

# A1 – Linux Fundamentals

BY: MIA WEBER

CSCI 420 SOFTWARE SECURITY

FEB. 6, 2023

## CONTENTS

Linux Fundamentals Part I .....	2
Linux Fundamentals Part II .....	4
Linux Fundamentals Part III .....	7

# Linux Fundamentals Part I

## 1.1 Introduction

Just like Windows, IOS, and MacOS, Linux is another operating system and can power anything from smart cars to android devices, home appliances, and more.

## 1.2 Where is Linux Used and Different Flavors of Linux

Linux is much more lightweight than Windows and therefore is commonly found on PoS (Point of Sale) machines, entertainment panels on cars, websites, and traffic control sensors. The term “Linux” is an umbrella term for many other operating system variants of “UNIX”. Because UNIX is open-source, Linux variants are flexible and can be specifically selected based on the goals of the task at hand. An Ubuntu server can actually run on a system with only 512MB of RAM, so Linux is very lightweight. The first release of a Linux operating system was in 1991.

## 1.3 Interacting with Your first Linux Machine

The goal of this module is to launch the built in browser version of Ubuntu which will be used to complete the rest of part I.

## 1.4 Running your First Few Commands

Part of what makes Linux environments so lightweight is that they often don’t come equipped with a GUI (or graphical user interface) unless one is installed by the user. This means that the majority of the interaction with the computer is done in the command line in the terminal. The first two commands that are introduced are **echo** which outputs exactly what the user tells the machine to output. For example, typing **echo “Hello World!”** into the terminal would result in the output “Hello World!” as seen below. The other command that we are shown is “whoami” which outputs the name of the current user that is logged in as seen below.

```
Welcome to Ubuntu 20.04.3 LTS (GNU/Linux 5.11.0-1017-aws x86_64)

* Documentation:  https://help.ubuntu.com
* Management:    https://landscape.canonical.com
* Support:        https://ubuntu.com/advantage

System information as of Mon Jan 30 16:38:28 UTC 2023

System load:  0.03          Processes:      110
Usage of /:   18.7% of 9.63GB Users logged in:   0
Memory usage: 40%          IPv4 address for ens5: 10.10.144.248
Swap usage:   0%

0 updates can be applied immediately.

The list of available updates is more than a week old.
To check for new updates run: sudo apt update

Last login: Mon Jan 30 16:37:53 2023 from 10.100.2.209
tryhackme@linux1:~$ echo "Hello World!"
Hello World!
tryhackme@linux1:~$ whoami
tryhackme
tryhackme@linux1:~$
```

## 1.5 Interacting With the Filesystem

The command **ls** lists the files and directories in the current location. One cool feature of **ls** is that you can list the contents of a directory without actually changing the current directory to the one you want to view. This can be seen in the screenshot below. The command **cd** changes the current directory. The command **cd** changes your current directory and can be seen in the screenshot below. The command **cat** displays the contents of a file and is followed by either the name of the file which is stored in the current location OR by the path to the file which is stored in a different location within the file system. This can be seen in the screenshot below. The command **pwd** stands for print working directory and tells the user exactly where they are within the file system. This command is especially useful in navigating back to the current location.

```
tryhackme@linux1:~$ ls
access.log  folder1  folder2  folder3  folder4
tryhackme@linux1:~$ cd folder4
tryhackme@linux1:~/folder4$ ls
note.txt
tryhackme@linux1:~/folder4$ cat note.txt
Hello World!
tryhackme@linux1:~/folder4$ pwd
/home/tryhackme/folder4
tryhackme@linux1:~/folder4$
```

## 1.6 Searching for Files

Linux has a lot of opportunities for increasing efficiency. For example, instead of using commands like the ones introduced above, we can use commands like **find** to automate the search for files in the file system. When you know the name of the file you are searching for but can't remember where it is stored then you can use the command as follows: **find -name passwords.txt** if you were looking for the file "passwords.txt". We can also search with more general terms using the wildcard (\*) in order to search for all files with a certain extension like this: **find -name \*.txt** which will return all text files. This is all shown in the screenshot below.

```
tryhackme@linux1:~$ find -name note.txt
./folder4/note.txt
tryhackme@linux1:~$ find -name *.txt
./folder4/note.txt
tryhackme@linux1:~$ find -name passwords.txt
tryhackme@linux1:~$
```

The command **grep** is also super helpful. The grep command allows us to search for specific occurrences of a specific value within files. Grep can be used to search the entire contents of a file for any entries of the value that we are searching for. In the example below, we use grep to search for a specific IP address within a log file. Grep can be thought of as a pattern matching tool.

```
tryhackme@linux1:~$ grep "81.143.211.90" access.log
tryhackme@linux1:~$ grep "THM" access.log
13.127.130.212 - - [04/May/2021:08:35:26 +0000] "GET THM{ACCESS} lang=en HTTP/1.1" 404 360 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/77.0.3865.120 Safari/537.36"
tryhackme@linux1:~$
```

## 1.7 An Introduction to Shell Operators

The **&** operator allows you to execute commands in the background of the terminal. Commands that will take a long time, such as file copies, provide a great opportunity to use the **&** operator. The **&&** operator allows you to execute multiple commands with one line of the terminal. It is important to note that the second command will only execute if the first command was successful. The **>** operator is called an output redirector and allows you to take the output of one command that you run and send it somewhere else. An example of using the **>** operator is redirecting the output of an echo command to a text file, effectively writing to a file. It is important to note that if there is anything already in the file, running a command to redirect the output of an echo command will overwrite anything that is already in the file. The **>>** operator is also an output redirector but unlike the **>** operator which will overwrite anything existing, the **>>** operator will just add the data to the end of the file. Using **>>** will simply add the redirected output to the end of the file whereas using **>** would overwrite anything existing with the redirected output. An example using each of these output redirector operators is below.

```
tryhackme@linux1:~$ echo password123 > passwords
tryhackme@linux1:~$ echo tryhackme >> passwords
tryhackme@linux1:~$ cat passwords
password123
tryhackme
tryhackme@linux1:~$
```

## 1.8 Conclusions and Summaries

The biggest takeaways from Part I were the importance of knowing and building proficiency with the basic Linux file system navigation commands such as **cd**, **ls**, and **pwd**. Linux can be incredibly powerful, lightweight, and efficient but the extent of each of those is dependent on how well the user can navigate and perform tasks in the environment. The most important thing for me to remember from this section is the distinction between each of the output redirectors (**>** and **>>**). Below is a screenshot of the completed Part I.



## Linux Fundamentals Part II

### 2.1 Introduction

Part II introduces flags and arguments, how to copy and move files in the terminal, an introduction to access mechanisms, and running some scripts and executables.

### 2.2 Accessing your Linux Machine using SSH

## 2.3 Introduction to Flags and Switches

```

tryhackme@linux2:~$ ls
File Edit View Search Terminal Help
tryhackme@linux2:~$ ls
important myfile myfolder unknown1
tryhackme@linux2:~$ ls -la
.. .bash_logout .cache important myfolder
. .bashrc .profile myfile unknown1
tryhackme@linux2:~$ ls --help
Usage: ls [OPTION]... [FILE]...
List information about the FILES (the current directory by default).
Sort entries alphabetically if none of -ctfuvsUX nor --sort is specified.
Mandatory arguments to long options are mandatory for short options too.
-a, --all do not ignore entries starting with .
-A, --almost-all do not list implied . and ..
-l, --long with -l, print the author of each file
-b, --escape print C-style escapes for nongraphic characters
-rs --block-size=SIZE with -l, scale sizes by SIZE when printing them;
e.g., '--block-size=M'; see SIZE format below
ow -B, --ignore-backups do not list implied entries ending with ~
-c with -lt: sort by, and show, ctime (time of last
ast modification of file status information);

```

```
tryhackme@linux2: ~  
File Edit View Search Terminal Help  
LS(1) User Commands LS(1)  
  
NAME  
ls - list directory contents  
  
SYNOPSIS  
ls [OPTION]... [FILE]...  
  
DESCRIPTION  
List information about the FILES (the current directory by default). Sort entries alphabetically if none of -cftuvSUX nor --sort is specified.  
  
Mandatory arguments to long options are mandatory for short options too.  
  
-a, --all  
do not ignore entries starting with .  
  
-A, --almost-all  
do not list implied . and ..  
  
--author  
Manual page ls(1) line 1 (press h for help or q to quit)
```

The **touch** command can be used to create a file and takes exactly one argument- the name of the file that we want to create. The touch command simply makes a blank file, and you would need to use commands like **echo** or text editors such as vi or nano to actually add data to the file. The command

**mkdir**, short for make directory can be used to create an empty directory. Like **touch** it takes one argument- the name of the new (and empty) directory. The command **rm**, short for remove, can delete a file. In order to delete a directory then we need to use **rm -R**. **-R** is a flag for recursive so that the directory and all files contained in that directory will be removed. The command **cp**, short for copy, copies a file. The command takes two arguments- the name of the existing file and the name we wish to assign to the new file when copying. The command **mv**, short for move, moves a file from one location to another. The command takes two arguments just like **cp**. The **mv** command allows you to both move a file to a new directory as well as simply rename a file. The command **file** can be used to determine the type of a file (or the file's extension such as text files or .txt files). All of these commands can be seen in the screenshots below:

```

tryhackme@linux2: ~
File Edit View Search Terminal Help
tryhackme@linux2:~$ touch note
tryhackme@linux2:~$ ls
important myfile myfolder note unknown1
tryhackme@linux2:~$ mkdir mydirectory
tryhackme@linux2:~$ ls
important mydirectory myfile myfolder note unknown1
tryhackme@linux2:~$ rm note
tryhackme@linux2:~$ ls
important mydirectory myfile myfolder unknown1
tryhackme@linux2:~$ rm -R mydirectory
tryhackme@linux2:~$ ls
important myfile myfolder unknown1
tryhackme@linux2:~$ file unknown1
unknown1: ASCII text
tryhackme@linux2:~$

```

## 2.5 Permissions 101

Because certain files can only be accessed by certain users, the switch **-l** can be used in conjunction with the **ls** command in order to display the permission rights for files. There are three main types of permissions that a user/group can be granted- read (r), write (w), and execute (x). An example of this is shown in the screenshot below. In Linux there is also an important distinction that needs to be made between users and groups. While groups may have certain privileges and permissions within a system, users may have an entirely different set of permissions so that a certain group could have executable permissions on a file, but a particular user might not. Sometimes users and groups will have the same permissions for a file, and other times they might be different. The command **su** allows us to change the user and takes one argument- the name of the user that you want to switch to. The flag **-l**, when added will allow you to mimic the actual user logging into the system and will update the terminal to drop you into the user's home directory.

```

tryhackme@linux2: ~
File Edit View Search Terminal Help
tryhackme@linux2:~$ ls -la
total 36
drwxr-xr-x 4 tryhackme tryhackme 4096 Jan 30 18:44 .
drwxr-xr-x 5 root      root      4096 May  4 2021 ..
-rw-r--r-- 1 tryhackme tryhackme 220 May  4 2021 .bash_logout
-rw-r--r-- 1 tryhackme tryhackme 3771 May  4 2021 .bashrc
-rwx----- 2 tryhackme tryhackme 4096 Jan 30 18:22 .cache
-rw-r--r-- 1 tryhackme tryhackme 807 May  4 2021 .profile
-rw-r--r-- 1 user2     user2     14 May  5 2021 important
-rwxr-xr-x 2 tryhackme tryhackme 4096 Jan 30 18:44 myfolder
-rw-r--r-- 1 tryhackme tryhackme 17 May  4 2021 unknown1
tryhackme@linux2:~$

```

## 2.6 Common Directories

**/etc** stores files that are essential to and used by your operating system. **/var** is for variable data such as data that isn't associated with a particular user on your system or log files for services and

processes. **/root** is the home directory for the root user. **/tmp** is a Linux specific directory that is volatile and is used to store data that only needs to be accessed once or twice. Once the computer is restarted, the contents of the **/tmp** folder are cleared out (similar to memory). Because of this, the **tmp** directory is a great place to store things like enumeration scripts.

## 2.7 Conclusions and Summaries

The biggest takeaways from Part II are using SSH to connect to a remote machine, using flags and switches to execute more specific commands in the terminal, more commands to interact with the filesystem, and learning about important root directories on a Linux install and where certain files or data types might be stored in a typical Linux machine. The most important thing for me to remember is that the **/root** directory is actually the home directory of the root user and not the **/home** directory. Below is the screenshot of the completed Part II.



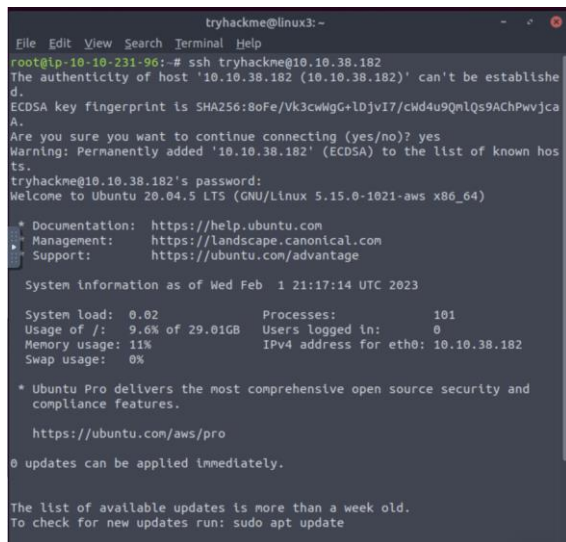
## Linux Fundamentals Part III

### 3.1 Introduction

The goal of this section is to learn about automation, package management, and service/application logging.

### 3.2 Deploy your Linux Machine

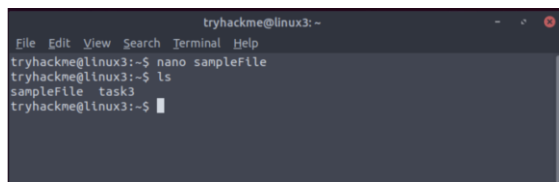
This step is deploying and setting up the Linux machine. Below is the process for using SSH to access the Linux machine.



```
tryhackme@linux3:~  
File Edit View Search Terminal Help  
root@ip-10-10-231-96:~# ssh tryhackme@10.10.38.182  
The authenticity of host '10.10.38.182 (10.10.38.182)' can't be established.  
ECDSA key fingerprint is SHA256:8oFe/Vk3cWgG+lDjvI7/cWd4u9QmlQs9AchPwvjca  
A.  
Are you sure you want to continue connecting (yes/no)? yes  
Warning: Permanently added '10.10.38.182' (ECDSA) to the list of known hosts.  
tryhackme@10.10.38.182's password:  
Welcome to Ubuntu 20.04.5 LTS (GNU/Linux 5.15.0-1021-aws x86_64)  
  
 * Documentation:  https://help.ubuntu.com  
 * Management:    https://landscape.canonical.com  
 * Support:       https://ubuntu.com/advantage  
  
System information as of Wed Feb  1 21:17:14 UTC 2023  
  
System load:  0.02          Processes:            101  
Usage of /:   9.6% of 29.01GB Users logged in:      0  
Memory usage: 11%          IPv4 address for eth0: 10.10.38.182  
Swap usage:   0%  
  
 * Ubuntu Pro delivers the most comprehensive open source security and  
  compliance features.  
  
  https://ubuntu.com/aws/pro  
  
0 updates can be applied immediately.  
  
The list of available updates is more than a week old.  
To check for new updates run: sudo apt update
```

### 3.3 Terminal Text Editors

Nano is one popular choice when it comes to terminal text editors. The command ***nano filename*** can be used to create or edit a file using nano. VIM is a more advanced terminal text editor. Some of the benefits to using VIM include how customizable it is because you can modify keyboard shortcuts to suit your needs. There is also a syntax highlighting functionality which is helpful for writing code. VIM also works on all terminals even when nano might not be installed. Below is a screenshot showing how to use nano to create and edit the contents of a file.



```
tryhackme@linux3:~  
File Edit View Search Terminal Help  
tryhackme@linux3:~$ nano sampleFile  
tryhackme@linux3:~$ ls  
sampleFile  task3  
tryhackme@linux3:~$
```

### 3.4 General/Useful Utilities

The command ***wget*** allows for the downloading of files from the web using HTTP just as if you were accessing files in a browser. All that wget requires is the address of the resource that we want to download. For example, it might look like ***wget http://path/of/file.txt***. Transferring Files from host- SCP (SSH). The SCP, or secure copy, command allows you to transfer files between two computers using SSH in order to provide authentication and encryption. SCP works on a model of source and destination which is what allows for the copying of files and directories from the current system to the remote system and from the remote system to the current system. An example of using SCP to copy files is below:



Variable	Value
The IP address of the remote system	192.168.1.30
User on the remote system	ubuntu
Name of the file on the local system	important.txt
Name that we wish to store the file as on the remote system	transferred.txt

With this information, let's craft our `scp` command (remembering that the format of SCP is just SOURCE and DESTINATION)

```
scp important.txt ubuntu@192.168.1.30:/home/ubuntu/transferred.txt
```

And now let's reverse this and layout the syntax for using `scp` to copy a file from a remote computer that we're not logged into

Variable	Value
IP address of the remote system	192.168.1.30
User on the remote system	ubuntu
Name of the file on the remote system	documents.txt
Name that we wish to store the file as on our system	notes.txt

The command will now look like the following:

```
scp ubuntu@192.168.1.30:/home/ubuntu/documents.txt notes.txt
```

Serving Files From your Host – WEB allows for downloading of files using **wget** and **curl** (based on built in python capabilities).

### 3.5 Processes 101

A process is a program that is running on a machine and is managed by the kernel. Each process will have an ID associated with it known as the PID. The PID increments so that, for example, the 60<sup>th</sup> process will have a PID of 60. The command **ps** will display a list of all running processes as well as some other information about the status of the process. The command **top** gives you real time information about the processes that are actively running. The command **kill** allows you to kill a process. The command requires the PID of the process that you would like to kill. **SIGTERM** kills a process but allows it to cleanup some tasks, **SIGKILL** kills a process without any cleanup, and **SIGSTOP** stops or suspends a process. Namespaces allow you split up your computer into different sections (like cutting a cake). The processes within that section will have access to certain amounts of computing power and make security a priority since only processes that are in the same namespace are able to see each other. Getting Processes/Services to start on boot: The command **systemctl** allows for interaction with the systemd process/daemon. The formatting is: **systemctl [option] [service]** where option can be replaced with Start, Stop, Enable, or Disable. Processes can run in the background or the foreground. Commands such as copying files is great to do in the background because it means that we can continue on with other commands without having to wait for the file copy command to finish first. We can use **Ctrl + Z** or the **&** to move a process to the background. The command **fg** brings the process back to the foreground.

### 3.6 Maintaining Your System: Automation

Crontab is one of the processes that is started during boot and is responsible for facilitating and managing cron jobs. Crontabs have 6 specific values: MIN (minute), HOUR (hour), DOM (day of month),

MON (month), DOW (day of the week), CMD (actual command to execute). Crontabs can be edited by using the command: **crontab -e**.

### 3.7 Maintaining Your System: Package Management

**Introducing Packages & Software Repos:** Linux is especially helpful when developers are ready to release their programs and tools into the wild thanks to increased user accessibility and the merit of open-source tools. **Managing Your Repositories (Adding and Removing):** We can use the **apt** command to install software onto an Ubuntu system. Apt contains a whole suite of tools that allows for the management of packages and sources of the software and to install or remove software at the same time. We can add a repository with the **add-apt-repository** command. GPG (Gnu Privacy Guard) keys are a safety check from the developers that ensure that the software you download is authentic- because the keys match.

### 3.8 Maintaining Your System: Logs

View and get information from logs about the health and activity of your status.

### 3.9 Conclusions & Summaries

The biggest takeaways from Part III are the most efficient ways to make use of terminal text editors like vi and nano, general utilities such as downloading and serving contents using a python webserver, looking more specifically processes and how to monitor them, as well as maintaining and automating a system using crontabs, package management, and reviewing logs. The most important thing for me to remember from Part III is how important it can be to monitor and manage processes, packages, and logs. I don't have very much experience working with logs or managing packages so that was a big area where I learned a lot about the importance of paying attention to packages and logs.

