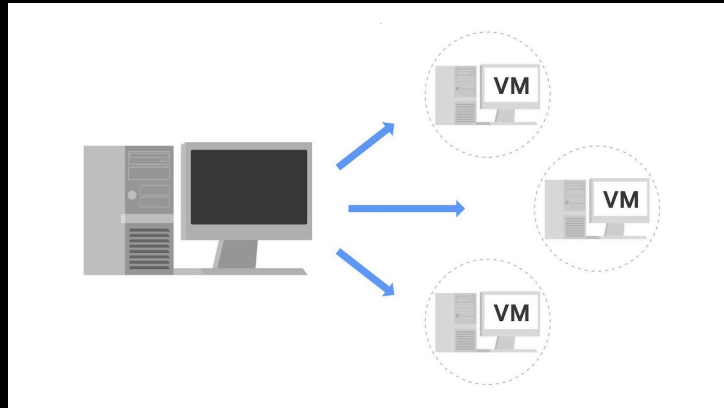


# Tools of the Trade: Linux and SQL

## Module 1: Intro to OSs

### Virtualization Technology



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.

## **The Command Line in Use**

### **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.

## **Module 2: The Linux Operating System**

### **Introduction to Linux**

#### The Origins of Linux

- Linux was developed from two separate projects in the early 1990s: Linus Torvalds created the Linux kernel to improve the UNIX operating system, and Richard Stallman started GNU, an open-source operating system also based on UNIX.
- The combination of Torvalds' kernel and Stallman's GNU project led to the creation of what is now known as Linux.

#### Key Characteristics of Linux

- Linux is an open-source operating system, meaning its source code is freely accessible, allowing anyone to use, share, and modify it under the GNU Public License.
- This open-source philosophy has fostered a large community of developers who collaborate to advance computing and create over 600 different distributions, or varieties, of Linux.

#### Linux in Cybersecurity

- As a security analyst, you will frequently use Linux to examine logs for system issues and verify access and authorization within identity and access management systems.
- Specific Linux distributions are designed for specialized security tasks, such as digital forensics for investigating event alerts or penetration testing to identify system vulnerabilities.

### **Linux Architecture**

#### Key Components of Linux Architecture

- User: You are the user, initiating tasks and commands on the computer. Linux is a multi-user system, allowing multiple users to access resources simultaneously.
- Applications: These are programs that perform specific tasks, like a word processor or calculator. Applications are often distributed through package managers.

## Interacting with Linux

- **Shell:** The shell is a command-line interpreter that processes commands and outputs results, acting as the Command Line Interface (CLI) for communication with the system.
- **Filesystem Hierarchy Standard (FHS):** This component organizes data within the Linux OS, similar to a filing cabinet, ensuring data can be easily found and accessed.

## Underlying System Operations

- **Kernel:** The kernel manages processes and memory, communicating with hardware to execute commands from the shell. It uses drivers to enable applications to perform tasks efficiently.
- **Hardware:** These are the physical components of the computer, such as the CPU, mouse, and keyboard, which are distinct from software applications.

## 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

### Kali Linux Overview

- Kali Linux is specifically designed for penetration testing and digital forensics.
- It comes with many pre-installed tools and should be used in a virtual machine to prevent system damage and allow for reverting to previous states.

#### Penetration Testing with Kali Linux

- Penetration testing simulates attacks to identify vulnerabilities in systems, networks, and applications.
- Tools like Metasploit, Burp Suite, and John the Ripper are used for exploiting vulnerabilities, testing web applications, and guessing passwords, respectively.

#### Digital Forensics with Kali Linux

- Digital forensics involves collecting and analyzing data to understand what happened after a security incident.
- Kali Linux offers tools such as tcpdump for capturing network traffic, Wireshark for analyzing network traffic, and Autopsy for analyzing hard drives and smartphones.

## 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.

## AlmaLinux

AlmaLinux is a community-driven Linux distribution that was created as a stable replacement for CentOS. CentOS was an open-source distribution that is closely related to Red Hat, and its final stable release, CentOS 8, was in December 2021. CentOS used source code published by Red Hat to provide a similar platform. AlmaLinux is designed to be a drop-in replacement for CentOS 8. This ensures that applications and configurations that worked on CentOS will continue to function on AlmaLinux.

## 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.

## 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.

- **expr**: Used to perform basic math calculations.
- **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

## Different types of Shells

### 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.

## Module 3: Linux Commands in the Bash Shell

### Linux Commands via the Bash Shell

#### Filesystem Hierarchy Standard (FHS)

- The FHS organizes data in Linux, treating everything as a file within a directory.
- It's a hierarchical system, with the root directory (designated by /) at the highest level, and subdirectories branching off from it.

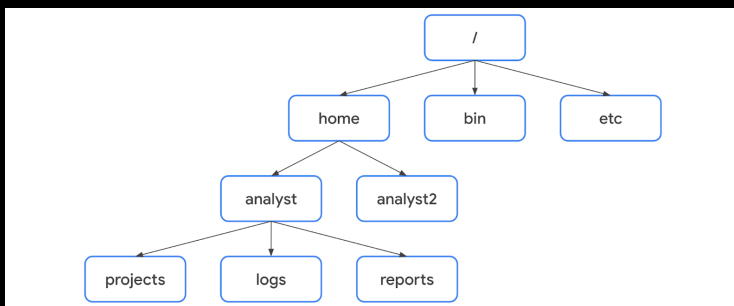
#### Navigating the File System

- The pwd command prints the current working directory, showing your location in the file system.
- The ls command lists the files and directories within the current working directory.

#### Changing Directories and Reading File Content

- The cd command allows you to change your current directory.
- The cat command displays the entire content of a file, while head shows only the beginning (defaulting to the first ten lines).

### Navigate Linux and read file content



#### 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

## Filter content in Linux

### grep

The `grep` command searches a specified file and returns all lines in the file containing a specified string or text. 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.

Let's look at another example: `grep error time_logs.txt`. Here `grep` is used to search for the text pattern. `error` is the term you are looking for in the `time_logs.txt` file. When you run this command, `grep` will scan the `time_logs.txt` file and print only the lines containing the word `error`.

### 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.

## **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.txt` 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.

## 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

### 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+rx,g+rx,o+rx 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.

## 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`.

-- Continued --



## Adding/Removing Users [example]:

Add a new user:

```
1 sudo useradd researcher9
```

```
1 sudo usermod -g research_team researcher9
```

^ Adds user to the **research\_team** group as their **primary group**

```
1 sudo useradd researcher9 -g research_team
```

^ **Performs the last 2 steps in one command instead of 2**

**NOTE: When a user is created, Linux automatically creates a group with the same name.**

Assign file ownership:

```
1 sudo chown researcher9 /home/researcher2/projects/project_r.txt
```

^ Assigns owner of **project\_r.txt** to **researcher9**

```
1 sudo usermod -a -G sales_team researcher9
```

^ Assigns **researcher9** to **sales\_team** as **secondary group**

Delete a user:

```
1 sudo userdel researcher9
```

v Expected outcome of the above command

```
1 Userdel: Group researcher9 not removed because it is not the primary group of user
```

```
1 sudo groupdel researcher9
```

## Man pages within the shell

Accessing Manual Pages

- The `man` command displays comprehensive information about other commands, functioning as a manual.
- For example, `man usermod` provides a general description and details on each of `usermod`'s options, like `-d` for changing a user's home directory.

Quick Command References

- The `whatis` command offers a concise, single-line description of a command.
- Using `whatis tail` quickly reveals that the `tail` command outputs the last part of files.

Searching for Commands

- The `apropos` command searches manual page descriptions for a specified string, helping you find commands when you don't know their exact names.

- You can refine `apropos` searches by adding the `-a` option and an additional string to return only commands containing both specified terms, such as `apropos -a change password`.

## Linux Resources

### 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."

```
1 man cat
```

#### 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.

```
1 apropos -a first part file
```

#### 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.

```
1  whatis rm
```

## Module 4: Databases and SQL

### Databases vs. Spreadsheets

- Databases are designed for large-scale data storage and simultaneous access by multiple users, unlike spreadsheets which are typically for smaller datasets and individual use.
- As a security analyst, you'll frequently interact with databases containing critical information such as login attempts, software updates, and machine ownership.

### Relational Databases and Their Structure

- Relational databases, the focus of this course, organize data into interconnected tables.
- Each table consists of columns (fields) that define the type of data, and rows (records) that contain specific data entries.

### Keys in Relational Databases

- **Primary Key:** A unique identifier for each row in a table, ensuring no duplicate or empty values.
- **Foreign Key:** A column in one table that acts as a primary key in another, establishing relationships between tables and allowing for data connection.

## Query databases with SQL

### Understanding SQL and Queries

- SQL, or **Structured Query Language**, is a programming language used to create, interact with, and request information from databases.
- A query is a request for data from one or more database tables, and nearly all relational databases use some version of SQL for this purpose.

### SQL for Log Retrieval

- Security analysts use SQL to efficiently search through large security logs, which record events within an organization's systems.
- This allows analysts to quickly identify relevant data points, such as improperly configured machines or unusual patterns indicating malicious activity.

### SQL for Data Analytics

- SQL is also valuable for basic data analytics, enabling security analysts to filter data and support security-related decisions.
- Examples include identifying machines that haven't received the latest security patches or determining optimal times for machine updates.

## 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`, and `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`, and `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.

## Basic SQL Query

### 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:

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

Run

Reset

### 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:

```
1 SELECT customerid, city, country
2 FROM customers
3 ORDER BY city;
```

Run

Reset

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:

```
1 SELECT customerid, city, country
2 FROM customers
3 ORDER BY city DESC;
```

Run  
Reset

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:

```
1 SELECT customerid, city, country
2 FROM customers
3 ORDER BY country, city;
```

Run  
Reset

## The WHERE clause and basic operators

### 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:

```
1 SELECT firstname, lastname, title, email
2 FROM employees
3 WHERE title = 'IT Staff';
```

Run

Reset

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:

```
1 SELECT lastname, firstname, title, email
2 FROM employees
3 WHERE title LIKE 'IT%';
```

Run  
Reset

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:

```
1 SELECT firstname, lastname, state, country
2 FROM customers
3 WHERE state LIKE 'N_';
```

Run  
Reset

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

## Operators for filtering dates and numbers

### 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 operators in your filters to make sure you return only the rows you need →

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

operator	use
<	less than
>	greater than
=	equal to
<=	less than or equal to
>=	greater than or equal to
<>	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:

```
1 SELECT firstname, lastname, birthdate
2 FROM employees
3 WHERE birthdate > '1970-01-01';
```

Run

Reset

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:

```
1 SELECT firstname, lastname, hiredate
2 FROM employees
3 WHERE hiredate BETWEEN '2002-01-01' AND '2003-01-01';
```

Run

Reset

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.

## More on filters with AND, OR, and NOT

### 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:

```
1 SELECT firstname, lastname, email, country, supportrepid
2 FROM customers
3 WHERE supportrepid = 5 AND country = 'USA';
```

Run

Reset

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:

```
1 SELECT firstname, lastname, email, country
2 FROM customers
3 WHERE country = 'Canada' OR country = 'USA';
```

Run

Reset

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**:

```
1 SELECT firstname, lastname, email, country
2 FROM customers
3 WHERE NOT country = 'USA';
```

Run  
Reset

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:

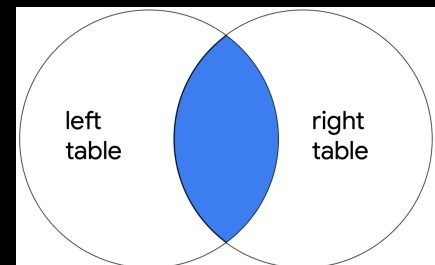
```
1 SELECT firstname, lastname, email, country
2 FROM customers
3 WHERE NOT country = 'Canada' AND NOT country = 'USA';
```

Run  
Reset

## Compare types of Joins

### Inner 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.

— — Continued — —

## 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

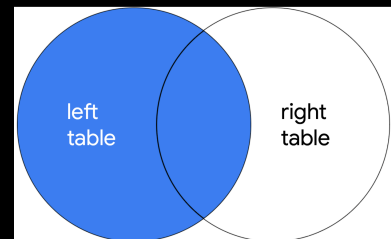
Outer joins expand what is returned from a join. Each type of outer join returns all rows from either one table or both tables.

### 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**.

-- Continued --

## 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

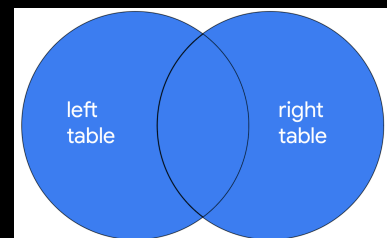
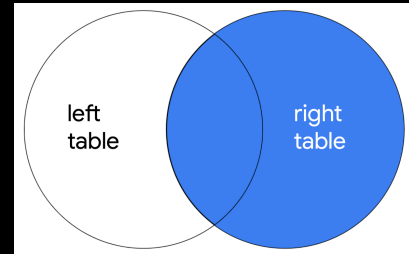
**FULL OUTER JOIN** returns all records from both tables. You can think of it as a way of completely merging two tables.

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.

## Key takeaways



When working in SQL, there are multiple ways to join tables. All joins return the records that match on a specified column. **INNER JOIN** will return only these records. Outer joins also return all other records from one or both of the tables. **LEFT JOIN** returns all records from the first or left table, **RIGHT JOIN** returns all records from the second or right table, and **FULL OUTER JOIN** returns all records from both tables.

## 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:

```
1 SELECT COUNT(firstname)
2 FROM customers;
```

Run

Reset

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:

```
1 SELECT COUNT(firstname)
2 FROM customers
3 WHERE country = 'USA';
```

Run

Reset

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.