authorization required to complete a task or function. In other words, users should not have privileges that are beyond what is necessary. Not following the principle of least privilege can create security risks.

The **chmod** command can help you manage this authorization. The **chmod** command changes permissions on files and directories.

Using chmod

The **chmod** command requires two arguments. The first argument indicates how to change permissions, and the second argument indicates the file or directory that you want to change permissions for. For example, the following command would add all permissions to **login_sessions.txt**:

chmod u+rwx,g+rwx,o+rwx login_sessions.txt

If you wanted to take all the permissions away, you could use

chmod u-rwx,g-rwx,o-rwx login_sessions.txt

Another way to assign these permissions is to use the equals sign (**=**) in this first argument. Using **=** with **chmod** sets, or assigns, the permissions exactly as specified. For example, the following command would set read permissions for **login_sessions.txt** for user, group, and other:

chmod u=r,g=r,o=r login_sessions.txt

This command overwrites existing permissions. For instance, if the user previously had write permissions, these write permissions are removed after you specify only read permissions with **=**.

The following table reviews how each character is used within the first argument of **chmod**:

Character	Description
u	indicates changes will be made to user permissions
g	indicates changes will be made to group permissions
o	indicates changes will be made to other permissions
+	adds permissions to the user, group, or other
-	removes permissions from the user, group, or other
=	assigns permissions for the user, group, or other

Note: When there are permission changes to more than one owner type, commas are needed to separate changes for each owner type. You should not add spaces after those commas.

The principle of least privilege in action

As a security analyst, you may encounter a situation like this one: There's a file called **bonuses.txt** within a compensation directory. The owner of this file is a member of the Human Resources department with a username of **hrrep1**. It has been decided that **hrrep1** needs access to this file. But, since this file contains confidential information, no one else in the **hr** group needs access.

You run **ls -l** to check the permissions of files in the compensation directory and discover that the permissions for **bonuses.txt** are **-rw-rw----**. The group owner type has read and write permissions that do not align with the principle of least privilege.

To remedy the situation, you input **chmod g-rw bonuses.txt**. Now, only the user who needs to access this file to carry out their job responsibilities can access this file.

Add and delete users

Authentication is the process of a user proving that they are who they say they are in the system.

Just like in a physical building, not all users should be allowed in. Not all users should get access to the system. But we also want to make sure everyone who should have access to the system has it. That's why we need to add users. New users can be new to the organization or new to a group. This could be related to a change in organizational structure or simply a directive from management to move someone.

And also, when users leave the organization, they need to be deleted. They should no longer have access to any part of the system. Or if they simply changed groups, they should be deleted from groups that they are no longer a part of.

Now that we've sorted out why it's important to add and delete users, let's discuss a different type of user, the root user. A root user, or superuser, is a user with elevated privileges to modify the system. Regular users have limitations, where the root does not.

Individuals who need to perform specific tasks can be temporarily added as root users. Root users can create, modify, or delete any file and run any program. Only root users or accounts with root privileges can add new users. So you may be wondering how you become a superuser. Well, one way is logging in as the root user, but running commands as the root user is considered to be bad practice when using Linux.

Why is running commands as a root user potentially problematic? The first problem with logging in as root is the security risks. Malicious actors will try to breach the root account. Since it's the most powerful account, to stay safe, the root account should have logins disabled. Another problem is that it's very easy to make irreversible mistakes.

It's very easy to type the wrong command in the CLI, and if you're running as the root user, you run a higher risk of making an irreversible mistake, such as permanently deleting a directory. Finally, there's the concern of accountability. In a multi-user environment like Linux, there are many users. If a user is running as root, there is no way to track who exactly ran a command. One solution to help solve this problem is sudo.

sudo is a command that temporarily grants elevated permissions to specific users. This provides more of a controlled approach compared to root, which runs every command with root privileges. sudo solves lots of problems associated with running as root.

sudo comes from super-user-do and lets you execute commands as an elevated user without having to sign in and out of another account. Running sudo will prompt you to enter the password for the user you're currently logged in as. Not all users on a system can become a superuser. Users must be granted sudo access through a configuration file called the sudoers file.

Now that we've learned about sudo, let's learn how we can use it with another command to add users. This command is useradd. useradd adds a user to the system. Only root or users with sudo privileges can use a useradd command. Let's look at a specific example in which we need to add a user. We'll imagine a new representative is joining the sales department and will be given the username of salesrep7. We're tasked with adding them to the system.

Let's try adding the new user. First, we need to use the sudo command, followed by the useradd command, and then last, the username we want to add, in this case, salesrep7. This command doesn't display anything on the screen. But since we get a new Bash cursor and not an error message, we can feel confident that the command worked successfully. If it didn't, an error message would have appeared. Sometimes an error has to do with something simple like misspelling useradd. Or, it might be because we didn't have sudo privileges.

Now let's learn how to do the opposite. Let's learn how to delete a user with userdel. userdel deletes a user from the system. Similarly, we need root permissions that we'll access through sudo to use userdel. Let's go back to our example of the user we added. Let's imagine two months later, the sales representative that we just added to the system leaves the company. That user should no longer have access to the system. Let's delete that user from the system.

Again, the sudo command is used first, then we add the userdel command. Last, we add the name of the user we want to delete. Again, we know it ran successfully because there is a new Bash cursor and not an error message.

Responsible use of sudo

To manage authorization and authentication, you need to be a **root user,** or a user with elevated privileges to modify the system. The root user can also be called the “super user.” You become a root user by logging in as the root user. However, running commands as the root user is not recommended in Linux because it can create security risks if malicious actors compromise that account. It’s also easy to make irreversible mistakes, and the system can’t track who ran a

command. For these reasons, rather than logging in as the root user, it's recommended you use **sudo** in Linux when you need elevated privileges.

The **sudo** command temporarily grants elevated permissions to specific users. The name of this command comes from "super user do." Users must be given access in a configuration file to use **sudo**. This file is called the "sudoers file." Although using **sudo** is preferable to logging in as the root user, it's important to be aware that users with the elevated permissions to use **sudo** might be more at risk in the event of an attack.

You can compare this to a hotel with a master key. The master key can be used to access any room in the hotel. There are some workers at the hotel who need this key to perform their work. For example, to clean all the rooms, the janitor would scan their ID badge and then use this master key. However, if someone outside the hotel's network gained access to the janitor's ID badge and master key, they could access any room in the hotel. In this example, the janitor with the master key represents a user using **sudo** for elevated privileges. Because of the dangers of **sudo**, only users who really need to use it should have these permissions.

Additionally, even if you need access to **sudo**, you should be careful about using it with only the commands you need and nothing more. Running commands with **sudo** allows users to bypass the typical security controls that are in place to prevent elevated access to an attacker.

Note: Be aware of **sudo** if copying commands from an online source. It's important you don't use **sudo** accidentally.

Authentication and authorization with sudo

You can use **sudo** with many authentication and authorization management tasks. As a reminder, **authentication** is the process of verifying who someone is, and **authorization** is the concept of granting access to specific resources in a system. Some of the key commands used for these tasks include the following:

useradd

The **useradd** command adds a user to the system. To add a user with the username of **fgarcia** with **sudo**, enter **sudo useradd fgarcia**. There are additional options you can use with **useradd**:

- **-g**: Sets the user's default group, also called their primary group
- **-G**: Adds the user to additional groups, also called supplemental or secondary groups

To use the **-g** option, the primary group must be specified after **-g**. For example, entering **sudo useradd -g security fgarcia** adds **fgarcia** as a new user and assigns their primary group to be **security**.

To use the **-G** option, the supplemental group must be passed into the command after **-G**. You can add more than one supplemental group at a time with the **-G** option. Entering **sudo useradd -G finance,admin fgarcia** adds **fgarcia** as a new user and adds them to the existing **finance** and **admin** groups.

usermod

The **usermod** command modifies existing user accounts. The same **-g** and **-G** options from the **useradd** command can be used with **usermod** if a user already exists.

To change the primary group of an existing user, you need the **-g** option. For example, entering **sudo usermod -g executive fgarcia** would change **fgarcia**'s primary group to the **executive** group.

To add a supplemental group for an existing user, you need the **-G** option. You also need a **-a** option, which appends the user to an existing group and is only used with the **-G** option. For

example, entering **sudo usermod -a -G marketing fgarcia** would add the existing **fgarcia** user to the supplemental **marketing** group.
Note: When changing the supplemental group of an existing user, if you don't include the **-a** option, **-G** will replace any existing supplemental groups with the groups specified after **usermod**. Using **-a** with **-G** ensures that the new groups are added but existing groups are not replaced.
There are other options you can use with **usermod** to specify how you want to modify the user, including:

- **-d**: Changes the user's home directory.
- **-l**: Changes the user's login name.
- **-L**: Locks the account so the user can't log in.

The option always goes after the **usermod** command. For example, to change **fgarcia**'s home directory to **/home/garcia_f**, enter **sudo usermod -d /home/garcia_f fgarcia**. The option **-d** directly follows the command **usermod** before the other two needed arguments.

userdel

The **userdel** command deletes a user from the system. For example, entering **sudo userdel fgarcia** deletes **fgarcia** as a user. Be careful before you delete a user using this command.
The **userdel** command doesn't delete the files in the user's home directory unless you use the **-r** option. Entering **sudo userdel -r fgarcia** would delete **fgarcia** as a user and delete all files in their home directory. Before deleting any user files, you should ensure you have backups in case you need them later.
Note: Instead of deleting the user, you could consider deactivating their account with **usermod -L**. This prevents the user from logging in while still giving you access to their account and associated permissions. For example, if a user left an organization, this option would allow you to identify which files they have ownership over, so you could move this ownership to other users.

chown

The **chown** command changes ownership of a file or directory. You can use **chown** to change user or group ownership. To change the user owner of the **access.txt** file to **fgarcia**, enter **sudo chown fgarcia access.txt**. To change the group owner of **access.txt** to **security**, enter **sudo chown :security access.txt**. You must enter a colon (**:**) before **security** to designate it as a group name. Similar to **useradd**, **usermod**, and **userdel**, there are additional options that can be used with **chown**.

Get help in Linux

Man pages within the shell

One of the great things about Linux is that you can get help right through the command line. The first command that can help you in this way is: man.

man displays information on other commands and how they work. The name of this command comes from the word manual. Let's examine this more closely by using man to get information about the usermod command. After man, we type the name of this command. The information that man returns includes a general description. It also contains information about each of usermod's options. For example, the option -d can be added to usermod to change a user's home directory. man provides a lot of information, but sometimes we just need a quick reference

on what a command does. In that case, you use whatis. whatis displays a description of a command on a single line.

Let's say you heard a co-worker mention a command like tail. You've never heard of this command before, but you can find out what it does. Simply use the command, whatis tail, and learn that it outputs the last part of files.

Sometimes we might not even know what command to look up. This is where apropos can help us. apropos searches the manual page descriptions for a specified string. Let's try it out. Let's say you have a task that requires you to change a password, but you're not quite sure how to do this. If we use the apropos command with the string password, this will display a large number of commands with that word. This helps somewhat, but it still may be difficult to find what we need. But we can filter this by adding the -a option and an additional string. This option will return only the commands that contain both strings. In our case, since we want to change the password, let's look for commands with both: change and password. Now, the output has been limited to the most relevant commands.

Linux community

Linux has a large online community, and this is a huge resource for Linux users of all levels. You can likely find the answers to your questions with a simple online search. Troubleshooting issues by searching and reading online is an effective way to discover how others approached your issue. It's also a great way for beginners to learn more about Linux.

The UNIX and Linux Stack Exchange is a trusted resource for troubleshooting Linux issues. The Unix and Linux Stack Exchange is a question and answer website where community members can ask and answer questions about Linux. Community members vote on answers, so the higher quality answers are displayed at the top. Many of the questions are related to specific topics from advanced users, and the topics might help you troubleshoot issues as you continue using Linux.

Integrated Linux support

Linux also has several commands that you can use for support.

man

The **man** command displays information on other commands and how they work. It's short for "manual." To search for information on a command, enter the command after **man**. For example, entering **man chown** returns detailed information about **chown**, including the various options you can use with it. The output of the **man** command is also called a "man page."

apropos

The **apropos** command searches the man page descriptions for a specified string. Man pages can be lengthy and difficult to search through if you're looking for a specific keyword. To use **apropos**, enter the keyword after **apropos**.

You can also include the **-a** option to search for multiple words. For example, entering **apropos -a graph editor** outputs man pages that contain both the words "graph" and "editor" in their descriptions.

whatis

The **whatis** command displays a description of a command on a single line. For example, entering **whatis nano** outputs the description of **nano**. This command is useful when you don't need a detailed description, just a general idea of the command. This might be as a reminder. Or, it might be after you discover a new command through a colleague or online resource and want to know more.

Terms and definitions from Course 4, Module 3

- **Absolute file path:** The full file path, which starts from the root
- **Argument (Linux):** Specific information needed by a command
- **Authentication:** The process of verifying who someone is
- **Authorization:** The concept of granting access to specific resources in a system
- **Bash:** The default shell in most Linux distributions
- **Command:** An instruction telling the computer to do something
- **File path:** The location of a file or directory
- **Filesystem Hierarchy Standard (FHS):** The component of the Linux OS that organizes data
- **Filtering:** Selecting data that match a certain condition
- **nano:** A command-line file editor that is available by default in many Linux distributions
- **Options:** Input that modifies the behavior of a command
- **Permissions:** The type of access granted for a file or directory
- **Principle of least privilege:** The concept of granting only the minimal access and authorization required to complete a task or function
- **Relative file path:** A file path that starts from the user's current directory
- **Root directory:** The highest-level directory in Linux
- **Root user (or superuser):** A user with elevated privileges to modify the system
- **Standard input:** Information received by the OS via the command line
- **Standard output:** Information returned by the OS through the shell

Reference guide: Linux

Table of contents

Navigate the file system

The following Linux commands are helpful when navigating the file system.

cd

Navigates between directories

> `cd reports`
> Navigates from the current working directory to its subdirectory `reports`
>
> `cd /home/analyst/reports`
> Navigates to the `reports` directory; the full path is required when `reports` is not a subdirectory of the current working directory
>
> `cd ..`
> Navigates to the directory that is one level above the current working directory

ls

Displays the names of the files and directories

> `ls`
> Displays the names of the files and directories in the current working directory
>
> `ls /home/analyst/reports`
> Displays the names of the files and directories in the `reports` directory; providing an argument that specifies the path to a directory is necessary to display the contents of a directory other than the user's current working directory
>
> `ls -a`
> Displays hidden files when displaying the names of files and directories inside the current working directory
>
> `ls -l`
> Displays permissions to files and directories in the current working directory; also displays other additional information, including owner name, group, file size, and the time of the last modification
>
> `ls -la`
> Displays permissions to files and directories in the current working directory, including hidden files; also displays other additional information, including owner name, group, file size, and the time of last modification

pwd

Prints the working directory to the screen

> `pwd`
> Prints the working directory to the screen, such as `/home/analyst`

whoami

Returns the username of the current user

> `whoami`
> Returns the username of the current user, such as `analyst` or `fgarcia`

Read files

The following Linux commands are helpful when reading files.

cat

Displays the content of a file

> `cat updates.txt`
> Displays the content of the `updates.txt` file

head

Displays just the beginning of a file, by default 10 lines

> `head updates.txt`
> Displays only the first 10 lines of the `updates.txt` file
>
> `head -n 5 updates.txt`
> Displays only the first five lines of the `updates.txt` file; the `-n` option allows users to specify the number of lines to return

less

Returns the content of a file one page at a time

> `less updates.txt`
> Returns the content of `updates.txt` one page at a time; the `less` command changes the terminal window to a display that allows users to easily move forward and backward through content

tail

Displays just the end of a file, by default 10 lines

> `tail updates.txt`
> Displays only the last 10 lines of the `updates.txt` file
>
> `tail -n 5 updates.txt`
> returns only the last five lines of the `updates.txt` file; the `-n` option allows users to specify the number of lines to return

Manage the file system

The following Linux commands are helpful when managing the file system.

cp

Copies a file or directory into a new location; the file will not be removed from the previous location

> `cp permissions.txt /home/analyst/logs`
> Copies the `permissions.txt` file from the user's current working directory to the `logs` directory

mkdir

Creates a new directory

> `mkdir network`
> Creates a new directory named `network` in the user's current working directory
>
> `mkdir /home/analyst/logs network`
> Creates a new directory named `network` in the `logs` directory; the full path is required when `logs` is not a subdirectory of the current directory

mv

Moves a file or directory to a new location; the file is also removed from the previous location

> `mv permissions.txt /home/analyst/logs`
> Moves the `permissions.txt` file from the user's current working directory to the `logs` directory
>
> `mv permissions.txt perm.txt`
> Moves the `permissions.txt` file from the user's current working directory to the new file name `perm.txt` in the user's current working directory; this results in renaming the `permissions.txt` file as `perm.txt`

nano

Opens or creates a file in the nano command-line file editor

> `nano permissions.txt`
> Opens an existing `permissions.txt` file in the nano file editor, or creates the `permissions.txt` file in the nano file editor if it doesn't already exist in the current working directory

rm

Removes, or deletes, a file

`rm permissions.txt`
removes the `permissions.txt` file from the user's current working directory

`rm home/analyst/reports/permissions.txt`
Removes the `permissions.txt` file from from the `reports` directory; the full path is required if the user's current working directory is not `reports`

rmdir

Removes, or deletes, a directory; only removes directories if they are empty

`rmdir network`
Removes the empty `network` subdirectory of the user's current working directory from the file system

`rmdir /home/analyst/logs/network`
Removes the empty `network` directory from the file system; the full path is required when `network` is not a subdirectory of the current directory

touch

Creates a new file

`touch permissions.txt`
Creates a new file named `permissions.txt` in the user's current working directory

`touch /home/analyst/reports/permissions.txt`
Creates a new file named `permissions.txt` in the `reports` directory; the full path is required if the user wants to create `permissions.txt` in any directory other than the current working directory

Filter content

The following Linux commands are helpful when filtering content.

find

Searches for directories and files that meet specified criteria

`find /home/analyst/projects`
Searches for all files starting at the `projects` directory

`find /home/analyst/projects -name "*log*"`
Searches for all files in the `projects` directory that contain the word `log` in the file name; the `-name` option searches for a specified string and is case-sensitive; the `*` wildcard represents zero or more unknown characters

`find /home/analyst/projects -iname "*log*"`

Searches for all files in the projects directory that contain the word log in the file name; the -iname option searches for a specified string and is not case-sensitive; the * wildcard represents zero or more unknown characters

find /home/analyst/projects -mtime -3
Searches for all files in the projects directory that have been modified within the past three days; the -mtime option bases its search for files or directories that were modified on days

find /home/analyst/projects -mmin -15
Searches for all files in the projects directory that have been modified within the past 15 minutes; the -mmin option bases its search for files or directories that were modified on minutes

grep

Searches a specified file and returns all lines in the file containing a specified string

grep OS updates.txt
Searches the updates.txt file and returns all lines containing the string OS

| (piping)

Sends the standard output of one command as standard input to another command for further processing; accessed using the pipe character (|)

ls /home/analyst/reports | grep users
Redirects the standard output of ls /home/analyst/reports to be standard input for the grep users command, meaning that grep users identifies files and subdirectories in the /home/analyst/reports directory that contain the string users within their file name

Manage users and their permissions

The following Linux commands are helpful when managing user permissions. (Also review the subentries for ls -l and ls -la in the ls entry of the **Navigate the file system** section.)

chmod

Changes permissions on files and directories

chmod u+rwx,g+rwx,o+rwx login_sessions.txt
Changes user (u), group (g), and other (o) permissions to add (+) read (r), write (w), and execute (x) permissions for the login_sessions.txt file

chmod g-rw bonuses.txt
Changes the group (g) permissions to remove (-) read (r) and write (w) permissions for the bonuses.txt file
chmod u=r,g=r,o=r login_sessions.txt

Changes user (u), group (g), and other (o) permissions to assign (=) read (r) permissions for the login_sessions.txt file

chown

Changes ownership of a file or directory; used with sudo

sudo chown fgarcia access.txt
Changes the user owner of the access.txt file to fgarcia

sudo chown :security access.txt
Changes the group owner of access.txt to security; a colon (:) must be entered before the group name

groupdel

Deletes a group from the system; used with sudo

sudo groupdel accounting
Deletes accounting as a group

sudo

Temporarily grants elevated permissions to specific users; users must be in a sudoers file to use have access to sudo

sudo useradd fgarcia
Grants elevated permissions to the user running this command and so that this user can use the useradd command to add fgarcia as a new user to the system

useradd

Adds a user to the system; used with sudo

sudo useradd fgarcia
Adds fgarcia as a new user to the system

sudo useradd -g security fgarcia
Adds fgarcia as a new user and uses the -g option to set their primary group as security

sudo useradd -G finance,admin fgarcia
Adds fgarcia as a new user and uses the -G option to add them to the supplemental groups of finance and admin

userdel

Deletes a user from the system; used with sudo

sudo userdel fgarcia
Deletes fgarcia as a user

`sudo userdel -r fgarcia`
Deletes `fgarcia` as a user and deletes all files in their home directory

usermod

Modifies existing user accounts; used with `sudo`

`sudo usermod -g executive fgarcia`
Uses the `-g` option to change the existing `fgarcia` user's primary group to the `executive` group

`sudo usermod -G accounting fgarcia`
Uses the `-G` option to replace any supplemental groups the the existing `fgarcia` user is in with the supplemental `accounting` group; removes all other supplemental groups `fgarcia` is in

`sudo usermod -a -G marketing fgarcia`
Uses the `-a -G` options to add the existing `fgarcia` user to the supplemental `marketing` group; does not remove `fgarcia` from other supplemental groups

`sudo usermod -d /home/garcia_f fgarcia`
Uses the `-d` option to change the existing `fgarcia` user's home directory to `/home/garcia_f`

`sudo usermod -L fgarcia`
Uses the `-L` option to lock the existing `fgarcia` user's account so they cannot log in

`sudo usermod -l garcia_f fgarcia`
Uses the `-l` option to change the existing `fgarcia` user's login name to `garcia_f`

Get help in Linux

The following Linux commands are helpful when getting help in Linux.

apropos

Searches the manual page descriptions for a specified string

`apropos password`
Returns the manual pages of commands that contain the keyword `password`

`apropos -a graph editor`
Returns the manual pages of commands that contain both the keywords `graph` and `editor`; the `-a` option specifies to only return commands that contain all specified strings

man

Displays information on other commands and how they work; the output is called a “man page,” which is short for "manual page"

> `man chown`
> Displays detailed information about `chown` and how it works

whatis

Displays a description of a command on a single line

> `whatis nano`
> Displays the description of `nano` on a single line

Module 4 - Databases and SQL

Introduction to SQL and Databases

We can define a database as an organized collection of information or data. Databases are often compared to spreadsheets. Some of you may have used Google Sheets or another common spreadsheet program in the past. While these programs are convenient ways to store data, spreadsheets are often designed for a single user or a small team to store less data. In contrast, databases can be accessed by multiple people simultaneously and can store massive amounts of data. Databases can also perform complex tasks while accessing data. As a security analyst, you'll often need to access databases containing useful information. For example, these could be databases containing information on login attempts, software and updates, or machines and their owners.

Now that we know how important databases are for us, let's talk about how they're organized and how we can interact with them. Using databases allows us to store large amounts of data while keeping it quick and easy to access. There are lots of different ways we can structure a database, but in this course, we'll be working with relational databases. A relational database is a structured database containing tables that are related to each other.

Let's learn more about what makes a relational database. We'll start by examining an individual table in a larger database of organizational information. Each table contains fields of information. For example, in this table on employees, these would include fields like employee_id, device_id, and username.

These are the columns of the tables. In addition, tables contain rows also called records. Rows are filled with specific data related to the columns in the table. For example, our first row is a record for an employee whose id is 1,000 and who works in the marketing department.

Relational databases often have multiple tables. Consider an example where we have two tables from a larger database, one with employees of the company and another with machines given to those employees. We can connect two tables if they share a common column. In this example, we establish a relationship between them with a common employee_id column. The columns that relate two tables to each other are called keys. There are two types of keys. The first is called a primary key. The primary key refers to a column where every row has a unique entry. The primary key must not have any duplicate values, or any null or empty values. The primary key allows us to uniquely identify every row in our table. For the table of employees, employee_id is a primary key. Every employee_id is unique and there are no employee_ids that are duplicate or empty.

The second type of key is a foreign key. The foreign key is a column in a table that is a primary key in another table. Foreign keys, unlike primary keys, can have empty values and duplicates. The foreign key allows us to connect two tables together. In our example, we can look at the employee_id column in the machines table. We previously identified this as a primary key in the employees table, so we can use this to connect every machine to their corresponding employee.

Query databases with SQL

SQL is a programming language used to create, interact with, and request information from a database.

Before learning more about SQL, we need to define what query means. A query is a request for data from a database table or a combination of tables. Nearly all relational databases rely on some version of SQL to query data. The different versions of SQL only have slight differences in their structure, like where to place quotation marks. Whatever variety of SQL you use, you'll find it to be a very important tool in your work as a security analyst.

First, let's discuss how SQL can help you retrieve logs. A log is a record of events that occur within an organization's systems. As a security analyst, you might be tasked with reviewing logs for various reasons. For example, some logs might contain details on machines used in a company, and as an analyst,

SQL is also a very common language used for basic data analytics, another set of skills that will set you apart as a security analyst. As a security analyst, you can use SQL's filtering to find data to support security-related decisions and analyze when things may go wrong. For instance, you can identify what machines haven't received the latest patch. This is important because patches are updates that help secure against attacks. As another example, you can use SQL to determine the best time to update a machine based on when it's least used.

SQL filtering versus Linux filtering

Accessing SQL

There are many interfaces for accessing SQL and many different versions of SQL. One way to access SQL is through the Linux command line.

To access SQL from Linux, you need to type in a command for the version of SQL that you want to use. For example, if you want to access SQLite, you can enter the command sqlite3 in the command line.

After this, any commands typed in the command line will be directed to SQL instead of Linux commands.

Differences between Linux and SQL filtering

Although both Linux and SQL allow you to filter through data, there are some differences that affect which one you should choose.

Purpose

Linux filters data in the context of files and directories on a computer system. It’s used for tasks like searching for specific files, manipulating file permissions, or managing processes.

SQL is used to filter data within a database management system. It’s used for querying and manipulating data stored in tables and retrieving specific information based on defined criteria.

Syntax

Linux uses various commands and command-line options specific to each filtering tool. Syntax varies depending on the tool and purpose. Some examples of Linux commands are find, sed, cut, e grep
SQL uses the Structured Query Language (SQL), a standardized language with specific keywords and clauses for filtering data across different SQL databases. Some examples of SQL keywords and clauses are WHERE, SELECT, JOIN

Structure

SQL offers a lot more structure than Linux, which is more free-form and not as tidy.
For example, if you wanted to access a log of employee log-in attempts, SQL would have each record separated into columns. Linux would print the data as a line of text without this organization. As a result, selecting a specific column to analyze would be easier and more efficient in SQL.
In terms of structure, SQL provides results that are more easily readable and that can be adjusted more quickly than when using Linux.

Joining tables

Some security-related decisions require information from different tables. SQL allows the analyst to join multiple tables together when returning data. Linux doesn't have that same functionality; it doesn't allow data to be connected to other information on your computer. This is more restrictive for an analyst going through security logs.

Best uses

As a security analyst, it's important to understand when you can use which tool. Although SQL has a more organized structure and allows you to join tables, this doesn't mean that there aren't situations that would require you to filter data in Linux.
A lot of data used in cybersecurity will be stored in a database format that works with SQL. However, other logs might be in a format that is not compatible with SQL. For instance, if the data is stored in a text file, you cannot search through it with SQL. In those cases, it is useful to know how to filter in Linux.

SQL queries

Basic queries

Let's say we have access to the employees table. The employees table has five columns. Two of them, employee_id and device_id, contain the information that we need. We'll write a query to this table that returns only those two columns from the table.

The two SQL keywords we need for basic SQL queries are SELECT and FROM. SELECT indicates which columns to return. FROM indicates which table to query. The use of these keywords in SQL is very similar to how we would use these words in everyday language. For example, we can ask a friend to select apples and bananas from the big box when going out to buy fruit. This is already very similar to SQL.

So let's go ahead and use SELECT and FROM in SQL to return the information we need on employees and the computers they use. We start off by typing in the SQL statement. After FROM, we've identified that the information will be pulled from the employees table. And after SELECT, employee_id and device_id indicate the two columns we want to return from this table. Notice how a comma separates the two columns that we want to return.

It's also worth mentioning a couple of key aspects related to the syntax of SQL here. Syntax refers to the rules that determine what is correctly structured in a computing language. In SQL, keywords are not case-sensitive, so you could also write select and from in lowercase, but we're placing them in capital letters because it makes the query easier to understand. Another aspect of this syntax is that semicolons are placed at the end of the statement.

And now, we'll run the query by pressing Enter. The output gives us the information we need to match employees to their computers. We just ran our very first SQL query!

Suppose you wanted to know what department the employee using the computer is from, or their username, or the office they work in. To do that, we can use SQL to make another statement that prints out all of the columns from the table. We can do this by placing an asterisk after SELECT. This is commonly referred to as select all. Now, let's run this query to the employees table in SQL. And now we have the full table in the output.

Basic SQL query

There are two essential keywords in any SQL query: **SELECT** and **FROM**. You will use these keywords every time you want to query a SQL database. Using them together helps SQL identify what data you need from a database and the table you are returning it from.
The video demonstrated this SQL query:

```
SELECT employee_id, device_id
FROM employees;
```

In readings and quizzes, this course uses a sample database called the **Chinook** database to run queries. The **Chinook** database includes data that might be created at a digital media company. A security analyst employed by this company might need to query this data. For example, the database contains eleven tables, including an **employees** table, a **customers** table, and an **invoices** table. These tables include data such as names and addresses.
As an example, you can run this query to return data from the **customers** table of the **Chinook** database:

```
SELECT customerid, city, country
FROM customers;
```

SELECT

The **SELECT** keyword indicates which columns to return. For example, you can return the **customerid** column from the **Chinook** database with

```
SELECT customerid
```

You can also select multiple columns by separating them with a comma. For example, if you want to return both the **customerid** and **city** columns, you should write **SELECT customerid, city**.
If you want to return all columns in a table, you can follow the **SELECT** keyword with an asterisk (*****). The first line in the query will be **SELECT ***.

Note: Although the tables you're querying in this course are relatively small, using **SELECT *** may not be advisable when working with large databases and tables; in those cases, the final output may be difficult to understand and might be slow to run.

FROM

The **SELECT** keyword always comes with the **FROM** keyword. **FROM** indicates which table to query. To use the **FROM** keyword, you should write it after the **SELECT** keyword, often on a new line, and follow it with the name of the table you're querying. If you want to return all columns from the **customers** table, you can write:

```
SELECT *
FROM customers;
```

When you want to end the query here, you put a semicolon (**;**) at the end to tell SQL that this is the entire query.

Note: Line breaks are not necessary in SQL queries, but are often used to make the query easier to understand. If you prefer, you can also write the previous query on one line as **SELECT * FROM customers;**

ORDER BY

Database tables are often very complicated, and this is where other SQL keywords come in handy. **ORDER BY** is an important keyword for organizing the data you extract from a table. **ORDER BY** sequences the records returned by a query based on a specified column or columns. This can be in either ascending or descending order.

Sorting in ascending order

To use the **ORDER BY** keyword, write it at the end of the query and specify a column to base the sort on. In this example, SQL will return the **customerid**, **city**, and **country** columns from the **customers** table, and the records will be sequenced by the **city** column:

```
SELECT customerid, city, country

FROM customers

ORDER BY city;
```

The **ORDER BY** keyword sorts the records based on the column specified after this keyword. By default, as shown in this example, the sequence will be in ascending order. This means

- if you choose a column containing numeric data, it sorts the output from the smallest to largest. For example, if sorting on **customerid**, the ID numbers are sorted from smallest to largest.
- if the column contains alphabetic characters, such as in the example with the **city** column, it orders the records from the beginning of the alphabet to the end.

Sorting in descending order

You can also use the **ORDER BY** with the **DESC** keyword to sort in descending order. The **DESC** keyword is short for "descending" and tells SQL to sort numbers from largest to smallest, or

alphabetically from Z to A. This can be done by following **ORDER BY** with the **DESC** keyword. For example, you can run this query to examine how the results differ when **DESC** is applied:

```
SELECT customerid, city, country

FROM customers

ORDER BY city DESC;
```

Now, cities at the end of the alphabet are listed first.

Sorting based on multiple columns

You can also choose multiple columns to order by. For example, you might first choose the **country** and then the **city** column. SQL then sorts the output by **country**, and for rows with the same **country**, it sorts them based on **city**. You can run this to explore how SQL displays this:

```
SELECT customerid, city, country

FROM customers

ORDER BY country, city;
```

Basic filters on SQL queries

Filtering is selecting data that match a certain condition. Think of filtering as a way of only choosing the data we want. Let's say we wanted to select apples from a fruit cart. Filtering allows us to specify what kind of apples we want to choose. When we go buy apples, we might explicitly say, "Choose only apples that are fresh." This removes apples that aren't fresh from the selection. This is a filter! As a security analyst, you might filter a log-in attempts table to find all attempts from a specific country. This could be done by applying a filter on the country column. For example, you could filter to just return records containing Canada.

Before we get started, we need to focus on an important part of the syntax of SQL. Let's learn about operators. An operator is a symbol or keyword that represents an operation. An example of an operator would be the equal to operator. For example, if we wanted to find all records that have USA in the country column, we use country = 'USA' To filter a query in SQL, we simply add an extra line to the SELECT and FROM statement we used before. This extra line will use a WHERE clause. In SQL, WHERE indicates the condition for a filter. After the keyword WHERE, the specific condition is listed using operators. So if we wanted to find all of the login attempts made in the United States, we would create this filter. In this particular condition, we're indicating to return all records that have a value in the country column that is equal to USA.

Let's try putting it all together in SQL. We're going to start with selecting all the columns from the log_in_attempts table. And then add the WHERE filter. Don't forget the semicolon! This tells us we finished the SQL statement. Now, let's run this query! Because of our filter, only the rows where the country of the log-in attempt was USA are returned.

In the previous example, the condition for our filter was based simply on returning records that are equal to a particular value. We can also make our conditions more complex by searching for a pattern instead of an exact word. For example, in the employees table, we have a column for office. We could search for records in this column that match a certain pattern. Perhaps we might want all offices in the East building. To search for a pattern, we used the percentage sign to act as a

wildcard for unspecified characters. If we ran a filter for 'East%', this would return all records that start with East -- for example, the offices East-120, East-290, and East-435.

When searching for patterns with the percentage sign, we cannot use the equals operator. Instead, we use another operator, LIKE. LIKE is an operator used with WHERE to search for a pattern in a column.

Since LIKE is an operator, similar to the equal sign, we use it instead of the equal sign. So, when our goal is to return all values in the office column that start with the word East, LIKE would appear in a WHERE clause.

Let's go back to the example in which we wanted to filter for log-in attempts made in the United States. Imagine that we realize that our database contains inconsistencies with how the United States is represented. Some entries use US while others use USA. Let's get into SQL and apply this new type of filter with LIKE. We're going to start with the same first two lines of code because we want to select all columns from the log-in attempts table. And we're going to add a filter with LIKE so that records will be returned if they contain a value in the country column beginning with the characters US. This includes both US and USA. Let's run this query to check if the output changes. This returns all the entries where the user location was in the United States. And now we can use the LIKE clause to filter columns based on a pattern!

How filtering helps

As a security analyst, you'll often be responsible for working with very large and complicated security logs. To find the information you need, you'll often need to use SQL to filter the logs. In a cybersecurity context, you might use filters to find the login attempts of a specific user or all login attempts made at the time of a security issue. As another example, you might filter to find the devices that are running a specific version of an application.

WHERE

To create a filter in SQL, you need to use the keyword **WHERE**. **WHERE** indicates the condition for a filter.
If you needed to email employees with a title of IT Staff, you might use a query like the one in the following example. You can run this example to examine what it returns:

```
SELECT firstname, lastname, title, email

FROM employees

WHERE title = 'IT Staff';
```

Rather than returning all records in the **employees** table, this **WHERE** clause instructs SQL to return only those that contain **'IT Staff'** in the **title** column. It uses the equals sign (**=**) operator to set this condition.
Note: You should place the semicolon (**;**) where the query ends. When you add a filter to a basic query, the semicolon is after the filter.

Filtering for patterns

You can also filter based on a pattern. For example, you can identify entries that start or end with a certain character or characters. Filtering for a pattern requires incorporating two more elements into your **WHERE** clause:

- a wildcard
- the **LIKE** operator

Wildcards

A **wildcard** is a special character that can be substituted with any other character. Two of the most useful wildcards are the percentage sign (**%**) and the underscore (**_**):

- The percentage sign substitutes for any number of other characters.
- The underscore symbol only substitutes for one other character.

These wildcards can be placed after a string, before a string, or in both locations depending on the pattern you're filtering for.
The following table includes these wildcards applied to the string **'a'** and examples of what each pattern would return.

Pattern	Results that could be returned
'a%'	**apple123, art, a**
'a_'	**as, an, a7**
'a__'	**ant, add, a1c**
'%a'	**pizza, Z6ra, a**
'_a'	**ma, 1a, Ha**
'%a%'	**Again, back, a**
'_a_'	**Car, ban, ea7**

LIKE

To apply wildcards to the filter, you need to use the **LIKE** operator instead of an equals sign (**=**).
LIKE is used with **WHERE** to search for a pattern in a column.
For instance, if you want to email employees with a title of either **'IT Staff'** or **'IT Manager'**, you can use **LIKE** operator combined with the **%** wildcard:

```
SELECT lastname, firstname, title, email

FROM employees

WHERE title LIKE 'IT%';
```

This query returns all records with values in the **title** column that start with the pattern of **'IT'**. This means both **'IT Staff'** and **'IT Manager'** are returned.

As another example, if you want to search through the invoices table to find all customers located in states with an abbreviation of **'NY'**, **'NV'**, **'NS'** or **'NT'**, you can use the **'N_'** pattern on the **state** column:

```
SELECT firstname,lastname, state, country

FROM customers

WHERE state LIKE 'N_';
```

This returns all the records with state abbreviations that follow this pattern.

More SQL filters

Let's explore the three common data types that you will find in databases: string, numeric, and date and time. String data is data consisting of an ordered sequence of characters. These characters could be numbers, letters, or symbols. For example, you'll encounter string data in user names, such as a user name: analyst10. Numeric data is data consisting of numbers, such as a count of log-in attempts. Unlike strings, mathematical operations can be used on numeric data, like multiplication or addition. Date and time data refers to data representing a date and/or time.

Previously, we applied filters using string data, but now let's work with numeric and date and time data. As a security analyst, you'll often need to query numbers and dates. For example, we could filter patch dates to find machines that need an update, or we could filter log-in attempts to return only those made in a certain period of time. We learned about operators in the last video, and we're going to use them again for numbers and dates.

Common operators for working with numeric or date and time data types include: equals, greater than, less than, not equal to, greater than or equal to, and less than or equal to. Let's say you want to find the log-in attempts made after 6 pm. Because this is past normal business hours, you want to look for suspicious patterns. You can identify these attempts by using the greater than operator in your filter. We'll start writing our query in SQL. We begin by indicating that we want to select all columns FROM the log_in_attempts table. Then we'll add our filter with WHERE.

Our condition indicates that the value in the time column must be greater than, or for dates and times, later than '18:00', which is how 6 pm is written in SQL. Let's run this and examine the output. Perfect! Now we have a list of log-in attempts made after 6 pm.

We can also filter for numbers and dates by using the BETWEEN operator. BETWEEN is an operator that filters for numbers or dates within a range. An example of this would be when looking for all patches installed within a certain range. Let's do this! Let's find all the patches installed between March 1st, 2021 and September 1st, 2021. In our query, we start with selecting all records FROM the machines table.

And we add the BETWEEN operator in the WHERE statement.

Let's break down the statement. First, after WHERE, we indicate which column to filter, in our case, OS_patch_date. Next, comes our operator BETWEEN. We then add the beginning of our range, type AND, then finish by adding the end of our range and a semicolon. Now, let's run this and explore the output. And now we have a list of all machines patched between those two dates!

Numbers, dates, and times in cybersecurity

Security analysts work with more than just **string data**, or data consisting of an ordered sequence of characters.
They also frequently work with **numeric data**, or data consisting of numbers. A few examples of numeric data that you might encounter in your work as a security analyst include:

- the number of login attempts
- the count of a specific type of log entry
- the volume of data being sent from a source
- the volume of data being sent to a destination

You'll also encounter **date and time data**, or data representing a date and/or time. As a first example, logs will generally timestamp every record. Other time and date data might include:

- login dates
- login times
- dates for patches
- the duration of a connection

Comparison operators

In SQL, filtering numeric and date and time data often involves operators. You can use the following operators in your filters to make sure you return only the rows you need:

operator	use
<	less than
>	greater than
=	equal to
<=	less than or equal to
>=	greater than or equal to
<>	not equal to

Note: You can also use **!=** as an alternative operator for not equal to.

Incorporating operators into filters

These comparison operators are used in the **WHERE** clause at the end of a query. The following query uses the **>** operator to filter the **birthdate** column. You can run this query to explore its output:

```
SELECT firstname, lastname, birthdate

FROM employees

WHERE birthdate > '1970-01-01';
```

This query returns the first and last names of employees born after, but not on, **'1970-01-01'** (or January 1, 1970). If you were to use the **>=** operator instead, the results would also include results on exactly **'1970-01-01'**.
In other words, the **>** operator is exclusive and the **>=** operator is inclusive. An **exclusive operator** is an operator that does not include the value of comparison. An **inclusive operator** is an operator that includes the value of comparison.

BETWEEN

Another operator used for numeric data as well as date and time data is the **BETWEEN** operator. **BETWEEN** filters for numbers or dates within a range. For example, if you want to find the first and last names of all employees hired between January 1, 2002 and January 1, 2003, you can use the **BETWEEN** operator as follows:

```
SELECT firstname, lastname, hiredate

FROM employees

WHERE hiredate BETWEEN '2002-01-01' AND '2003-01-01';
```

Note: The **BETWEEN** operator is inclusive. This means records with a **hiredate** of January 1, 2002 or January 1, 2003 are included in the results of the previous query.

Filters with AND, OR, and NOT

Vulnerabilities, for instance, might depend on more than one factor. For example, a security vulnerability might be related to machines using a specific email client on a specific operating system. So, to find the possible vulnerabilities, we need to find machines using both the email client and the operating system.

To make a query with multiple conditions that must be met, we use the AND operator between two separate conditions. AND is an operator that specifies that both conditions must be met simultaneously. Bringing this back to our fruit and vegetable analogy, this is the same as asking someone to select apples from the big box where the apples are large and fresh. This means our results won't include any small apples even if they're fresh, or any rotten apples even if they're large. They'll only include large fresh apples. The apples must meet both conditions.

Going back to our database, the machines table lists all operating systems and email clients. We want a list of machines running Operating System 1 and a list of machines using Email Client 1. We'll use the left and right circles in the Venn diagram to represent these groups. We need SQL to select the machines that have both OS 1 and Email Client 1. The filled-in area at the intersection of these circles represents this condition. Let's take this and implement it in SQL.

First, we're going to start by building the first lines of the query, telling SQL to SELECT* all columns FROM the machines table. Then, we'll add the WHERE clause.

Let's examine this more closely. First, we indicate the first condition that it must meet, that the operating system column has a value of '0S 1'

Then, we use AND to join this to another condition. And finally, we enter the other condition, in this case that the email client column should have a value of 'Email Client 1'

And this is how you use the AND operator in SQL! Let's run this to get the query results. Perfect! All the results match both our conditions!

Let's keep going and explore more ways to combine different conditions by working with the OR operator. The OR operator is an operator that specifies that either condition can be met. In

a Venn diagram, let's say each circle represents a condition. When they are joined with OR, SQL would select all rows that satisfy one of the conditions. And it's also ok if it meets both conditions.

Let's run another query and use the OR operator. Let's say that we wanted the filter to identify machines that have either OS 1 or OS 3 because both types need a patch. We'll type in these conditions.

Let's examine this more closely. After WHERE, our first condition indicates we want to filter, so that the query selects machines with 'OS 1' We use the OR operator because we also want to find records that match another condition. This additional condition is placed after OR and indicates to also select machines running 'OS 3' Executing the query, our results now include records that have a value of either OS 1 or OS 3 in the operating system column. Good job, we're running some complex queries.

The last operator we're going to go into is the NOT operator. NOT negates a condition. In a diagram, we can show this by selecting every entry that does not match our condition. The condition is represented by the circle. The filled-in portion outside the circle represents what gets returned. This is all data that does not match the condition. For example, when picking out fruit, you can be looking for any fruit that is not an apple. That is a lot more efficient than telling your friend you want a banana or an orange or a lime, and so on.

Suppose you wanted to update all of the devices in your company except for the ones using OS 3. Bringing this into SQL, we can write this query.

We place NOT after WHERE and before the condition of the filter. Executing these queries gives us the list of all the machines that aren't running OS 3, and now we know which machines to update.

Logical operators

AND, **OR**, and **NOT** allow you to filter your queries to return the specific information that will help you in your work as a security analyst. They are all considered logical operators.

AND

First, **AND** is used to filter on two conditions. **AND** specifies that both conditions must be met simultaneously.

As an example, a cybersecurity concern might affect only those customer accounts that meet both the condition of being handled by a support representative with an ID of 5 and the condition of being located in the USA. To find the names and emails of those specific customers, you should place the two conditions on either side of the **AND** operator in the **WHERE** clause:

```
SELECT firstname, lastname, email, country, supportrepid

FROM customers

WHERE supportrepid = 5 AND country = 'USA';
```

Running this query returns four rows of information about the customers. You can use this information to contact them about the security concern.

OR

The **OR** operator also connects two conditions, but **OR** specifies that either condition can be met. It returns results where the first condition, the second condition, or both are met.

For example, if you are responsible for finding all customers who are either in the USA or Canada so that you can communicate information about a security update, you can use an **OR** operator to find all the needed records. As the following query demonstrates, you should place the two conditions on either side of the **OR** operator in the **WHERE** clause:

```
SELECT firstname, lastname, email, country

FROM customers

WHERE country = 'Canada' OR country = 'USA';
```

The query returns all customers in either the US or Canada.

Note: Even if both conditions are based on the same column, you need to write out both full conditions. For instance, the query in the previous example contains the filter **WHERE country = 'Canada' OR country = 'USA'**.

NOT

Unlike the previous two operators, the **NOT** operator only works on a single condition, and not on multiple ones. The **NOT** operator negates a condition. This means that SQL returns all records that don't match the condition specified in the query.

For example, if a cybersecurity issue doesn't affect customers in the USA but might affect those in other countries, you can return all customers who are not in the USA. This would be more efficient than creating individual conditions for all of the other countries. To use the **NOT** operator for this task, write the following query and place **NOT** directly after **WHERE**:

```
SELECT firstname, lastname, email, country

FROM customers

WHERE NOT country = 'USA';
```

SQL returns every entry where the customers are not from the USA.

Pro tip: Another way of finding values that are not equal to a certain value is by using the **<>** operator or the **!=** operator. For example, **WHERE country <> 'USA'** and **WHERE country != 'USA'** are the same filters as **WHERE NOT country = 'USA'**.

Combining logical operators

Logical operators can be combined in filters. For example, if you know that both the USA and Canada are not affected by a cybersecurity issue, you can combine operators to return customers in all countries besides these two. In the following query, **NOT** is placed before the first condition, it's joined to a second condition with **AND**, and then **NOT** is also placed before that second condition. You can run it to explore what it returns:

```
SELECT firstname, lastname, email, country

FROM customers
```

```
WHERE NOT country = 'Canada' AND NOT country = 'USA';
```

SQL joins

Join tables in SQL

First, let's start talking about the syntax of joins. Since we're working with two tables now, we need a way to tell SQL what table we're picking columns from. In our example database, we have an employee_id column in both the employees table and the machines table. In SQL statements that contain two columns, SQL needs to know which column we're referring to. The way to resolve this is by writing the name of the table first, then a period, and then the name of a column. So, we would have employees followed by a period, followed by the column name. This is the employee_id column for the employees table. Similarly, this is the employee_id column for the machines table. Now that we understand this syntax, let's apply it to a join!

Imagine that we want to get a deeper understanding of the employees accessing the machines in our company. By joining the employees and the machines tables, we can do this! We first need to identify the shared column that we'll use to connect the two tables. In this case, we'll use a primary key and one table to connect to another table where it's a foreign key. The primary key of the employees table is employee_id, which is a foreign key in the machines table. employee_id is a primary key in the employees table because it has a unique value for every row in the employees table, and no empty values. We don't have a guarantee that the employee_id column in the machines table follows the same criteria since it's a foreign key and not a primary key.

Next, we'll use a type of join called an INNER JOIN. An INNER JOIN returns rows matching on a specified column that exists in more than one table. Tables usually contain many more rows, but to further explain what we mean by INNER JOIN, let's focus on just four rows from the employees table and four rows from the machines table. We'll also look at just a few columns of each table for this example. Let's say we choose employee_id in both tables to perform an INNER JOIN. Let's look at the two rows where there is a match. Both tables have 1188 and 1189 in their respective employee_id columns, so they are considered a match. The results of the join is the two rows that have 1188 and 1189 and all columns from both tables.

Before we move on to the queries, we have to talk about the NULL values in the tables. In SQL, NULL represents a missing value due to any reason. In this case, this might be machines that are not assigned to any employee. Now, let's bring this into SQL and do an INNER JOIN on the full tables. Let's imagine we want to join these tables in order to get a list of users and their office location that also shows what operating system they use on their machines. employee_id is a common column between these tables, and we can use this to join them. But we won't need to show this column in the results. First, let's start with a basic query that indicates we want to select the username, office, and operating_system columns. We want employees to be our first or left table, so we'll use that in our FROM statement. Now, we write the part of the query that tells SQL to join the machines table with the employees table.

Let's break down this query. INNER JOIN tells SQL to perform the INNER JOIN. Then, we name the second table we want to combine with the first. This is called the right table. In this case, we want to join machines with the employees table that was already identified after FROM. Lastly, we tell SQL what column to base the join on. In our case, we're using the employee_id column. Since we're using two tables, we have to identify the table and follow that with the column name. So, we have employees.employee_id. And machines.employee_id.

Types of joins

There are three types of outer joins: LEFT JOIN, RIGHT JOIN, and FULL OUTER JOIN. Similar to inner joins, outer joins combine two tables together; however, they don't necessarily need a match between columns to return a row. Which rows are returned depends on the type of join.

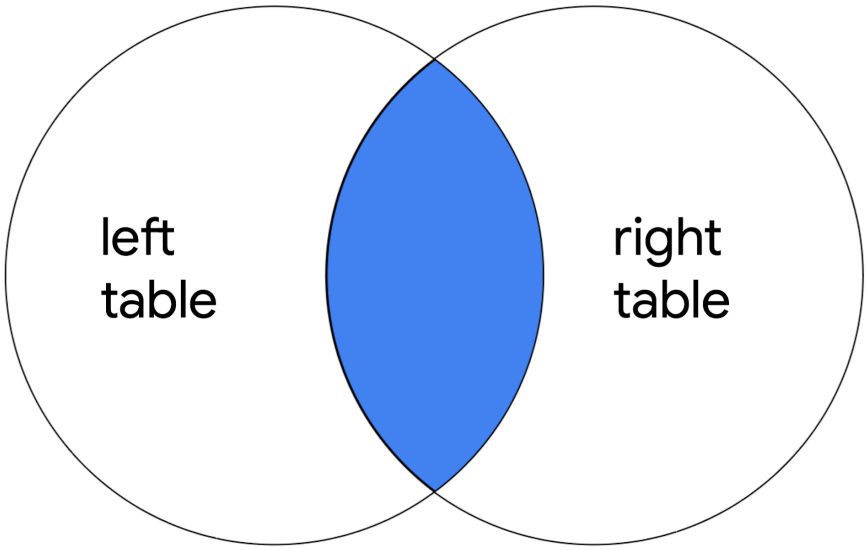
LEFT JOIN returns all of the records of the first table, but only returns rows of the second table that match on a specified column. Like we did in the previous video, let's examine this type of join by looking at just four rows of two tables with a small number of columns. Employees is the left table, or the first table, and machines is the right table, or the second table. Let's join on employee_id. There's a matching value in this column for two of the four records. When we execute the join, SQL returns these rows with the matching value, all other rows from the left table, and all columns from both tables. Records from the employees table that didn't match but were returned through the LEFT JOIN contain NULL values in columns that came from the machines table. Next, let's talk about right joins.

RIGHT JOIN returns all of the records of the second table but only returns rows from the first table that match on a specified column. With a RIGHT JOIN on the previous example, the full result returns matching rows from both, all the rows from the second table, and all the columns in both tables. For the values that don't exist in either table, we are left with a NULL value. Last, we'll discuss full outer joins.

FULL OUTER JOIN returns all records from both tables. Using our same example, a FULL OUTER JOIN returns all columns from all tables. If a row doesn't have a value for a particular column, it returns NULL. For example, the machines table do not have any rows with employee_id 1190, so the values for that row and the columns that came from the machines table is NULL. To implement left joins, right joins, and full outer joins in SQL, you use the same syntax structure as the INNER JOIN but use these keywords: LEFT JOIN, RIGHT JOIN, and FULL OUTER JOIN

nner joins

The first type of join that you might perform is an inner join. **INNER JOIN** returns rows matching on a specified column that exists in more than one table.



It only returns the rows where there is a match, but like other types of joins, it returns all specified columns from all joined tables. For example, if the query joins two tables with **SELECT ***, all columns in both of the tables are returned.

Note: If a column exists in both of the tables, it is returned twice when **SELECT *** is used.

The syntax of an inner join

To write a query using **INNER JOIN**, you can use the following syntax:

```
SELECT *
FROM employees
INNER JOIN machines ON employees.device_id = machines.device_id;
```

You must specify the two tables to join by including the first or left table after **FROM** and the second or right table after **INNER JOIN**.

After the name of the right table, use the **ON** keyword and the **=** operator to indicate the column you are joining the tables on. It's important that you specify both the table and column names in this portion of the join by placing a period (**.**) between the table and the column.

In addition to selecting all columns, you can select only certain columns. For example, if you only want the join to return the **username**, **operating_system** and **device_id** columns, you can write this query:

```
SELECT username, operating_system, employees.device_id
FROM  employees
INNER JOIN machines ON employees.device_id = machines.device_id;
```

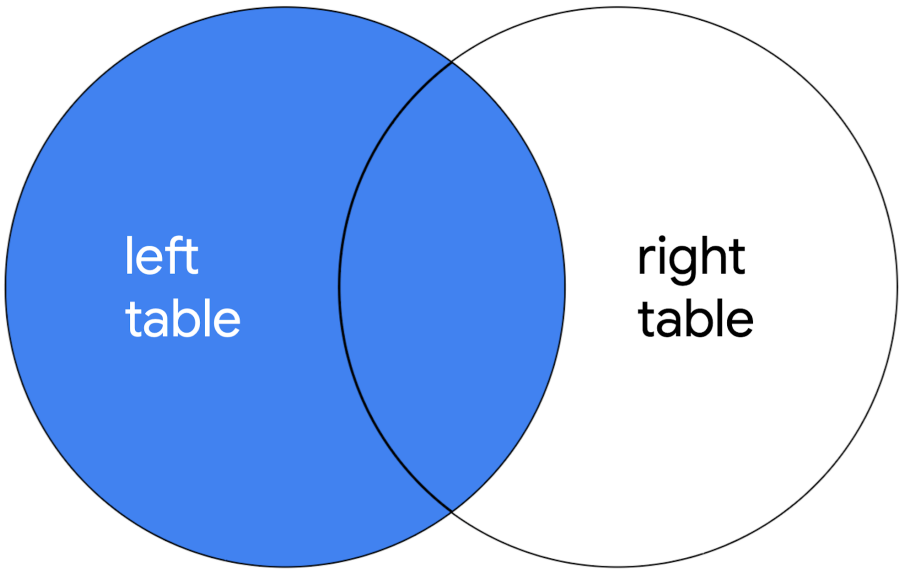
Note: In the example query, **username** and **operating_system** only appear in one of the two tables, so they are written with just the column name. On the other hand, because **device_id** appears in both tables, it's necessary to indicate which one to return by specifying both the table and column name (**employees.device_id**).

Outer joins

Outer joins expand what is returned from a join. Each type of outer join returns all rows from either one table or both tables.

Left joins

When joining two tables, **LEFT JOIN** returns all the records of the first table, but only returns rows of the second table that match on a specified column.



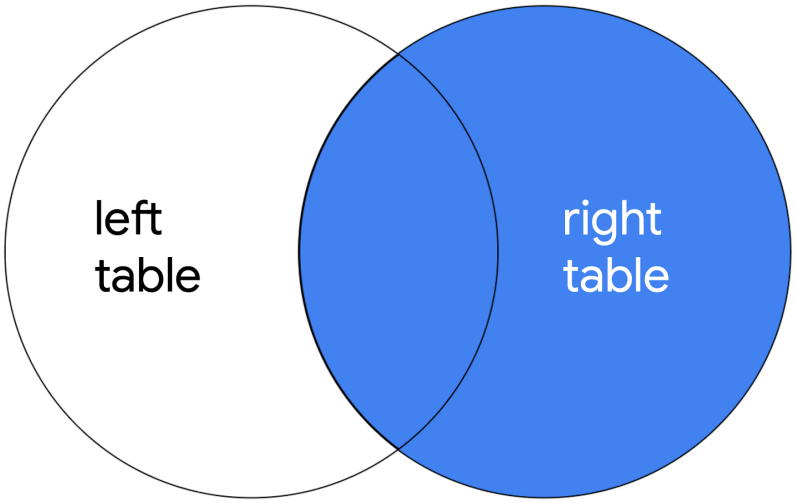
The syntax for using **LEFT JOIN** is demonstrated in the following query:

```
SELECT *
FROM employees
LEFT JOIN machines ON employees.device_id = machines.device_id;
```

As with all joins, you should specify the first or left table as the table that comes after **FROM** and the second or right table as the table that comes after **LEFT JOIN**. In the example query, because **employees** is the left table, all of its records are returned. Only records that match on the **device_id** column are returned from the right table, **machines**.

Right joins

When joining two tables, **RIGHT JOIN** returns all of the records of the second table, but only returns rows from the first table that match on a specified column.



The following query demonstrates the syntax for **RIGHT JOIN**:

```
SELECT *
FROM employees
RIGHT JOIN machines ON employees.device_id = machines.device_id;
```

RIGHT JOIN has the same syntax as **LEFT JOIN**, with the only difference being the keyword **RIGHT JOIN** instructs SQL to produce different output. The query returns all records from **machines**, which is the second or right table. Only matching records are returned from **employees**, which is the first or left table.

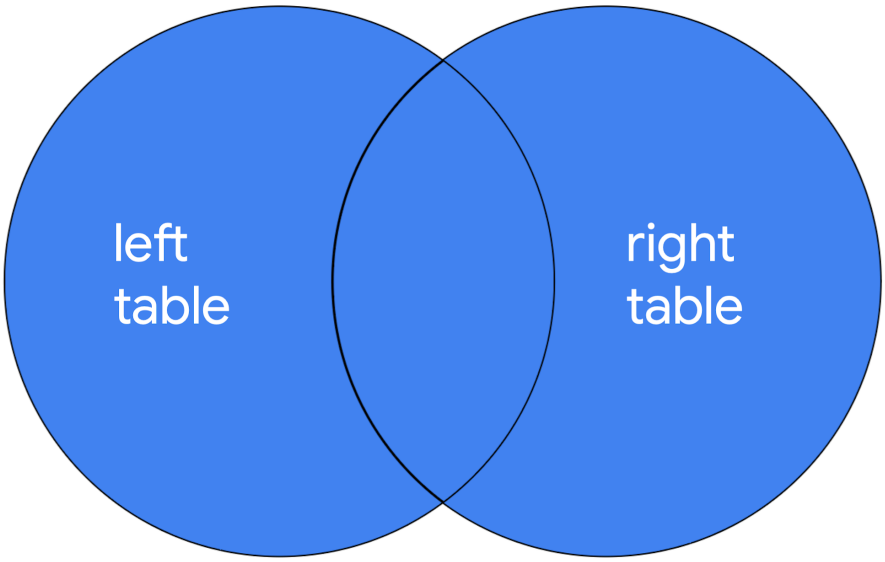
Note: You can use **LEFT JOIN** and **RIGHT JOIN** and return the exact same results if you use the tables in reverse order. The following **RIGHT JOIN** query returns the exact same result as the **LEFT JOIN** query demonstrated in the previous section:

```
SELECT *
FROM machines
RIGHT JOIN employees ON employees.device_id = machines.device_id;
```

All that you have to do is switch the order of the tables that appear before and after the keyword used for the join, and you will have swapped the left and right tables.

Full outer joins

FULL OUTER JOIN returns all records from both tables. You can think of it as a way of completely merging two tables.



You can review the syntax for using **FULL OUTER JOIN** in the following query:

```
SELECT *
FROM employees
FULL OUTER JOIN machines ON employees.device_id = machines.device_id;
```

The results of a **FULL OUTER JOIN** query include all records from both tables. Similar to **INNER JOIN**, the order of tables does not change the results of the query.

Aggregate functions

In SQL, **aggregate functions** are functions that perform a calculation over multiple data points and return the result of the calculation. The actual data is not returned.

There are various aggregate functions that perform different calculations:

- **COUNT** returns a single number that represents the number of rows returned from your query.
- **AVG** returns a single number that represents the average of the numerical data in a column.
- **SUM** returns a single number that represents the sum of the numerical data in a column.

Aggregate function syntax

To use an aggregate function, place the keyword for it after the **SELECT** keyword, and then in parentheses, indicate the column you want to perform the calculation on.

For example, when working with the **customers** table, you can use aggregate functions to summarize important information about the table. If you want to find out how many customers there are in total, you can use the **COUNT** function on any column, and SQL will return the total number of records, excluding **NULL** values. You can run this query and explore its output:

```
SELECT COUNT(firstname)

FROM customers;
```

The result is a table with one column titled **COUNT(firstname)** and one row that indicates the count.

If you want to find the number of customers from a specific country, you can add a filter to your query:

```
SELECT COUNT(firstname)
```

```
FROM customers

WHERE country = 'USA';
```

With this filter, the count is lower because it only includes the records where the **country** column contains a value of **'USA'**.
There are a lot of other aggregate functions in SQL. The syntax of placing them after **SELECT** is exactly the same as the **COUNT** function.

Continuing to learn SQL

SQL is a widely used querying language, with many more keywords and applications. You can continue to learn more about aggregate functions and other aspects of using SQL on your own. Most importantly, approach new tasks with curiosity and a willingness to find new ways to apply SQL to your work as a security analyst. Identify the data results that you need and try to use SQL to obtain these results.
Fortunately, SQL is one of the most important tools for working with databases and analyzing data, so you'll find a lot of support in trying to learn SQL online. First, try searching for the concepts you've already learned and practiced to find resources that have accurate easy-to-follow explanations. When you identify these resources, you can use them to extend your knowledge. Continuing your practical experience with SQL is also important. You can also search for new databases that allow you to perform SQL queries using what you've learned.

Query a database

The `SELECT`, `FROM`, and `ORDER BY` keywords are used when retrieving information from a database.

FROM

Indicates which table to query; required to perform a query

> `FROM employees`
> Indicates to query the `employees` table

ORDER BY

Sequences the records returned by a query based on a specified column or columns

> `ORDER BY department`
> Sorts the records in ascending order by the `department` column; `ORDER BY department ASC` also sorts the records in ascending order by the `department` column
>
> `ORDER BY city DESC`

Sorts the records in descending order by the `city` column

`ORDER BY country, city`

Sorts the records in ascending order by multiple columns; first sorts the output by `country`, and for records with the same `country`, sorts them based on `city`

SELECT

Indicates which columns to return; required to perform a query

`SELECT employee_id`

Returns the `employee_id` column

`SELECT *`

Returns all columns in a table

Apply filters to SQL queries

`WHERE` and the other SQL keywords and characters that follow are used when applying filters to SQL queries.

AND

Specifies that both conditions must be met simultaneously in a filter that contains two conditions

`WHERE region = 5 AND country = 'USA'`

Returns all records with a value in the `region` column of `5` and a value in the `country` column of `'USA'`

BETWEEN

Filters for numbers or dates within a range; `BETWEEN` is followed by the first value to include in the range, the `AND` operator, and the last value to include in the range

`WHERE hiredate BETWEEN '2002-01-01' AND '2003-01-01'`

Returns all records with a value in the `hiredate` column that is between `'2002-01-01'` and `'2003-01-01'`

= (equal to)

Used in filters to return only the records that contain a value in a specified column that is equal to a particular value

`WHERE birthdate = '1980-05-15'`

Returns all records with a value in the `birthdate` column that equals `'1980-05-15'`

> (greater than)

Used in filters to return only the records that contain a value in a specified column that is greater than a particular value

> `WHERE birthdate > '1970-01-01'`
> Returns all records with a value in the `birthdate` column that is greater than `'1970-01-01'`

>= (greater than or equal to)

Used in filters to return only the records that contain a value in a specified column that is greater than or equal to a particular value

> `WHERE birthdate >= '1965-06-30'`
> Returns all records with a value in the `birthdate` column that is greater than or equal to `'1965-06-30'`

< (less than)

Used in filters to return only the records that contain a value in a specified column that is less than a particular value

> `WHERE date < '2023-01-31'`
> Returns all records with a value in the `date` column that is less than `'2023-01-31'`

<= (less than or equal to)

Used in filters to return only the records that contain a value in a specified column that is less than or equal to a particular value

> `WHERE date <= '2020-12-31'`
> Returns all records with a value in the `date` column that is less than or equal to `'2020-12-31'`

LIKE

Used with `WHERE` to search for a pattern in a column

> `WHERE title LIKE 'IT%'`
> Returns all records with a value in the `title` column that matches the pattern of `'IT%'`

> `WHERE state LIKE 'N_'`
> Returns all records with a value in the `state` column that matches the pattern of `'N_'`

NOT

Negates a condition

> `WHERE NOT country = 'Mexico'`
> Returns all records with a value in the `country` column that is not `'Mexico'`

<> (not equal to)

Used in filters to return only the records that contain a value in a specified column that is not equal to a particular value; `!=` also used as an operator for not equal to

> `WHERE date <> '2023-02-28'`
> Returns all records with a value in the `date` column that is not equal to `'2023-02-28'`

!= (not equal to)

Used in filters to return only the records that contain a value in a specified column that is not equal to a particular value; `<>` also used as an operator for not equal to

> `WHERE date != '2023-05-14'`
> Returns all records with a value in the `date` column that is not equal to `'2023-05-14'`

OR

Specifies that either condition can be met in a filter that contains two conditions

> `WHERE country = 'Canada' OR country = 'USA'`
> Returns all records with a value in the `country` column of either `'Canada'` or `'USA'`

% (percentage sign)

Substitutes for any number of other characters; used as a wildcard in a pattern that follows `LIKE`

> `'a%'`
> Represents a pattern consisting of the letter `'a'` followed by zero or more characters
>
> `'%a'`
> Represents a pattern consisting of zero or more characters followed by the letter `'a'`
>
> `'%a%'`

Represents a pattern consisting of the letter 'a' surrounded by zero or more characters on each side

_ (underscore)

Substitutes for one other character; used as a wildcard in a pattern that follows LIKE

'a_'
Represents a pattern consisting of the letter 'a' followed by one character

'a__'
Represents a pattern consisting of the letter 'a' followed by two characters

'_a'
Represents a pattern consisting of one character followed by the letter 'a'

'_a_'
Represents a pattern consisting of the letter 'a' surrounded by one character on each side

WHERE

Indicates the condition for a filter; must be used to begin a filter

```
WHERE title = 'IT Staff'
```
Returns all records that contain 'IT Staff' in the title column; WHERE is placed before the condition of title = 'IT Staff' to create the filter

Join tables

The following SQL keywords are used to join tables.

FULL OUTER JOIN

Returns all records from both tables; the column used to join the tables is specified following FULL OUTER JOIN with syntax that includes ON and equal to (=)

```
SELECT *
FROM employees
FULL OUTER JOIN machines ON employees.device_id = machines.device_id;
```
Returns all records from the employees table and machines table; uses the device_id column to join the two tables

INNER JOIN

Returns records matching on a specified column that exists in more than one table; the column used to join the tables is specified following `INNER JOIN` with syntax that includes `ON` and equal to (`=`)

```
SELECT *
FROM employees
INNER JOIN machines ON employees.device_id = machines.device_id;
```

Returns all records that have a value in the `device_id` column in the `employees` table that matches a value in the `device_id` column in the `machines` table

LEFT JOIN

Returns all the records of the first table, but only returns records of the second table that match on a specified column; the first (or left) table appears directly after the keyword `FROM`; the column used to join the tables is specified following `LEFT JOIN` with syntax that includes `ON` and equal to (`=`)

```
SELECT *
FROM employees
LEFT JOIN machines ON employees.device_id = machines.device_id;
```

Returns all records from the `employees` table but only the records from the `machines` table that have a value in the `device_id` column that matches a value in the `device_id` column in the `employees` table

RIGHT JOIN

Returns all of the records of the second table, but only returns records from the first table that match on a specified column; the second (or right) table appears directly after the `RIGHT JOIN` keyword; the column used to join the tables is specified following `RIGHT JOIN` with syntax that includes `ON` and equal to (`=`)

```
SELECT *
FROM employees
RIGHT JOIN machines ON employees.device_id = machines.device_id;
```

Returns all records from the `machines` table but only the records from the `employees` table that have a value in the `device_id` column that matches a value in the `device_id` column in the `machines` table

Perform calculations

The following SQL keywords are aggregate functions and are helpful when performing calculations.

AVG

Returns a single number that represents the average of the numerical data in a column; placed after `SELECT`

> `SELECT AVG(height)`
> Returns the average height from all records that have a value in the `height` column

COUNT

Returns a single number that represents the number of records returned from a query; placed after `SELECT`

> `SELECT COUNT(firstname)`
> Returns the number of records that have a value in the `firstname` column

SUM

Returns a single number that represents the sum of the numerical data in a column; placed after `SELECT`

> `SELECT SUM(cost)`
> Returns the sum of costs from all records that have a value in the `cost` column