**3.2 Student Guide: Commanding the Command Line**

**Overview**

In today's class, you will expand your command line skills by working in your terminal to complete tasks that require file searching and bulk operations. Throughout class, you will complete a series of activities involving the man, find, grep, and wc commands.

**Class Objectives**

By the end of class, you will be able to:

* Identify and explain the structure of a terminal command.
* Explain how options modify the default behavior of a terminal command.
* Use the man command to list instructions and options for each command.
* Use the find command to locate files based on search parameters.
* Use the grep command to search within the contents of files.
* Use the wc command to count words and lines.
* Combine multiple commands in sequence with pipes to accomplish intermediate IT tasks.

**Slideshow**

* The lesson slides are available on Google Drive here: [3.2 Slides](https://docs.google.com/presentation/d/1xDlYES9Bs_CW6x005yPnjEQCyHArA8QKeg5RDrIfru0/edit#slide=id.g4f80a3047b_0_990).

**01. Welcome Back to Terminal**

Today you will be expanding your command line skill set. Let's review the commands taught in the last class.

Commands for navigating a file directory:

* pwd: Displays the current working directory.
* ls: Lists the directories and files in the current directory.
* cd: Navigates into a directory.
* cd ../: Navigates out of a directory.
* clear: Clears out the terminal history on the page.

Commands for making and removing files and directories:

* mkdir: Creates a new directory.
* rmdir: Removes a directory.
* touch: Creates an empty file.
* rm: Removes a file.

Commands for moving and copying files:

* cp: Copies files.
* mv: Moves files.

Commands for previewing files:

* more: Shows a file one page at a time. Space bar is used to move from page to page.
* less: Similar to more, but allows scrolling up and down pages.
* head: Previews the top 10 lines of a file.
* tail: Previews the bottom 10 lines of a file.

Commands for concatenating and redirecting:

* cat: Concatenates and combines multiple files together.
* >: Writes to a file, and overwrites file if the file name already exists.
* >>: Writes to a file, and appends to the file if the file name already exists.

**02. Activity: Warm-Up**

* [Activity File: Warm-Up](file:///C:\University-of-Minnesota-Boot-Camp\uofm-virt-cyber-pt-06-2021-u-lol\-\blob\master\03-Terminal-and-Bash\2\Activities\02_warmup\unsolved\readme.md)
* [Directories/Files: Warm Up](file:///C:\University-of-Minnesota-Boot-Camp\uofm-virt-cyber-pt-06-2021-u-lol\-\blob\master\03-Terminal-and-Bash\2\Resources\warmup.zip)
* [Solution Guide: Warm-Up](file:///C:\University-of-Minnesota-Boot-Camp\uofm-virt-cyber-pt-06-2021-u-lol\-\blob\master\03-Terminal-and-Bash\2\Activities\02_warmup\solved\readme.md)

**03. Command-Line Structure**

So far we have covered many basic command line commands to complete common IT tasks.

These commands have to follow a specific structure to run successfully.

* The structure we are already familiar with is:

<command> <argument>

* **Arguments** appear immediately after the command if they are inputs for the command. For example: touch myfile.
  + The command is touch.
  + The argument is myfile.

IT professionals often need to run commands with more specific parameters than can be included in the command itself.

* For example: IT analysts need to clean up server space. They want to list out the files by size, so they can delete the largest files first. However, the default behavior of the ls command does not list out the files by size.

Fortunately, commands have **options** that can modify their behavior.

We'll cover three methods for using options:

**Method 1: Adding an option to modify the command's default behavior.**

Example: ls -S

* The added option modifies the behavior of the command. Adding -S after the ls command changes the behavior of the ls command from simply listing the files, to listing them by size, with the largest first.
* The syntax for the above command:
  + ls is the command.
  + -S is the option.
* Options always start with a hyphen.
* The option, just like the command, is case sensitive, meaning it matters whether you use capital or lowercase letters.
  + The lowercase -s will provide a very different result than an uppercase -S.
  + The lowercase -s option prints the size of each file.
* Each command has its own set of options. An option used in one command may behave completely differently for another.

For example:

* + ls -s will print the size of each file in a directory.
  + cat -s will suppress repeated empty output lines.

**Method 2: Adding an option and an argument to modify the default behavior on files or directories.**

Example: cat -n logfile1.txt

* We can use this command to display line numbers on a file called logfile1.txt.
* The default behavior of the cat command is to concatenate multiple files or simply display the contents of a single file.
* To modify the behavior of the cat command for a file, we can run: cat -n logfile1.txt.
  + Adding the option -n modifies the behavior of the cat command to display the line numbers preceding each line.
* The syntax for the above command:
  + cat is the command.
  + -n is the option.
  + logfile1.txt is the argument for the cat command.

**Method 3: Adding options that require their own arguments, called parameters.**

Example: head -n 4 logfile1.txt

* **Parameters** provide additional details on how to modify a command's default behavior.
* We can use the above command to preview the first four lines of logfile1.txt.
* The default behavior of head will display the top 10 lines of a file.
* head's -n option modifies the default behavior by changing the number of lines displayed.
  + The -n option requires a parameter to specify the number of lines to display.
* The syntax for the above command:
  + head is the command.
  + -n is the option. This specific option requires a parameter.
  + 4 is the parameter for the -n option.
  + logfile1.txt is the argument for the head command.
  + logfile1.txt is the argument for the head command.

**Options Demonstration Setup**

Now we will practice using command line options by covering the following scenario:

* You are a security analyst at ACME Corp. Your manager has tasked you with cleaning up some of the evidence files, as server space is running low.
* They asked you to delete the three largest evidence files as long as they don't contain the user Sheila. They will need those files for a future investigation.
* You have been told that the log files do not contain more than 40 lines.

**Options Demonstration**

1. Access the /03-instructor/day2/options/ directory:
   * cd /03-instructor/day2/options/
2. We will first take a look at the files in this directory by running ls.

Note that this shows the following six files:

* + fileA
  + fileB
  + fileC
  + fileD
  + fileE
  + fileF

1. Now, we need to know which are the three largest files.
   * Run the following command:
     + ls -S
   * Note that this lists the files in order of largest to smallest:
     + fileE fileB fileC fileA fileF fileD
2. We now know the three largest files are fileE fileB fileC. Before we delete them we need to confirm that the user Sheila is not named in these files.
   * We can do this by previewing the top 40 lines of the files, since none of them are longer than this.
   * To do this, we will run the following command on the first file: head -n 40 fileE.
   * Note the syntax:
     + head is the command.
     + -n is the option, but requires a parameter.
     + 40 is an argument for the -n option.
     + fileE is the argument for the head command.
   * Note that the user Sheila does not exist in this file.
3. Run the other head commands to preview the other two largest files:
   * head -n 40 fileB
   * head -n 40 fileC
     + Note that fileC contains the user Sheila, so this will not be deleted.
4. Run the following command to delete the two files, as requested by your manager.
   * rm fileE fileB

**Demonstration Summary**

* Terminal commands can use **options** to modify their default behavior.
* Commands can also use **options** with **arguments** to modify their default behavior with files or directories.
* Some **options** require **arguments** called **parameters**, which provide additional details on how to modify the default behavior.

**4. Welcome to Man Pages**

We just covered how commands have default behaviors, and options can modify this default behavior to perform additional tasks.

Additionally, each command:

* Has its own unique list of options.
* Has certain options that require parameters.

So, how can IT and security professionals know and manage all of these options for all of these commands?

* The command line has a valuable resource known as **manual** or **man pages**.
* Man pages are a form of documentation available on the terminal.
* Each command has an associated man page.
* Each man page contains the following:
  + Name of the command
  + Synopsis, which includes the syntax of the command
  + Description
  + Options and option parameters

The command to display the man page is very simple: man <command>.

* For example: man ls will display the man page of the ls command.

**Man Pages Demonstration Setup**

Man pages can assist IT and security professionals with learning new commands.

The following scenario shows how man pages can assist with learning a new command. We will also use this scenario for the following demonstration.

* You are a security analyst at ACME Corp, and your manager has asked you to count the number of server logins on October 13, 2019, as they believe it is higher than usual.
* They told you to use the command wc to count the amount of logins on the server login file.
* You have not used the wc command before and will need to use the man page to learn how.

**Man Pages Demonstration**

1. Access the /03-instructor/day2/manpages/ directory:
   * cd /03-instructor/day2/manpages/
2. Since we have not used the wc command before, we need to learn what it does and how it works. We will run the command to view the man page for wc:
   * man wc
3. Scroll through the man page for the wc command and note the following:
   * The **Name** defines and provides a brief summary of the command:
     + NAME: wc - print newline, word, and byte counts for each file
   * The **Synopsis** shows the format:
     + SYNOPSIS: wc [OPTION]... [FILE]...
     + This means the basic syntax is the wc command, followed by the option, followed by the file to run the command against.
   * The **Description** shows a more detailed definition of the wc command.
   * Below the Description are the options and parameters available for wc.
   * To exit the man page, enter q.
4. Next, we are tasked with counting the number of logins (lines) within a log file called 10\_13\_logs.txt.
   * In the man pages file, several options are listed. The best one for our task is -l, the line count option.
5. Lastly, we will run the command with the option: wc -l 10\_13\_logs.txt.
   * The results show the line count, 53, and the name of the file:
     + 53 10\_13\_logs.txt.

**Man Pages Demonstration Summary**

* Man pages: Documentation that exists in the terminal and provides details and options about command line commands.
* man <command>: The syntax to display the man page for a particular command.
* Man pages can be used to learn about new commands and the options of those commands.

**5. Activity: Learning New Commands**

* [Activity File: Learning New Commands](file:///C:\University-of-Minnesota-Boot-Camp\uofm-virt-cyber-pt-06-2021-u-lol\-\blob\master\03-Terminal-and-Bash\2\Activities\06_learning_new_commands\unsolved\readme.md)
* [Directories/Files: Learning New Commands](file:///C:\University-of-Minnesota-Boot-Camp\uofm-virt-cyber-pt-06-2021-u-lol\-\blob\master\03-Terminal-and-Bash\2\Resources\learning_new_commands.zip)

**6. Activity Review: Learning New Commands**

* [Solution Guide: Learning New Commands](file:///C:\University-of-Minnesota-Boot-Camp\uofm-virt-cyber-pt-06-2021-u-lol\-\blob\master\03-Terminal-and-Bash\2\Activities\06_learning_new_commands\solved\readme.md)

**7. The find Command**

In order to find files or directories, we have been navigating in and out of multiple directories. But this process is very time consuming and some file systems have hundreds of directories to search through.

* For example: You might be asked to find access logs within a server you aren't familiar with. Security professionals are often not provided the exact location of the file, so you may have to navigate through hundreds of directories to find these access logs.

There is a terminal command called find designed to simplify this task by searching for specified files and directories with a single command.

* By default, find will search through the current directory and the subdirectories within that current directory.
* However, find does not look at the contents of a file, only the file name or the directory name.

Now we will now discuss the syntax for the various methods of finding a file. We will be using the same base command throughout the examples, but we will slightly modify the command to achieve different results.

**Syntax for Finding a File**

1. find -type f
   * We'll start with the command used to find **all files**. In this specific example, we are finding all files in our current directory and its subdirectories.
   * In order to do this, we will use the option -type and the required parameter f to indicate that we are searching for files.
2. find -type f -name log.txt
   * Now we will find a **specific file**. In this example, we are finding the files called log.txt in our current directory and its subdirectories.
   * In order to do this, we are using an option, -name, to search for an exact match of the specified parameter log.txt.
3. find -type f -iname log.txt
   * To remove case sensitivity and find all relevant files regardless of whether the file name matches the parameter's case, we will change the -name option to -iname. This option requires a parameter identifying what you are looking for.
   * We can use the above example to find the files called log.txt (lowercase) and LOG.TXT (uppercase) in your current directory and its subdirectories.
4. find -type f -iname \*.txt
   * In this example, we are using a symbol known as a **wildcard** in order to search for all files that end with .txt.
   * The \* wildcard symbol indicates that any file ending with .txt will be included in the results, regardless of what comes before .txt. Using wildcards with find is known as a wildcard search.
   * We can run the above example to find all files ending with lowercase .txt or uppercase .TXT in our current directory and its subdirectories.
   * At times, you may need to search for part of a file name.
     + For example, you may want to look for all file names that begin with a certain date, regardless of what the file name ends with.
     + This can be done with a wildcard, signified by an asterisk \*.
   * Wildcards can come before text, such as \*.txt, or after text such as 0517\*
     + As with the above example, if we use the find command to search for \*.txt, the command might return the following:
       - log1.txt
       - apachefile.txt
       - FILEAB.txt
     + If we use the find command to search for 0517\*, the command might return:
       - 0517apache.log
       - 0517textdata.txt
5. sudo find /root/desktop -type f -iname log.txt
   * In the final example, we will use find to search for a file located in another directory. Specifically, we will find the case insensitive log.txt in the /root/desktop directory.
   * To do this, we place the desired directory after the find command and before the the -type option.

**Syntax for Finding a Directory**

1. find -type d
   * The syntax for finding a directory is exactly the same as for finding a file, except that the option -type requires a parameter, d.
   * The above command will find all directories in your current directory and its subdirectories.
2. find -type d -name logs
   * To find a specific directory name, we add the option -name, which requires the parameter of the name of the directory.
   * With the -name option, find is searching for an exact, case-sensitive match of the parameter you specify.
   * In this example, we are searching for a directory called logs in our current directory and its subdirectories.
3. find -type d -iname logs
   * To find a specific directory with **case insensitivity**, we will change the -name option to -iname, which also requires the parameter of the directory we are searching for.
   * In this example, we are looking for a directory called logs or LOGS.
4. find -type d -iname \*1013
   * The \* symbol indicates this is a wildcard search.
   * In this example, we are finding all directories that end with the date 1013.
     + The command would output directories ending in 1013, despite what comes before. For example:
       - apache1013.log
       - textdata1013.txt
5. sudo find /root/desktop -type d -iname logs
   * Lastly, we can search for a directory within a directory that we're not currently located in.
   * Specifically, this command finds directories called logs in the /root/desktop directory.

**find Demonstration Setup**

* Your manager at ACME Corp has tasked you with finding logs for a certain type of webserver called Apache, for the date of October 13th.
* They told you the directory should be named apache and the log files should have the date noted as 1013 in their file names.
* Since there are many directories, you will use the find command to complete these tasks.

**find Demonstration**

1. The first step is to navigate into the instructor directory called find\_demonstration. Run:
   * cd /03-instructor/day2/
   * cd find\_demonstration
2. Next, you can find the directory called apache with the following command:
   * find -type d -iname apache
     + find: The command used to search for the specified file or directory.
     + -type: The option used to distinguish if we are looking for a file or directory.
     + d: The parameter for the -type option, indicating that we are searching for a directory.
     + -iname: The option indicating that we are searching for a specific case-insensitive value.
     + apache: The parameter indicating the value (file name) we are searching for.
3. Run the command and note that the results show the directory containing the name apache:
   * ./apache
4. Next, we will find the log files that have the date 1013 in their name. Run the following command:
   * find -type f -iname \*1013\*
     + find: The command used to search for the specified file or directory.
     + -type: The option used to distinguish if we are looking for a file or directory.
     + f: The parameter for the -type option, indicating that we are searching for file.
     + -iname: The option indicating that we are searching for a specific case-insensitive value.
     + \*1013\*: The parameter indicating the value that we are searching for.
       - The wildcards on either end indicate that the value 1013 can be located anywhere in the file name.
5. Run the command and note that the results show the path to the two files that contain 1013 in their file name:
   * ./apache/1013\_backuplogs
   * ./apache/apache\_1013

**Demonstration Summary**

* find: Command line command used to locate a file or a directory.
* type f: Option used to find files.
* type d : Option used to find directories.
* name: Additional option used for finding specific file or directory names.
* iname: Additional option used for finding case insensitive names.

**08. Activity: Finding Your Way**

* [Activity File: Finding Your Way](file:///C:\University-of-Minnesota-Boot-Camp\uofm-virt-cyber-pt-06-2021-u-lol\-\blob\master\03-Terminal-and-Bash\2\Activities\09_finding_your_way\unsolved\readme.md)
* [Directories/Files: Finding Your Way](file:///C:\University-of-Minnesota-Boot-Camp\uofm-virt-cyber-pt-06-2021-u-lol\-\blob\master\03-Terminal-and-Bash\2\Resources\finding_your_way.zip)

**09. Activity Review: Finding Your Way**

* [Solution Guide: Finding Your Way](file:///C:\University-of-Minnesota-Boot-Camp\uofm-virt-cyber-pt-06-2021-u-lol\-\blob\master\03-Terminal-and-Bash\2\Activities\09_finding_your_way\solved\readme.md)

**10. Break**

**11. grep Command**

The find command only searches for the names of files, not the contents within.

However, IT and security professionals are often tasked with searching for specific data inside of a file.

* For example: You might be tasked with finding out if a specific user logged in on a specific day. You would first find the access log file for that day and then need to verify if the specific user was inside that log file.

We have previously used preview commands, such as head, more, tail, and less, in order to view a file's contents. The challenges of using preview commands to search for data inside a file include:

* Files that are large in size take a long time to scan for data.
* If you have more than one file to scan, it can take a long time to preview multiple files.
* Manually previewing and scanning files invites human error. Even the best security professionals can overlook a data point if it blends with the other data in the file.

There is a command called grep that allows us to search within a file or multiple files to find a specific data point.

* grep, which stands for "global regular expression print," is a command to search for data inside of files.
* grep by default returns the entire line that the desired data point is found in.
* grep by default will only search for data in the current directory and not in sub-directories.

**Syntax for grep**

1. <grep data\_point File\_to\_search\_inside>
   * In this basic syntax, grep is used to find a specific data point within a **single** file.
   * Next, we'll run through a few examples of the grep command, highlighting the various features and options we can use.
2. grep bob log1.txt
   * In this example, we are using grep to find the lines in which the user bob is mentioned in the file log1.txt.
     + grep: The command being run.
     + bob: The specific data point being searched for.
     + log1.txt: The file being searched for the data point.
   * After this command is run, all the lines where the data point bob was found inside of the file log1.txt will be displayed.
   * If no matches of bob are found in the file, nothing will be returned.
3. grep bob \*.txt
   * In this next example, we are using grep to find a specific data point within **multiple** files.
   * Specifically, we are using grep to find where bob exists within in all .txt files.
     + bob is the specific data point being searched for.
     + \*.txt is the wildcard. \* indicates that the command will search through all files that end with .txt.
   * After this command is run, it will display the files where the value of bob was found, followed by the lines where it was found inside of all the .txt files.
4. grep -i bob \*.txt
   * This grep command can be used to find a **case-insensitive** specific data point within multiple files.
   * Specifically, this command finds the lines where the user bob or BOB exist in all TXT files.
     + grep: The command being run.
     + -i: The option for grep that indicates case insensitivity.
     + bob: The specific data case-insensitive point being searched for.
     + \*.txt is the wildcard of \* that indicates it will search through all files that end with .txt.
5. grep -il bob \*.txt
   * In the final example, we are showing that grep can be used to find the **file name** that contains a specific data point.
   * Specifically, this command only outputs the **file name** where the user of bob or BOB exist within all TXT files. When this command is run, it will only display the names of the file that contain bob.
     + grep: The command being run.
     + -il: Two options placed together.
       - i: An option for grep that indicates case insensitivity.
       - l: An additional option that indicates to only return the file name.
       - Note that i and l are two separate options. However, we can place them under a single hyphen.
     + bob: The specific data point.
     + \*.txt: The wildcard.

**grep Demonstration Setup**

* You manager has now asked for your help with a security investigation into an illegal money transfer that took place on May 17th.
* They told you that the suspect, Sally Stealer, stated that she has never logged in to the company's banking website and also said she definitely did not transfer any money on May 17th.
* You must use the grep command to search the application logs to see if Sally Stealer logged in on that day and, if so, whether she transferred any funds.

**grep Demonstration**

1. Navigate into the instructor directory called grep\_demonstration. Run:
   * cd /03-instructor/day2/
   * cd grep\_demonstration
2. The next step is to see all the log files that exist in this directory. Run ls.
   * This will show three log files, two for 0517, and one for 0519.
3. We will first use grep to see in which files the user Sally Stealer appears. This will show if Sally has ever logged in.
   * Run the following command: grep -il Sallystealer \*
     + grep: The command being run.
     + -il: i is an option for grep that indicates case insensitivity. l indicates to only return the file names.
       - Note that i and l are two separate options. However, we can place them under a single hyphen.
     + Sallystealer: The specific data point being searched for.
     + \*: The wildcard, indicating a search for all files in the current directory
4. Run the command and note that the results show two files that clearly prove that Sally Stealer does have activity.
   * banklogs0517
   * banklogs0519
5. Next, we need to prove if and when Sally Stealer transferred any funds on the day of May 17th.
   * For this example, we will search for the word "transfer."
   * Run the following command: grep -i transfer banklogs0517
     + grep: The command being run.
     + -i: The option indicating case insensitivity.
     + transfer: The specific data point being searched for.
     + banklogs0517: The file we will search through, since the transfer happened on 0517, and this was the only file from this date in which Sally Stealer appears.
6. After running the command, note that the results show the following line:
   * 81.220.24.207 - - [17/May/2015:10:05:52 +0000] "SALLYSTEALER : Transfer funds : $1,000,754 from Company DDA 012 to Personal SAV 876:
   * This clearly proves that Sally Stealer did transfer funds from a company account to a personal account.
   * Furthermore, this happened at 10:05:52 on May 17th.
   * This happened at 10:05:52 on May 17th.

**Demonstration Summary**

* grep: Command-line command to find a data point inside of a file.
  + The basic syntax:

<grep data\_point File\_to\_search\_inside>

* + grep by default will return the **whole line** containing a data point.
* i: Option indicating to search for the data point with case insensitivity.
* l: Option indicating to return only the file names of the files containing the data point.

**12. Activity: grep**

* [Activity File: grep](file:///C:\University-of-Minnesota-Boot-Camp\uofm-virt-cyber-pt-06-2021-u-lol\-\blob\master\03-Terminal-and-Bash\2\Activities\13_grep_activity\unsolved\readme.md)
* [Directories/Files: grep](file:///C:\University-of-Minnesota-Boot-Camp\uofm-virt-cyber-pt-06-2021-u-lol\-\blob\master\03-Terminal-and-Bash\2\Resources\finding_your_way.zip)

**13. Activity Review: grep**

* [Solution Guide: grep](file:///C:\University-of-Minnesota-Boot-Camp\uofm-virt-cyber-pt-06-2021-u-lol\-\blob\master\03-Terminal-and-Bash\2\Activities\13_grep_activity\solved\readme.md)

**14. Combining Commands with Piping**

Today we covered many powerful command line commands that IT and security professionals use, such as:

* find: Searches for file names or directories.
* grep: Searches for data points inside of files.
* wc: Counts lines or words inside of a file.

Security professionals are often tasked with combining these commands to complete certain tasks.

* For example: You may be asked to determine if a user exists in a log file, and how many times that user appears in the log file.
  + We can use the grep command to see *if* a user appears in the log file by redirecting the results into an output file.
  + We can use the wc -l command to determine *how many times* the user appears in the file by counting the results of this output.

Rather than running these multiple commands separately, we can combine them using **pipes**.

* A pipe takes the output of one command and redirects it to another command, in order to complete additional tasks on the output.
* A pipe is designated with the following symbol : |
* Multiple pipes can be used in a single command.
* Pipes are unidirectional, meaning the processing of the data only flows from left to right through the pipeline.

**Pipes Demonstration Setup**

* Your manager at ACME Corp has tasked you with continuing the previous investigation against Sally Stealer. They believe she may have transferred other large amounts of money.
* Your manager created a single file called, largetransfers.txt that contains all transfers over one million dollars.
* Your manager asked you to count how many of those transfers belong to Sally Stealer.

**Pipes Demonstration**

1. The first step is to navigate into the instructor directory called pipes\_demonstration.
   * cd /03-instructor/day2/
   * cd pipes\_demonstration
2. Next, we will use the following grep script to identify the transfers that belong to SallyStealer:
   * grep -i SallyStealer largetransfers.txt
   * It returns many of Sally's large transfers.
3. Next, we will use pipes to add an additional command to count the transfers from the previous query.
   * Run the following: grep -i SallyStealer largetransfers.txt | wc -l
     + grep -i "SallyStealer largetransfers.txt: Lists the lines that have the value SallyStealer.
     + |: Pipes (redirects) the results into the next command.
     + wc -l: Counts the number of lines resulting from the previous output.
4. Run the command and note that the result is now the count of 9.

**Pipes Demonstration Summary**

* A pipe takes the output of one command and redirects it to another command in order to complete an additional task on the output.
* A pipe is designated with the following symbol : |
* Multiple pipes can be used in a single command.
* Pipes are unidirectional, meaning the processing of the data flows from left to right through the pipeline.

**15. Activity: Gathering Evidence**

* [Activity File: Gathering Evidence](file:///C:\University-of-Minnesota-Boot-Camp\uofm-virt-cyber-pt-06-2021-u-lol\-\blob\master\03-Terminal-and-Bash\2\Activities\16_Gathering_Evidence\unsolved\readme.md)
* [Directories/Files: Gathering Evidence](file:///C:\University-of-Minnesota-Boot-Camp\uofm-virt-cyber-pt-06-2021-u-lol\-\blob\master\03-Terminal-and-Bash\2\Resources\Gathering_Evidence.zip)

**16. Activity Review: Gathering Evidence**

* [Solution Guide: Gathering Evidence](file:///C:\University-of-Minnesota-Boot-Camp\uofm-virt-cyber-pt-06-2021-u-lol\-\blob\master\03-Terminal-and-Bash\2\Activities\16_Gathering_Evidence\solved\readme.md)

© 2020 Trilogy Education Services, a 2U, Inc. brand. All Rights Reserved.