linux basics part 2; electric boogalooo

Welcome to week 3; Linux commands via bash the bourne-again shell

Learning to communicate in a new way can be exciting.
Maybe you've learned a new language and can remember this feeling.
Perhaps a lot of us share this excitement with young children as they are expanding their vocabulary.
Others, including me, remember a sense of wonder when we first used a specialized language to communicate with their computer.
In this section, we'll continue to learn more about Linux and how to communicate with the OS through its shell.

You'll utilize the command line to communicate with the OS.
You'll learn how to input commands in the shell and learn about some of the core Linux commands that you'll use as a security analyst.
Specifically, this includes navigating and managing the file system.
You'll also focus on authenticating and authorizing users.
This means you'll be able to use a command line to add and delete users from the system and to control what they have access to.
Finally, there's always more to learn.
We'll cover accessing resources that support learning new Linux commands.

I remember when I first learned about the command line and was shocked at the capabilities it provided.
I didn't need to click through multiple screens to get tasks done.
Although it took some practice and time to get used to, it has been one of the biggest tools at my disposal.
After this section, you'll have a practical experience in an area important to the work of a security analyst: using Linux commands.

Welcome back. Before we get into specific Linux commands, let's explore in more detail the basics of communicating with the OS through the shell.
Being able to utilize Linux commands is a foundational skill for all security professionals.
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.

In this example, our argument was a string of text.
Arguments can provide other types of information as well.
One thing that is really important in Linux is that all commands and arguments are case sensitive.
This includes file and directory names.
Keep that in mind as you learn more about how to use Linux in your day-to-day tasks as a security analyst.
Okay, now that we've covered the basics of entering Linux commands and arguments through the Bash shell, we're ready to learn some specific commands.
This is exciting, so let's get to our next video!

Core commands for navigation and reading files

Welcome back. I hope you're learning a lot about how to communicate with the Linux OS.
As we continue our journey into utilizing the Linux command line, we'll focus on how to navigate the Linux file system.

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, oldreports, 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.

Wow, this section had lots of action, and it's just the beginning!
I'm glad you learned how security analysts can use essential commands to navigate the system.
Next, we'll explore how to manage the system.

Navigate Linux and read file content

In this reading, you'll review how to navigate the file system using Linux commands in Bash. You'll further explore the organization of the Linux Filesystem Hierarchy Standard, review several common Linux commands for navigation and reading file content, and learn a couple of new commands.

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.

h.l.

The `cat` command in Linux is short for "concatenate", which means to link things together in a series or chain. The `cat` command is one of the most commonly used commands in Unix-like

operating systems like Linux. It reads data from files and outputs their contents. It can also concatenate and display the contents of more than one 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

note to future NaruZkurai, this control scheme is ascinine, i will be ripping this command then creating one called
nzkread

Key takeaways

It’s important for security analysts to be able to navigate Linux and the file system of the FHS. Some key commands for navigating the file system include *pwd*, *ls*, and *cd*. Reading file content is also an important skill in the security profession. This can be done with commands such as *cat*, *head*, *tail*, and *less*.

Find what you need with Linux

Now that we covered: pwd, ls, and cd and are familiar with these basic commands for navigating the Linux file system, let's look at a couple of ways to find what you need within this system.
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.

So now you have two different ways that you can filter in Linux while working as an analyst.
Navigating through files and filtering are just part of what you need to know.
Let's keep exploring the Linux command line.

Filter content in Linux

In this reading, you'll continue exploring Linux commands, which can help you filter for the information you need. You'll learn a new Linux command, *find*, which can help you search files and directories for specific information.

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.

Key takeaways

Filtering for information using Linux commands is an important skill for security analysts so that they can customize data to fit their needs. Three key Linux commands for this are *grep*, piping (|), and *find*. These commands can be used to navigate and filter for information in the file system.

Create and modify directories and files

Let's make some branches!
What do I mean by that?
Well, in a previous video, we discussed root directories and how other subdirectories branch off of the root directory.
Let's think again about the file directory system as a tree.
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, et'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 oldreports 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 oldreports has been deleted and no longer appears 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.

Great work!
We've covered a lot of topics here—from creating and removing directories and files to copying or moving them, and just now, we've added editing files.
You're well on your way to learning Linux commands!

Manage directories and files

Previously, you explored how to manage the file system using Linux commands. The following commands were introduced: *mkdir*, *rmdir*, *touch*, *rm*, *mv*, and *cp*. In this reading, you'll review these commands, the nano text editor, and learn another way to write to 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.

Key takeaways

Knowing how to manage the file system in Linux is an important skill for security analysts. Useful commands for this include: *mkdir*, *rmdir*, *touch*, *rm*, *mv*, and *cp*. When security analysts need to write to files, they can use the nano text editor, or the > and >> operators.

File permissions and ownership

Hi there. It's great to have you back!
Let's continue to learn more about how to work in Linux as a security analyst.
In this video, we'll explore file and directory permissions.
We'll learn how Linux represents permissions and how you can check for the permissions associated with files and directories.

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 other.
Other 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,
- 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 other have only read privileges.
After the permissions, ls -l first displays the username.
Here, that's us, analyst.
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.

I thought that was pretty interesting. Did you?
You now know a little more about file permissions and ownership.
This will be helpful when working in security because monitoring and setting correct permissions is essential for protecting information.
Take a small break and meet me in the next video.

Permission commands

Previously, you explored file permissions and the commands that you can use to display and change them. In this reading, you'll review these concepts and also focus on an example of how these commands work together when putting the principle of least privilege into practice.

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

Character	Example	Meaning
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

Character	Example	Meaning
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.

Key takeaways

Managing directory and file permissions may be a part of your work as a security analyst. Using *ls* with the *-l* and *-la* options allows you to investigate directory and file permissions. Using *chmod* allows you to change user permissions and ensure they are aligned with the principle of least privilege.

File permissions and ownership

Hi there. It's great to have you back!
Let's continue to learn more about how to work in Linux as a security analyst.
In this video, we'll explore file and directory permissions.
We'll learn how Linux represents permissions and how you can check for the permissions associated with files and directories.

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 other.
Other 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 other have only read privileges.
After the permissions, ls -l first displays the username.
Here, that's us, analyst.
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.

I thought that was pretty interesting. Did you?
You now know a little more about file permissions and ownership.
This will be helpful when working in security because monitoring and setting correct permissions is essential for protecting information.
Take a small break and meet me in the next video.

Change permissions

Hi there! In the previous video, you learned how to check permissions for a user.
In this video, we're going to learn about changing 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 other.
In this particular example, g indicates we will make some changes to group permissions, and o to permissions for other.
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 group.
The minus sign after o means we want to take them away from other.
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 other.
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 other only has read permissions.
We need to adjust these permissions.

We want to ensure analysts in the security group have write permission, but takeaway read permissions from the owner-type other, so we add write permissions for group and remove read permissions for other.
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 other have been removed.
As mentioned earlier, these hyphens indicate a lack of permissions.
Now, other is lacking all permissions.

Though it requires practice,
working in Linux becomes more natural with time.
I'm glad you're learning a little
more about how to use Linux.

Permission commands

Previously, you explored file permissions and the commands that you can use to display and change them. In this reading, you'll review these concepts and also focus on an example of how these commands work together when putting the principle of least privilege into practice.

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

Character	Example	Meaning
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

Character	Example	Meaning
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.

Key takeaways

Managing directory and file permissions may be a part of your work as a security analyst. Using *ls* with the *-l* and *-la* options allows you to investigate directory and file permissions. Using *chmod* allows you to change user permissions and ensure they are aligned with the principle of least privilege.

Add and delete users

Welcome back! In this video, we are going to discuss adding and deleting users.
This is related to the concept of authentication.
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.

Now, we've covered how to add and delete users and how these actions require sudo.
When using sudo, we have to use our best judgment.
These special privileges must be used responsibly to ensure a secure system.

Responsible use of sudo

Previously, you explored authorization, authentication, and Linux commands with *sudo*, *useradd*, and *userdel*. The *sudo* command is important for security analysts because it allows users to have elevated permissions without risking the system by running commands as the root user. You'll continue exploring authorization, authentication, and Linux commands in this reading and learn two more commands that can be used with *sudo*: *usermod* and *chown*.

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* .

Key takeaways

Authentication is the process of a user verifying their identity, and authorization is the process of determining what they have access to. You can use the *sudo* command to temporarily run commands with elevated privileges to complete authentication and authorization management tasks. Specifically, *useradd*, *userdel, usermod*, and *chown* can be used to manage users and file ownership.

The Linux community

There are so many others just like you who will be using the command line.
Linux's popularity and ease of use has created a large online community that constantly publishes information to help users learn how to operate Linux.
Since Linux is open-source, it has become a global community of users that contribute frequently.

This global community is a huge resource for all Linux users because users can find answers for everyday tasks.
Just searching on the internet will provide many answers.
The easiest way to troubleshoot a task is to search and read about how someone else has done it.
Looking for resources on how to execute a task is a good way for beginners to continue learning.

So far, you've learned how to add users, but imagine if later you want to add a new group.
One way to learn how to do this is to search online.
Let's give this a try through a Google search.
The search results give us many options for adding a group in Linux.

Another reputable source is a Unix & Linux Stack Exchange.
Their answers are ranked with points to display high-quality answers.
Many questions relate to more advanced users and are geared towards troubleshooting.

Well, now you know where to get some extra support whenever in doubt about topics in Linux.
There is a lot of support just a click away.
Coming up, we'll learn how to get support from within the command line itself. Join me.

Linux resources

Previously, you were introduced to the Linux community and some resources that exist to help Linux users. Linux has many options available to give users the information they need. This reading will review these resources. When you're aware of the resources available to you, you can continue to learn Linux independently. You can also discover even more ways that Linux can support your work as a security analyst.

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. aprops comes from the French phrase à *propos,* meaning *"to the purpose".* 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.

Key takeaways

There are many resources available for troubleshooting issues or getting support for Linux. Linux has a large global community of users who ask and answer questions on online resources, such as the Unix and Linux Stack Exchange. You can also use integrated support commands in Linux, such as *man*, *apropos*, and *whatis*.

Resources for more information

There are many resources available online that can help you learn new Linux concepts, review topics, or ask and answer questions with the global Linux community. The Unix and Linux Stack Exchangeis one example, and you can search online to find others.

Wrap-up; Glossary terms from week 3

Congratulations! You completed another section in this course.
Take a minute to think about what you've achieved.
You learned a lot in this section. Let's recap what we covered.

In this section, you utilized the command line to communicate with the OS.
Part of this was using commands for navigating and managing the file system.
And you used other commands for authenticating and authorizing users.
These are all tasks that a security analyst is likely to encounter.

Finally, you learned about accessing resources that support
learning new Linux commands.
With this knowledge, you'll be able to continue learning more and
more about using the command line.

We did it! we learned how to communicate with Linux.
That's a great accomplishment, and
one that will be very useful to you in your career as a security analyst.
You should be proud of the work that you've done so far.

Terms and definitions from Course 4, Week 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