Linux Terminal Commands

# BACKGROUND

Linux is an open source operating system invented by [Linus Torvalds](https://en.wikipedia.org/wiki/Linus_Torvalds). There seems to be a perception that Linux is all text based (command line), but most versions of Linux have a graphical user interface (GUI) and look strikingly similar to Windows or MacOS.

That being said, we will primarily be using the command line interface in Linux for the majority of stuff we do in this class.

# REQUIREMENTS

A working version of Linux (virtualized is preferred, but any instance will work).

# PART I: Launch the “terminal” in your instance of Linux.

1. The “terminal” is your command line interface. It’s quite powerful (in fact most professionals prefer using the terminal when working even though most of the actions can be performed with a mouse through the GUI).

To launch the terminal, find it in the menu of apps. Alternatively, you can start typing “terminal” into the search bar and the app will surface.

Take a screenshot of the terminal running and paste it in the box below:

| EVIDENCE #1 |
| --- |
| **PASTE THE IMAGE OF YOUR TERMINAL RUNNING** |

# PART II: Familiarize yourself with these popular commands.

1. The first command to use is “**ls**”. This stands for “list” and will provide a listing of all the files and folders (directories) in the current folder. You might be used to the command “**dir**” in DOS (or Windows Command Prompt) that does the same thing; it stands for “directory”.

Note that your prompt will show you three things:



* “**root**” means that is your username. Traditionally, a username of “root” means you have access to everything.
* “**kali**” is the hostname (name of the computer).
* “**#**” signals your role. A “#” means you have root access whereas a “$” means you are a normal user.

Other instances of Linux may have a different color scheme. In the next example, I have captured the prompt from Linux running on a Chromebook:



You can see that the username is different, the hostname is different, and the prompt indicates that I’m only a normal user and don’t have root access (that’s OK, because I can override that when I want with the command “**sudo**”, but more on that later).

Run **ls** and take a screenshot of the results. I’ve pasted a screenshot of my results in the box below, so you’ll have to delete the image and replace it with your own. My list is most likely different than yours because I have already created some files and directories.

**ls**

| EVIDENCE #2 |
| --- |
| **IMAGE SHOWING THE RESULTS OF ls** |

1. Let’s start using some parameters! In my example, the list is not so easy to read. I think it would be better if the results were listed in one column. If you have a clean install of Linux, you might not have this congestion issue because you don’t have a lot of files and folders. But follow along anyhow.

Linux supports the use of *parameters*; little extra instructions that tag along with a command. For instance, with the **ls** command, you can append different instructions:

* **-a** or **--all** - this will show hidden files (that is, files that start with a “.”)
* **-l** will show the *long* format (file type, permissions, size, etc.)
* **-r** will show files in a reverse sort
* **-t** will show files listed by modification time (newest first)

You can chain these together. For instance, **ls -ltr** will show a detailed list (**l**), organized by last modification (**t**), oldest to newest (**r**). Take a screenshot of a call to **ls -lrt** as well as the results and paste it here.

| EVIDENCE #3 |
| --- |
| **IMAGE SHOWING THE RESULTS OF ls -lrt** |

1. There are actually a number of different parameters that can be used for the **ls** command. If you ever want to know what they are, you *could* Google it, but it is easier if you just refer to the users manual for each command--it’s built into Linux! Use the **man** command (for “manual”) followed by the command you want to investigate.

**man ls**

When you invoke **man**, the material may be more than what can fit in the window. In these cases, you can hit the spacebar to advance to the next section. You can also hit “q” to quit the manual.

Take a screenshot of a call to **man ls**.

| EVIDENCE #4 |
| --- |
| **IMAGE SHOWING THE RESULTS OF man ls** |

1. Let’s take a look at the command **clear**. As you may have guessed, it will clear the text in the terminal. This is especially helpful if you like a clean workspace. Try it!

 **clear**

1. The **mkdir** command will make a directory. It’s analogous to creating a folder in Windows. I’m going to make a directory named after my first name, David. You should make a directory with your first name.

**mkdir david**

Next, I want to change the directory so I’m in the folder named David. Just creating a folder does not bring you into that folder; you’re still in the directory where the folder was created (in this case, root). We use the command **cd** to “change directory”. Note that if you just use **cd** by itself and don’t provide a directory, you will be returned to the root.

**cd david
ls**

Take a screenshot of creating the folder, going into the folder, and then running **ls**:

| EVIDENCE #5 |
| --- |
| **IMAGE SHOWING THE RESULTS OF mkdir, cd, and ls** |

Cool! Now that we are in the folder, let’s work on actually creating and editing a file. There are three steps: creating the file, editing the file, and displaying the content of the file. We’ll start with something stimple. Let’s create a file called daveFile.txt -- to create this, use the **touch** command. **touch** will create a file *if that file does not already exist*. If it does already exist, it will update the timestamp of that file to the current date and time. You should create a file with your name in the place of “dave” in daveFile.txt:

**touch daveFile.txt**

Now let’s edit the file. Editing files from the command line is not super pleasant, but it isn’t entirely unpleasant either. Many professionals like the Vim text editor (and it comes with most distributions of Linux). However, navigating Vim takes a little bit of work and can be frustrating. For our first foray into command line editing, let’s use something a bit more comfortable: NANO

Nano is easy to use. When you launch NANO, you’ll need to follow it with a filename (note that if the file does not exist, it will create the file for you--but if you don’t save the file in NANO, the file will disappear). Go ahead and use NANO to edit your file.

**nano daveFile.txt**

You should now be in the nano editor. It’s a bit primitive, but it gets the job done. Since daveFile.txt doesn’t have any contents, the screen is blank. I’m going to give it two lines of text--you do the same. Feel free to do two or more lines of text, and it can be anything you want.



When it comes time to save, you’ll have to go through some acrobatics. To save, depress CONTROL + X. Then you’ll be prompted to save your work. Type “Y”. Lastly, you’ll be prompted to provide a filename. If you don’t want to change the name of the file, just hit ENTER (or RETURN). Otherwise, furnish a new filename and then press ENTER or RETURN.

To output the contents of a file, you can use the **cat** command. If you’ve taken any programming languages (or you are proficient in Excel), you might recognize that “cat” is short for “concatenate”. While there are plenty of reasons why you may want to concatenate two files together (think of it as gluing the two files together), if you use **cat** with only one file, it will just output the contents.

Let’s output the contents of daveFile.txt into the terminal:

**cat daveFile.txt**

| EVIDENCE #6 |
| --- |
| **IMAGE SHOWING THE RESULTS OF touch, nano, and cat** |

# PART III: Advanced Linux commands

1. We’re almost done! It’s time to learn a pretty common networking tool, and then a way to capture the output. If you’ve dorked around in the Command Prompt in Windows, there’s a good chance you’ve seen **ping**. The ping command sends small data to a destination and lets you know how long it took to get there. This is a great way to test to see if you have network connectivity (it’s usually one of the first things I do when I’m working on a computer that doesn’t seem to have internet access). You can actually ping any valid address, but I like to ping 8.8.8.8 because that is Google’s primary DNS server (so I know it’s almost always up and running).

The ping command in Linux differs from a PC because by default, a ping in Linux will keep sending pings until interrupted (CONTROL + C). Conversely, a call to ping in Windows will send only four (4) pings.

Happily, you can use some of the parameters of ping in Linux to send a specific number of pings. So let’s ping Google five times using the “c” (or “count”) parameter:

**ping 8.8.8.8 -c 5**

Note that your instance of Linux may not have the **ping** command - you’ll know because when you run ping you’ll get an error message that says ‘command not found’. If that’s the case, you’ll need to install it. Don’t panic! It’s easy! Install it with this command:

**sudo apt install iputils-ping**

| EVIDENCE #7 |
| --- |
| **IMAGE SHOWING THE RESULTS OF ping** |

1. And let’s look at one more advanced maneuver - how to dump content of a command into a file. There are a few ways to do this, but we’ll be using the **>>** operator (which redirects output to a specific location--in this case, a file).

Let’s try dumping the contents of a ping command into a file called ping.txt. You *could* create the file and then either replace the contents (>) or append to the contents (>>) the results of a call to ping. But if the file doesn’t exist, it will be created on the fly.

**ping 8.8.8.8 -c 3 > ping.txt**

You might notice that the results of the ping were not displayed on the screen; that is because they were recorded into ping.txt (but if you wanted to display the output on the screen *and* also capture it in a text file, you should look at the tee command--but we won’t cover that here).

For the last part of this lab, let’s put it all together. Run ls to confirm your ping.txt was created, and then cat your ping.txt file to see the results of your ping.

**ls
cat ping.txt**

| EVIDENCE #8 |
| --- |
| **IMAGE SHOWING THE RESULTS OF ping >> ping.txt** |

# PART IV: App Store

1. There is an app store (of sorts) in Linux. Different distributions of Linux have different “package managers” that help you install software. Both Ubuntu and Kali are derivations of Debian Linux and use the apt command (Advanced Package Tool - though you can use dpkg too).
2. You have to know the name of the package (app) you want to install, but this course will tell you the different apps that need to be installed for labs. For this lab, let’s install neofetch. The neofetch package will tell you information about your operating system and your hardware. To install it, type
3. sudo apt install neofetch
4. This will install neofetch! For future use, you can also type sudo apt install neofetch -y and the -y switch will automatically say YES to any prompt that pops up.
5. Run neofetch and take a screenshot of it.

| EVIDENCE #9 |
| --- |
| **IMAGE SHOWING THE RESULTS OF neofetch** |

# CONCLUSION

Here are a list of the commands covered in this lab. If you don’t have a solid grasp on them, then you should research them a little more or play around with them in the Linux terminal.

ls

man

clear

mkdir (the opposite is rmdir, which removes the directory)

cd

touch

nano

cat

ping

> and >>