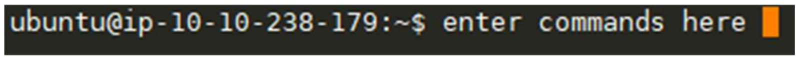
Linux Fundamental

Running Your First Few Commands

The "Terminal" is purely text-based and is intimidating at first. However, if we break down some of the commands, after some time, you quickly become familiar with using the terminal!



```
ubuntu@ip-10-10-238-179:~$ enter commands here
```

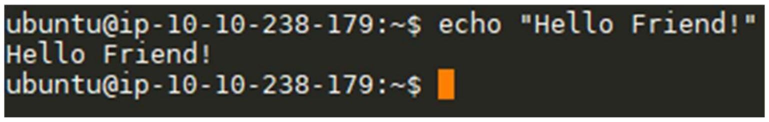
A picture of what a "Terminal" looks like

We need to be able to do basic functions like navigate to files, output their contents and make files! The commands to do so are self-explanatory (once you know what they are of course...)

Let's get started with two of the first commands which I have broken down in the table below:

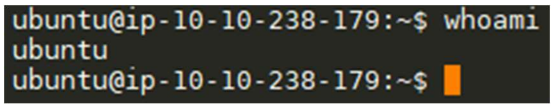
Command	Description
echo	Output any text that we provide
whoami	Find out what user we're currently logged in as!

See the screenshots below for an example of each command being used...



```
ubuntu@ip-10-10-238-179:~$ echo "Hello Friend!"
Hello Friend!
ubuntu@ip-10-10-238-179:~$
```

Using **echo** to output the text "Hello Friend"



```
ubuntu@ip-10-10-238-179:~$ whoami
ubuntu
ubuntu@ip-10-10-238-179:~$
```

Using **whoami** to find out the username of who we're logged in as

Try this on your Linux machine now!

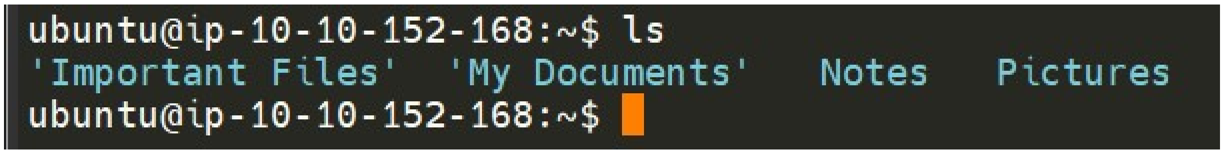
Interacting With the Filesystem

As I previously stated, being able to navigate the machine that you are logged into without relying on a desktop environment is pretty important. After all, what's the point of logging in if we can't go anywhere?

Command	Full Name
ls	listing
cd	change directory
cat	concatenate
pwd	print working directory

Listing Files in Our Current Directory (ls)

Before we can do anything such as finding out the contents of any files or folders, we need to know what exists in the first place. This can be done using the "ls" command (short for listing)



```
ubuntu@ip-10-10-152-168:~$ ls
'Important Files'  'My Documents'   Notes   Pictures
ubuntu@ip-10-10-152-168:~$
```

In the screenshot above, we can see there are the following directories/folders:

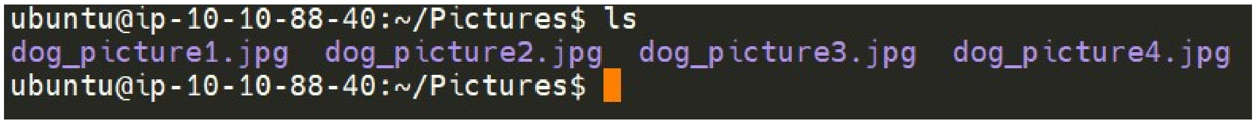
- Important Files
- My Documents
- Notes
- Pictures

Great! You can probably take a guess as to what to expect a folder to contain given by its name.

*Pro tip: You can list the contents of a directory without having to navigate to it by using **ls** and the name of the directory. I.e. ls Pictures*

Changing Our Current Directory (cd)

Now that we know what folders exist, we need to use the "**cd**" command (short for **c**hange **d**irectory) to change to that directory. Say if I wanted to open the "Pictures" directory - I'd do "**cd Pictures**". Where again, we want to find out the contents of this "Pictures" directory and to do so, we'd use "**ls**" again:



```
ubuntu@ip-10-10-88-40:~/Pictures$ ls
dog_picture1.jpg  dog_picture2.jpg  dog_picture3.jpg  dog_picture4.jpg
ubuntu@ip-10-10-88-40:~/Pictures$ 
```

In this case, it looks like there are 4 pictures of dogs!

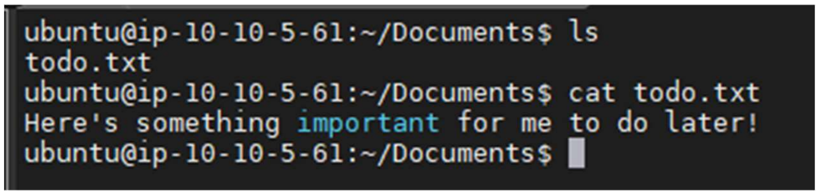
Outputting the Contents of a File (cat)

Whilst knowing about the existence of files is great — it's not all that useful unless we're able to view the contents of them.

We will come on to discuss some of the tools available to us that allows us to transfer files from one machine to another in a later room. But for now, we're going to talk about simply seeing the contents of text files using a command called "**cat".**

"Cat" is short for concatenating & is a fantastic way us to output the contents of files (not just text files!).

In the screenshot below, you can see how I have combined the use of "ls" to list the files within a directory called "Documents":



```
ubuntu@ip-10-10-5-61:~/Documents$ ls
todo.txt
ubuntu@ip-10-10-5-61:~/Documents$ cat todo.txt
Here's something important for me to do later!
ubuntu@ip-10-10-5-61:~/Documents$ 
```

We've applied some knowledge from earlier in this task to do the following:

1. Used "**ls**" to let us know what files are available in the "Documents" folder of this machine. In this case, it is called "todo.txt".
2. We have then used cat todo.txt to concatenate/output the contents of this "todo.txt" file, where the contents are "Here's something important for me to do later!"

*Pro tip: You can use cat to output the contents of a file within directories without having to navigate to it by using **cat** and the name of the directory. I.e. cat /home/ubuntu/Documents/todo.txt*

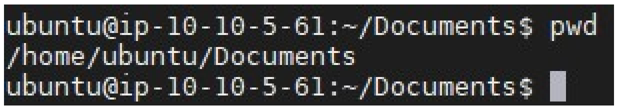
Sometimes things like usernames, passwords (yes - really..), flags or configuration settings are stored within files where "cat" can be used to retrieve these.

Finding out the full Path to our Current Working Directory (pwd)

You'll notice as you progress through navigating your Linux machine, the name of the directory that you are currently working in will be listed in your terminal.

It's easy to lose track of where we are on the filesystem exactly, which is why I want to introduce "**pwd**". This stands for **p**rint **w**orking **d**irectory.

Using the example machine from before, we are currently in the "Documents" folder — but where is this exactly on the Linux machine's filesystem? We can find this out using this "pwd" command like within the screenshot below:



```
ubuntu@ip-10-10-5-61:~/Documents$ pwd
/home/ubuntu/Documents
ubuntu@ip-10-10-5-61:~/Documents$
```

Let's break this down:

1. We already know we're in "Documents" thanks to our terminal, but at this point in time, we have no idea where "Documents" is stored so that we can get back to it easily in the future.
2. I have used the "**pwd**" (**p**rint **w**orking **d**irectory) command to find the full file path of this "Documents" folder.
3. We're helpfully told by Linux that this "Documents" directory is stored at "/home/ubuntu/Documents" on the machine — great to know!
4. Now in the future, if we find ourselves in a different location, we can just use cd /home/ubuntu/Documents to change our working directory to this "Documents" directory.

Although it doesn't seem like it so far, one of the redeeming features of Linux is truly how efficient you can be with it. With that said, you can only be as efficient as you are familiar with it of course. As you interact with OSs such as Ubuntu over time, essential commands like those we've already covered will start to become muscle-memory.

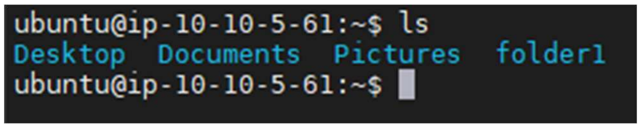
One fantastic way to show just how efficient you can be with systems like this is using a set of commands to quickly search for files across the entire system that our user has access to. No need to consistently use cd and ls to find out what is where. Instead, we can use commands such as find to automate things like this for us!

This is where Linux starts to become a bit more intimidating to approach -- but we'll break this down and ease you into it.

Using Find

The find command is fantastic in the sense that it can be used both very simply or rather complex depending upon what it is you want to do exactly. In fact, so much so, we have an entire room dedicated to using & practising the find command. However, let's stick to the fundamentals first.

Take the screenshot below, we can see a list of directories available to us:



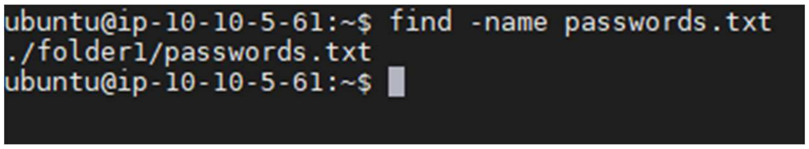
```
ubuntu@ip-10-10-5-61:~$ ls
Desktop  Documents  Pictures  folder1
ubuntu@ip-10-10-5-61:~$ 
```

1. Desktop
2. Documents
3. Pictures
4. folder1

Now, of course, directories can contain even more directories within themselves. It becomes a headache when we're having to through every single one just to try and look for specific files. We can use find to do just this for us!

Let's start simple and assume that we already know the name of the file we're looking for — but can't remember where it is exactly! In this case, we're looking for "passwords.txt"

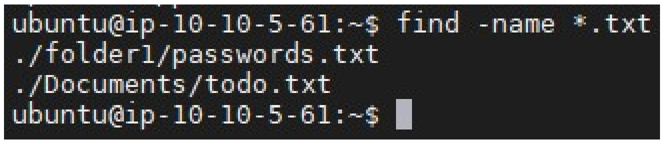
If we remember the filename, we can simply use find -name passwords.txt where the command will look through every folder in our current directory for that specific file like so:



```
ubuntu@ip-10-10-5-61:~$ find -name passwords.txt
./folder1/passwords.txt
ubuntu@ip-10-10-5-61:~$ 
```

"Find" has managed to *find* the file — it turns out it is located in folder1/passwords.txt — sweet. But let's say that we don't know the name of the file, or want to search for every file that has an extension such as ".txt". Find let's us do that too!

We can simply use what's known as a wildcard (*) to search for anything that has .txt at the end. In our case, we want to find every .txt file that's in our current directory. We will construct a command such as find -name *.txt . Where "Find" has been able to *find* every .txt file and has then given us the location of each one:



```
ubuntu@ip-10-10-5-61:~$ find -name *.txt
./folder1/passwords.txt
./Documents/todo.txt
ubuntu@ip-10-10-5-61:~$
```

Find has managed to *find*:

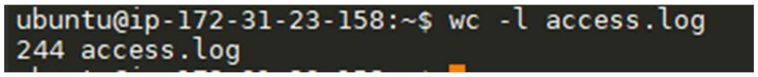
1. "passwords.txt" located within "folder1"
2. "todo.txt" located within "Documents"

That wasn't so tough huh!

Using Grep

Another great utility that is a great one to learn about is the use of grep. The grep command allows us to search the contents of files for specific values that we are looking for.

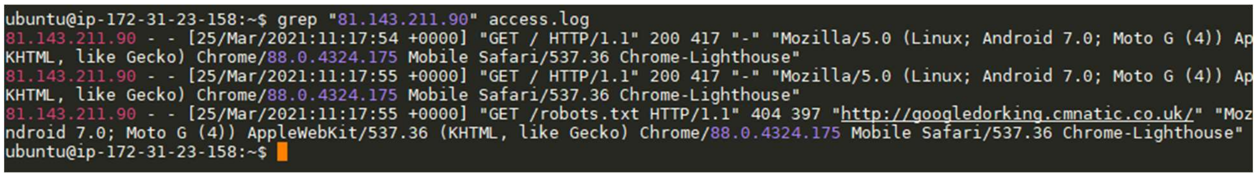
Take for example the access log of a web server. In this case, the access.log of a web server has 244 entries.



```
ubuntu@ip-172-31-23-158:~$ wc -l access.log
244 access.log
```

Using a command like cat isn't going to cut it too well here. Let's say for example if we wanted to search this log file to see the things that a certain user/IP address visited? Looking through 244 entries isn't all that efficient considering we want to find a specific value.

We can use grep to search the entire contents of this file for any entries of the value that we are searching for. Going with the example of a web server's access log, we want to see everything that the IP address "81.143.211.90" has visited (note that this is fictional)



```
ubuntu@ip-172-31-23-158:~$ grep "81.143.211.90" access.log
81.143.211.90 - - [25/Mar/2021:11:17:54 +0000] "GET / HTTP/1.1" 200 417 "-" "Mozilla/5.0 (Linux; Android 7.0; Moto G (4)) Ap
KHTML, like Gecko) Chrome/88.0.4324.175 Mobile Safari/537.36 Chrome-Lighthouse"
81.143.211.90 - - [25/Mar/2021:11:17:55 +0000] "GET / HTTP/1.1" 200 417 "-" "Mozilla/5.0 (Linux; Android 7.0; Moto G (4)) Ap
KHTML, like Gecko) Chrome/88.0.4324.175 Mobile Safari/537.36 Chrome-Lighthouse"
81.143.211.90 - - [25/Mar/2021:11:17:55 +0000] "GET /robots.txt HTTP/1.1" 404 397 "http://googledorking.cmnatic.co.uk/" "Moz
ndroid 7.0; Moto G (4)) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/88.0.4324.175 Mobile Safari/537.36 Chrome-Lighthouse"
ubuntu@ip-172-31-23-158:~$
```

"Grep" has searched through this file and has shown us any entries of what we've provided and that is contained within this log file for the IP.

Linux operators are a fantastic way to power up your knowledge of working with Linux. There are a few important operators that are worth noting. We'll cover the basics and break them down accordingly to bite-sized chunks.

At an overview, I'm going to be showcasing the following operators:

Symbol / Operator	Description
&	This operator allows you to run commands in the background of your terminal.
&&	This operator allows you to combine multiple commands together in one line of your terminal.
>	This operator is a redirector - meaning that we can take the output from a command (such as using cat to output a file) and direct it elsewhere.
>>	This operator does the same function of the > operator but appends the output rather than replacing (meaning nothing is overwritten).

Let's cover these in a bit more detail.

Operator "&"

This operator allows us to execute commands in the background. For example, let's say we want to copy a large file. This will obviously take quite a long time and will leave us unable to do anything else until the file successfully copies.

The "&" shell operator allows us to execute a command and have it run in the background (such as this file copy) allowing us to do other things!

Operator "&&"

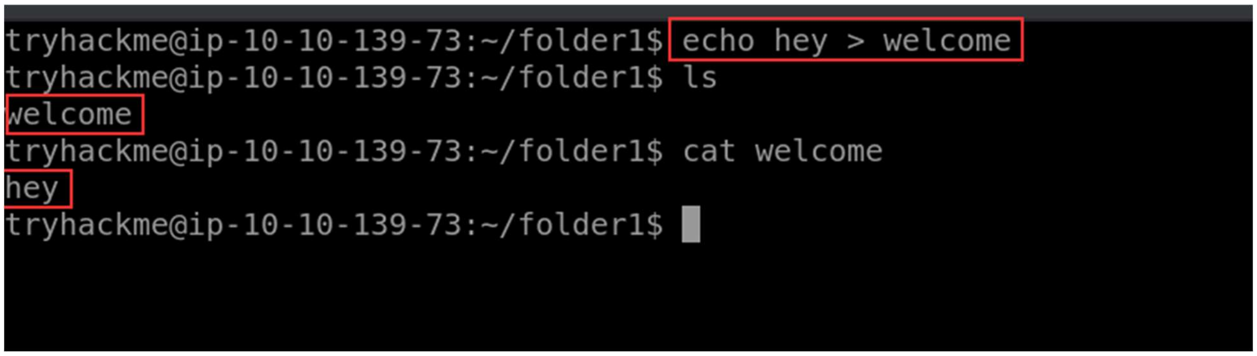
This shell operator is a bit misleading in the sense of how familiar is to its partner "&". Unlike the "&" operator, we can use "&&" to make a list of commands to run for example `command1 && command2`. However, it's worth noting that `command2` will only run if `command1` was successful.

Operator ">"

This operator is what's known as an output redirector. What this essentially means is that we take the output from a command we run and send that output to somewhere else.

A great example of this is redirecting the output of the echo command that we learned in Task 4. Of course, running something, such as echo howdy will return "howdy" back to our terminal — that isn't super useful. What we can do instead, is redirect "howdy" to something such as a new file!

Let's say we wanted to create a file named "welcome" with the message "hey". We can run echo hey > welcome where we want the file created with the contents "hey" like so:



```
tryhackme@ip-10-10-139-73:~/folder1$ echo hey > welcome
tryhackme@ip-10-10-139-73:~/folder1$ ls
welcome
tryhackme@ip-10-10-139-73:~/folder1$ cat welcome
hey
tryhackme@ip-10-10-139-73:~/folder1$ 
```

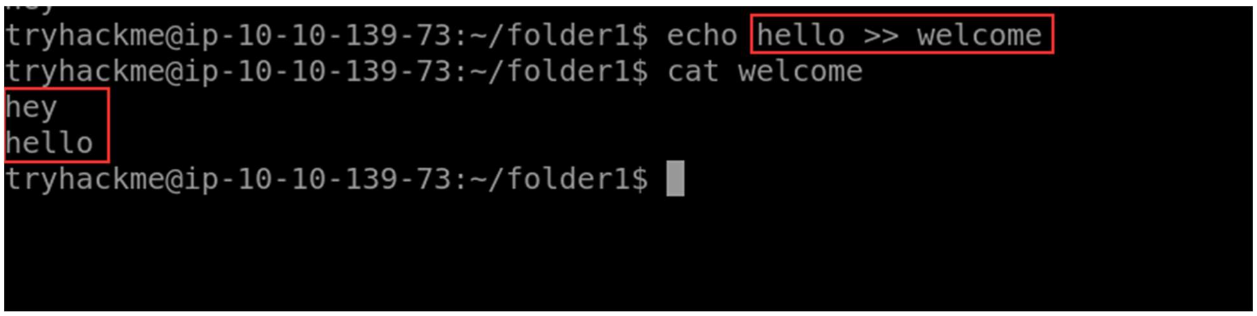
Note: If the file i.e. "welcome" already exists, the contents will be overwritten!

Operator ">>"

This operator is also an output redirector like in the previous operator (>) we discussed. However, what makes this operator different is that rather than overwriting any contents within a file, for example, it instead just puts the output at the end.

Following on with our previous example where we have the file "welcome" that has the contents of "hey". If were to use echo to add "hello" to the file using the > operator, the file will now only have "hello" and not "hey".

The >> operator allows to append the output to the bottom of the file — rather than replacing the contents like so:



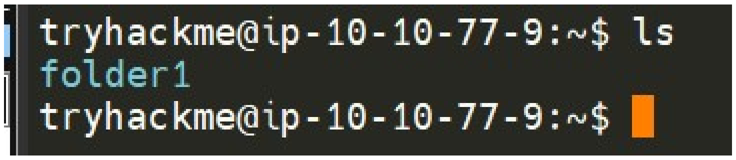
```
tryhackme@ip-10-10-139-73:~/folder1$ echo hello >> welcome
tryhackme@ip-10-10-139-73:~/folder1$ cat welcome
hey
hello
tryhackme@ip-10-10-139-73:~/folder1$
```

A majority of commands allow for arguments to be provided. These arguments are identified by a hyphen and a certain keyword known as flags or switches.

We'll later discuss how we can identify what commands allow for arguments to be provided and understanding what these do exactly.

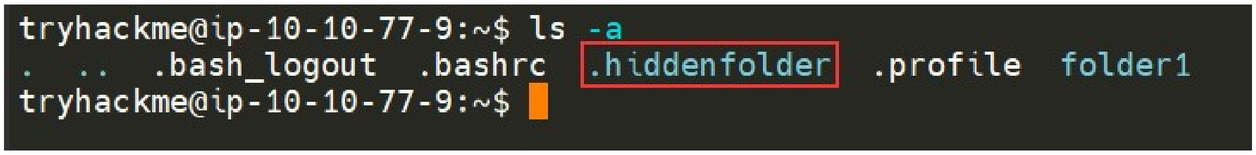
When using a command, unless otherwise specified, it will perform its default behaviour. For example, ls lists the contents of the working directory. However, hidden files are not shown. We can use flags and switches to extend the behaviour of commands.

Using our ls example, ls informs us that there is only one folder named "folder1" as highlighted in the screenshot below. Note that the contents in the screenshots below are only examples and are not those of those the instance that you deploy in this room.



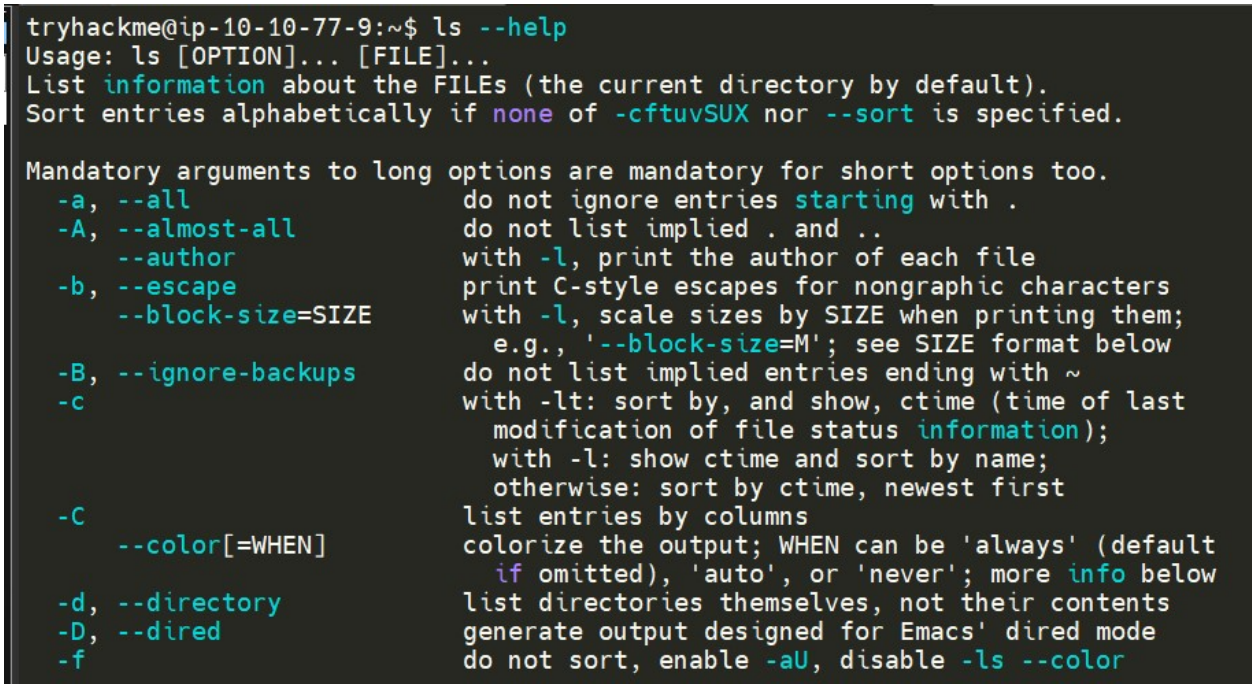
```
tryhackme@ip-10-10-77-9:~$ ls
folder1
tryhackme@ip-10-10-77-9:~$
```

However, after using the -a argument (short for --all), we now suddenly have an output with a few more files and folders such as ".hiddenfolder". Files and folders with "." are hidden files.



```
tryhackme@ip-10-10-77-9:~$ ls -a
.  ..  .bash_logout  .bashrc  .hiddenfolder  .profile  folder1
tryhackme@ip-10-10-77-9:~$
```

Commands that accept these will also have a --help option. This option will list the possible options that the command accepts, provide a brief description and example of how to use it.



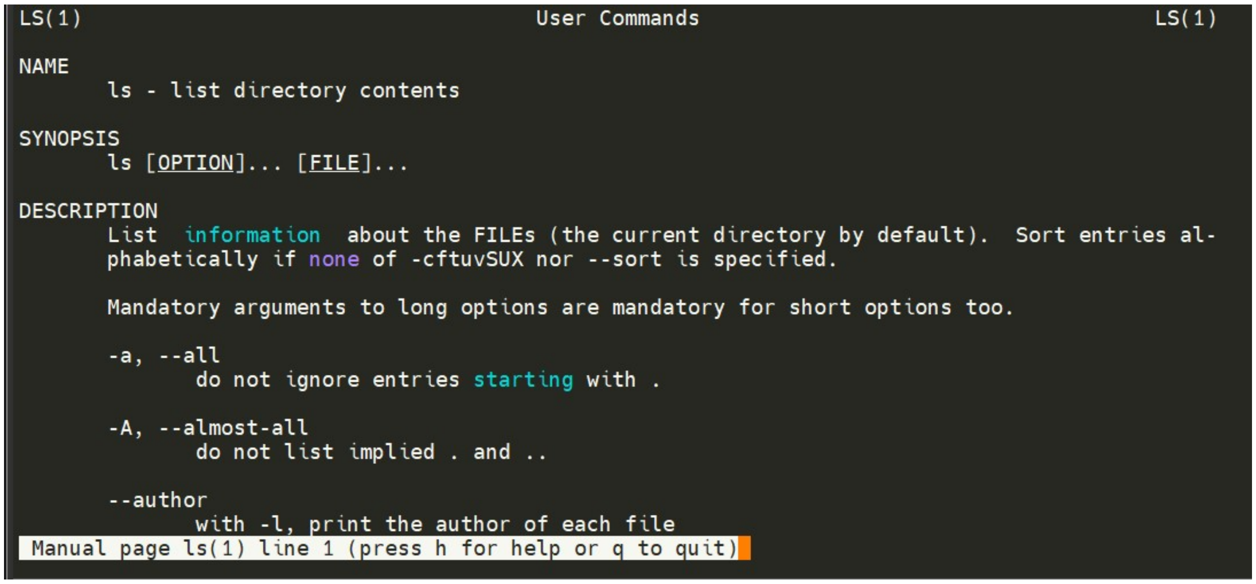
```
tryhackme@ip-10-10-77-9:~$ ls --help
Usage: ls [OPTION]... [FILE]...
List information about the FILEs (the current directory by default).
Sort entries alphabetically if none of -cftuvSUX nor --sort is specified.

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               with -l, print the author of each file
  -b, --escape               print C-style escapes for nongraphic characters
      --block-size=SIZE      with -l, scale sizes by SIZE when printing them;
                               e.g., '--block-size=M'; see SIZE format below
  -B, --ignore-backups       do not list implied entries ending with ~
  -c                         with -lt: sort by, and show, ctime (time of last
                               modification of file status information);
                               with -l: show ctime and sort by name;
                               otherwise: sort by ctime, newest first
  -C                         list entries by columns
      --color[=WHEN]         colorize the output; WHEN can be 'always' (default
                               if omitted), 'auto', or 'never'; more info below
  -d, --directory            list directories themselves, not their contents
  -D, --dired                generate output designed for Emacs' dired mode
  -f                         do not sort, enable -aU, disable -ls --color
```

This option is, in fact, a formatted output of what is called the man page (short for manual), which contains documentation for Linux commands and applications.

The Man(ual) Page

The manual pages are a great source of information for both system commands and applications available on both a Linux machine, which is accessible on the machine itself and online.

To access this documentation, we can use the man command and then provide the command we want to read the documentation for. Using our ls example, we would use man ls to view the manual pages for ls like so:



```
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 al-
       phabetically 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
              with -l, print the author of each file
 Manual page ls(1) line 1 (press h for help or q to quit)
```

We covered some of the most fundamental commands when interacting with the filesystem on the Linux machine. For example, we covered how to list and find the contents of folders using ls and find and navigating the filesystem using cd.

In this task, we're going to learn some more commands for interacting with the filesystem to allow us to:

- create files and folders
- move files and folders
- delete files and folders

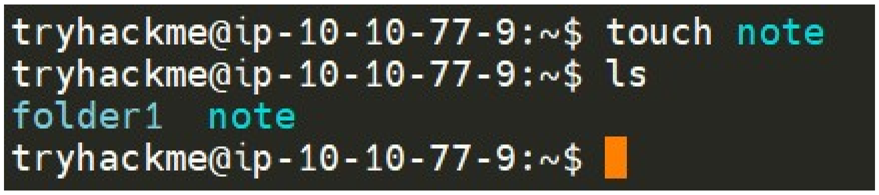
More specifically, the following commands:

Command	Full Name	Purpose
touch	touch	Create file
mkdir	make directory	Create a folder
cp	copy	Copy a file or folder
mv	move	Move a file or folder
rm	remove	Remove a file or folder
file	file	Determine the type of a file

Protip: Similarly to using cat, we can provide full file paths, i.e. directory1/directory2/note for all of these commands

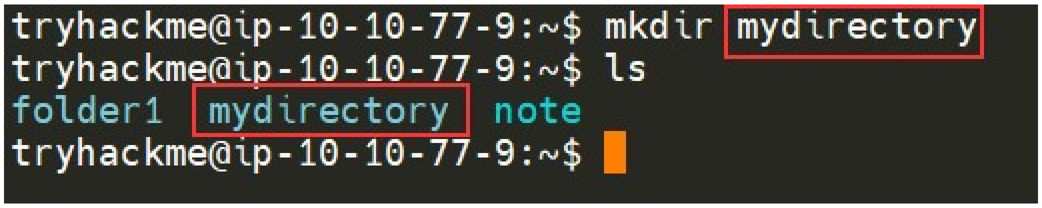
Creating Files and Folders (touch, mkdir)

Creating files and folders on Linux is a simple process. First, we'll cover creating a file. The touch command takes exactly one argument -- the name we want to give the file we create. For example, we can create the file "note" by using touch note. It's worth noting that touch simply creates a blank file. You would need to use commands like echo or text editors such as nano to add content to the blank file.



```
tryhackme@ip-10-10-77-9:~$ touch note
tryhackme@ip-10-10-77-9:~$ ls
folder1  note
tryhackme@ip-10-10-77-9:~$ 
```

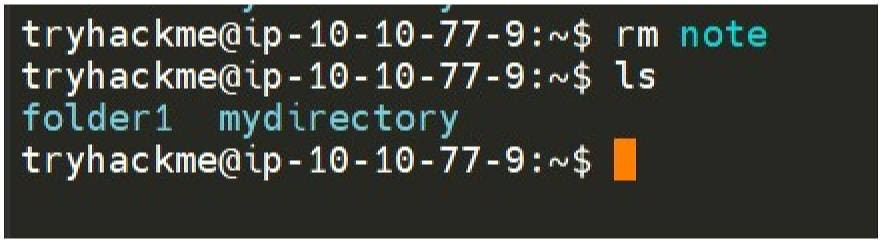
This is a similar process for making a folder, which just involves using the mkdir command and again providing the name that we want to assign to the directory. For example, creating the directory "mydirectory" using mkdir mydirectory.



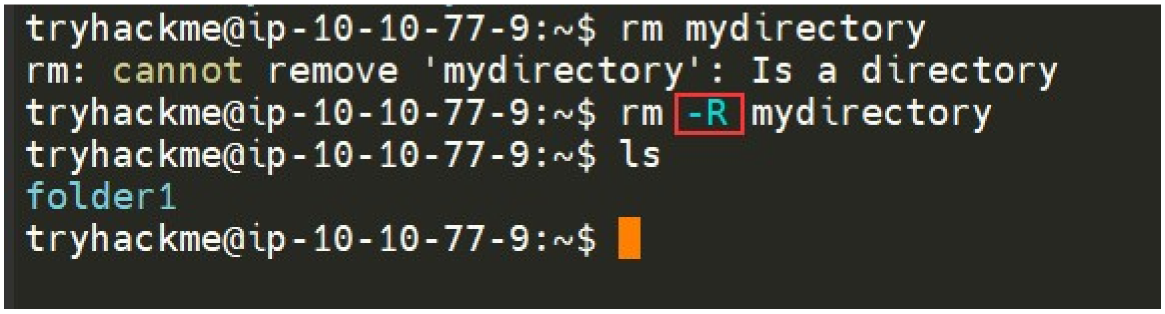
```
tryhackme@ip-10-10-77-9:~$ mkdir mydirectory
tryhackme@ip-10-10-77-9:~$ ls
folder1  mydirectory  note
tryhackme@ip-10-10-77-9:~$ 
```

Removing Files and Folders (rm)

rm is extraordinary out of the commands that we've covered so far. You can simply remove files by using rm. However, you need to provide the -R switch alongside the name of the directory you wish to remove.



```
tryhackme@ip-10-10-77-9:~$ rm note
tryhackme@ip-10-10-77-9:~$ ls
folder1  mydirectory
tryhackme@ip-10-10-77-9:~$ 
```



```
tryhackme@ip-10-10-77-9:~$ rm mydirectory
rm: cannot remove 'mydirectory': Is a directory
tryhackme@ip-10-10-77-9:~$ rm -R mydirectory
tryhackme@ip-10-10-77-9:~$ ls
folder1
tryhackme@ip-10-10-77-9:~$
```

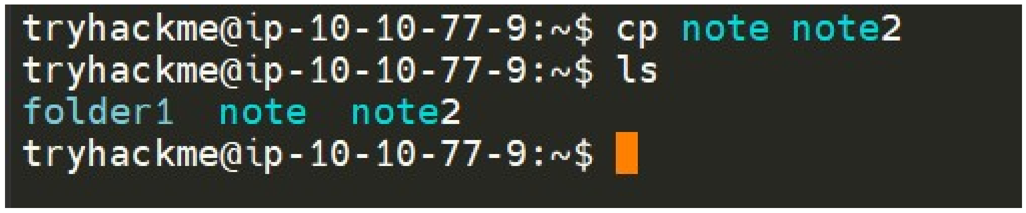
Copying and Moving Files and Folders (cp, mv)

Copying and moving files is an important functionality on a Linux machine. Starting with cp, this command takes two arguments:

1. the name of the existing file

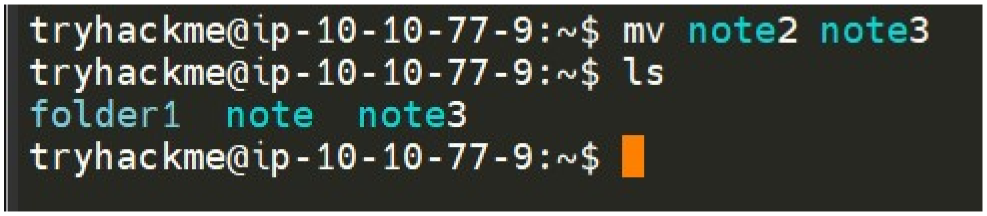
2. the name we wish to assign to the new file when copying

cp copies the entire contents of the existing file into the new file. In the screenshot below, we are copying "note" to "note2".



```
tryhackme@ip-10-10-77-9:~$ cp note note2
tryhackme@ip-10-10-77-9:~$ ls
folder1  note  note2
tryhackme@ip-10-10-77-9:~$
```

Moving a file takes two arguments, just like the cp command. However, rather than copying and/or creating a new file, mv will merge or modify the second file that we provide as an argument. Not only can you use mv to move a file to a new folder, but you can also use mv to rename a file or folder. For example, in the screenshot below, we are renaming the file "note2" to be named "note3". "note3" will now have the contents of "note2".

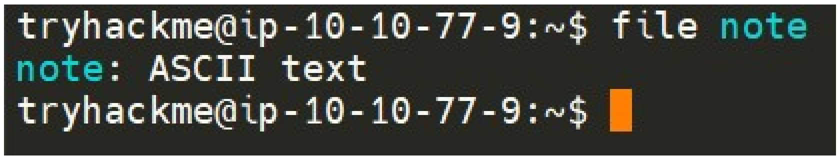


```
tryhackme@ip-10-10-77-9:~$ mv note2 note3
tryhackme@ip-10-10-77-9:~$ ls
folder1  note  note3
tryhackme@ip-10-10-77-9:~$
```

Determining File Type

What is often misleading and often catches people out is making presumptions from files as to what their purpose or contents may be. Files usually have what's known as an extension to make this easier. For example, text files usually have an extension of ".txt". But this is not necessary.

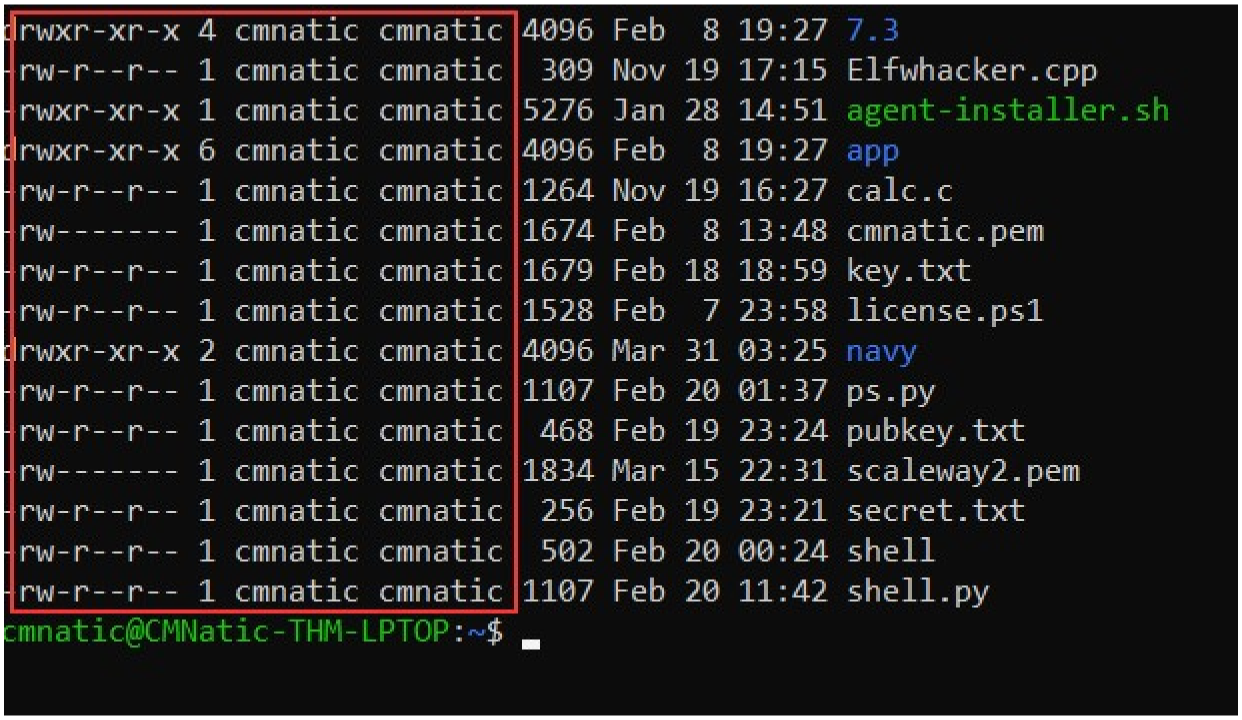
So far, the files we have used in our examples haven't had an extension. Without knowing the context of why the file is there -- we don't really know its purpose. Enter the `file` command. This command takes one argument. For example, we'll use `file` to confirm whether or not the "note" file in our examples is indeed a text file, like so `file note`.



```
tryhackme@ip-10-10-77-9:~$ file note
note: ASCII text
tryhackme@ip-10-10-77-9:~$
```

As you would have already found out by now, certain users cannot access certain files or folders. We've previously explored some commands that can be used to determine what access we have and where it leads us.

In our previous tasks, we learned how to extend the use of commands through flags and switches. Take, for example, the `ls` command, which lists the contents of the current directory. When using the `-l` switch, we can see ten columns such as in the screenshot below. However, we're only interested in the first three columns:

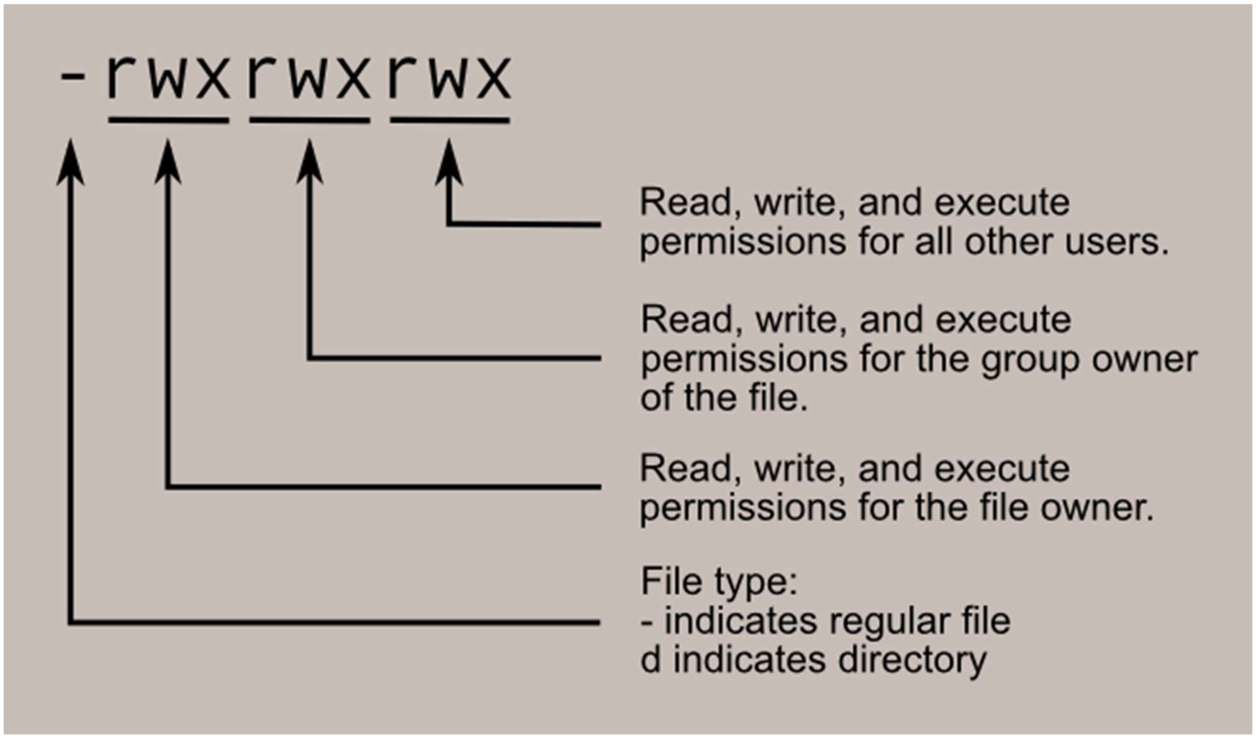


```
drwxr-xr-x 4 cmnatic cmnatic 4096 Feb  8 19:27 7.3
-rw-r--r-- 1 cmnatic cmnatic  309 Nov 19 17:15 Elfwhacker.cpp
-rwxr-xr-x 1 cmnatic cmnatic 5276 Jan 28 14:51 agent-installer.sh
drwxr-xr-x 6 cmnatic cmnatic 4096 Feb  8 19:27 app
-rw-r--r-- 1 cmnatic cmnatic 1264 Nov 19 16:27 calc.c
-rw------- 1 cmnatic cmnatic 1674 Feb  8 13:48 cmnatic.pem
-rw-r--r-- 1 cmnatic cmnatic 1679 Feb 18 18:59 key.txt
-rw-r--r-- 1 cmnatic cmnatic 1528 Feb  7 23:58 license.ps1
drwxr-xr-x 2 cmnatic cmnatic 4096 Mar 31 03:25 navy
-rw-r--r-- 1 cmnatic cmnatic 1107 Feb 20 01:37 ps.py
-rw-r--r-- 1 cmnatic cmnatic  468 Feb 19 23:24 pubkey.txt
-rw------- 1 cmnatic cmnatic 1834 Mar 15 22:31 scaleway2.pem
-rw-r--r-- 1 cmnatic cmnatic  256 Feb 19 23:21 secret.txt
-rw-r--r-- 1 cmnatic cmnatic  502 Feb 20 00:24 shell
-rw-r--r-- 1 cmnatic cmnatic 1107 Feb 20 11:42 shell.py
cmnatic@CMNatic-THM-LPTOP:~$
```

Although intimidating, these three columns are very important in determining certain characteristics of a file or folder and whether or not we have access to it. A file or folder can have a couple of characteristics that determine both what it is that and who we can do with it as -- such as the following:

- Read
- Write
- Execute

The diagram below is a great representation of how these permissions can be translated.


-rwxrwxrwx
Read, write, and execute permissions for all other users.
Read, write, and execute permissions for the group owner of the file.
Read, write, and execute permissions for the file owner.
File type:
- indicates regular file
d indicates directory

Let's use the "cmnatic.pem" file in our initial screenshot at the top of this task. It has the "-" indicator highlighting that it is a file and then "rw" followed after. This means that only the owner of the file can read and write to this"cmnatic.pem" file but cannot execute it.

Briefly: The Differences Between Users & Groups

The great thing about Linux is that permissions can be so granular, that whilst a user technically owns a file, if the permissions have been set, then a group of users can also have either the same or a different set of permissions to the exact same file without affecting the file owner itself.

Let's put this into a real-world context; the system user that runs a web server must have permissions to read and write files for an effective web application. However, companies such as web hosting companies will have to want to allow their customers to upload their own files for their website without being the webserver system user -- compromising the security of every other customer.

We'll learn the commands necessary to switch between users below.

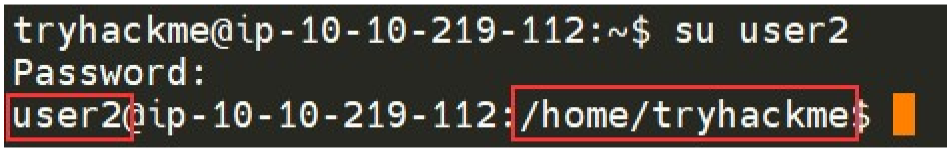
Switching Between Users

Switching between users on a Linux install is easy work thanks to the su command. Unless you are the root user (or using root permissions through sudo), then you are required to know two things to facilitate this transition of user accounts:

- The user we wish to switch to
- The user's password

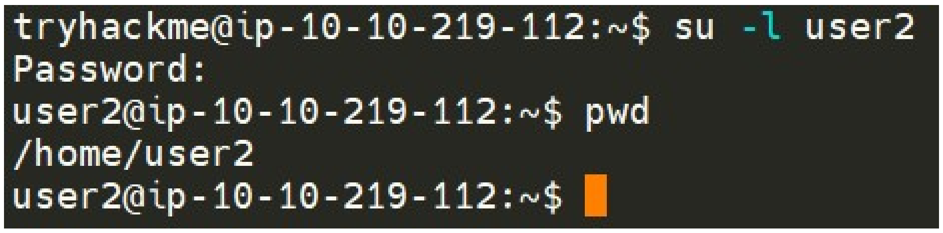
The su command takes a couple of switches that may be of relevance to you. For example, executing a command once you log in or specifying a specific shell to use. I encourage you to read the man page for su to find out more. However, I will cover the -l or --login switch.

Simply, by providing the -l switch to su, we start a shell that is much more similar to the actual user logging into the system - we inherit a lot more properties of the new user, i.e., environment variables and the likes.



```
tryhackme@ip-10-10-219-112:~$ su user2
Password:
user2@ip-10-10-219-112:/home/tryhackme$
```

For example, when using su to switch to "user2", our new session drops us into our previous user's home directory.



```
tryhackme@ip-10-10-219-112:~$ su -l user2
Password:
user2@ip-10-10-219-112:~$ pwd
/home/user2
user2@ip-10-10-219-112:~$
```

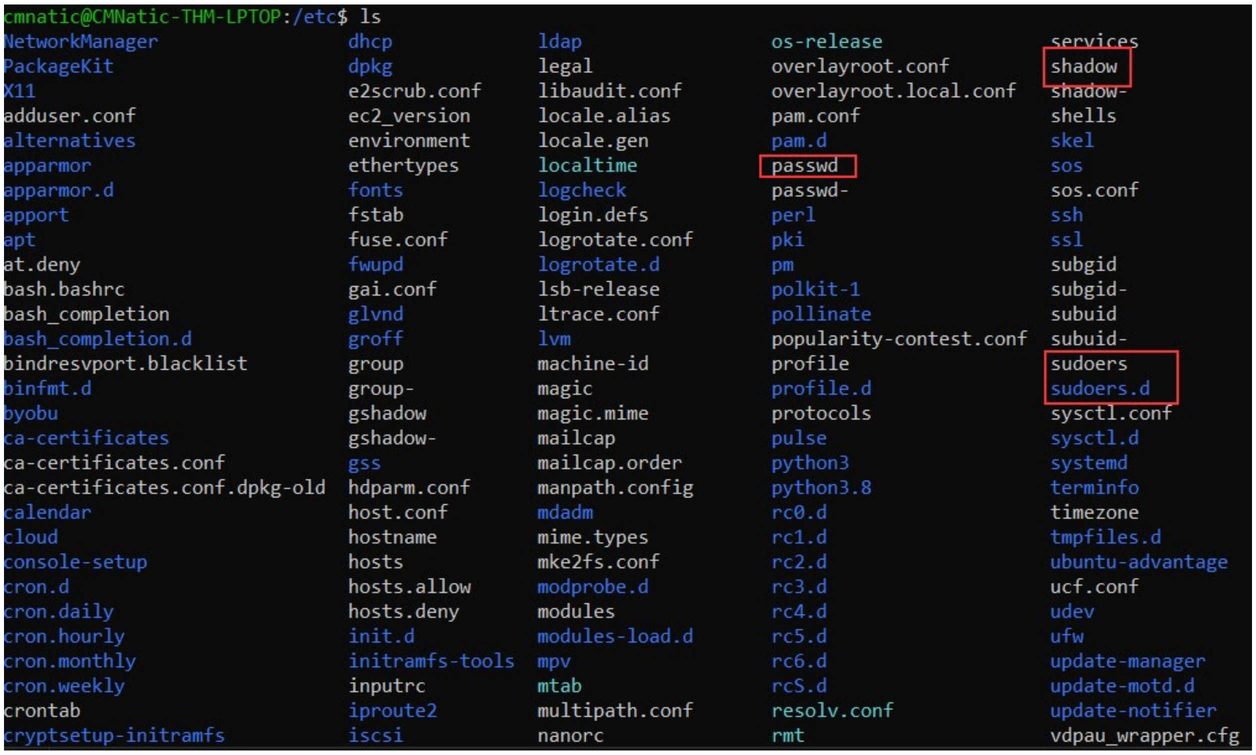
Where now, after using -l, our new session has dropped us into the home directory of "user" automatically.

/etc

This root directory is one of the most important root directories on your system. The etc folder (short for etcetera) is a commonplace location to store system files that are used by your operating system.

For example, the sudoers file highlighted in the screenshot below contains a list of the users & groups that have permission to run sudo or a set of commands as the root user.

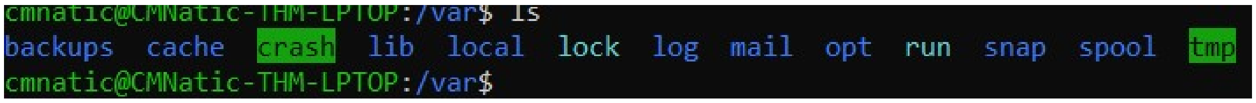
Also highlighted below are the "**passwd**" and "**shadow**" files. These two files are special for Linux as they show how your system stores the passwords for each user in encrypted formatting called sha512.



```
cmnatic@CMNatic-THM-LPTOP:/etc$ ls
NetworkManager                 dhcp             ldap                 os-release               services
PackageKit                     dpkg             legal                overlayroot.conf         shadow
X11                            e2scrub.conf     libaudit.conf        overlayroot.local.conf   shadow-
adduser.conf                   ec2_version      locale.alias         pam.conf                 shells
alternatives                   environment      locale.gen           pam.d                    skel
apparmor                       ethertypes       localtime            passwd                   sos
apparmor.d                     fonts            logcheck             passwd-                  sos.conf
apport                         fstab            login.defs           perl                     ssh
apt                            fuse.conf        logrotate.conf       pki                      ssl
at.deny                        fwupd            logrotate.d          pm                       subgid
bash.bashrc                    gai.conf         lsb-release          polkit-1                 subgid-
bash_completion                glvnd            ltrace.conf          pollinate                subuid
bash_completion.d              groff            lvm                  popularity-contest.conf  subuid-
bindresvport.blacklist         group            machine-id           profile                  sudoers
binfmt.d                       group-           magic                profile.d                sudoers.d
byobu                          gshadow          magic.mime           protocols                sysctl.conf
ca-certificates                gshadow-         mailcap              pulse                    sysctl.d
ca-certificates.conf           gss              mailcap.order        python3                  systemd
ca-certificates.conf.dpkg-old  hdparm.conf      manpath.config       python3.8                terminfo
calendar                       host.conf        mdadm                rc0.d                    timezone
cloud                          hostname         mime.types           rc1.d                    tmpfiles.d
console-setup                  hosts            mke2fs.conf          rc2.d                    ubuntu-advantage
cron.d                         hosts.allow      modprobe.d           rc3.d                    ucf.conf
cron.daily                     hosts.deny       modules              rc4.d                    udev
cron.hourly                    init.d           modules-load.d       rc5.d                    ufw
cron.monthly                   initramfs-tools  mpv                  rc6.d                    update-manager
cron.weekly                    inputrc          mtab                 rcS.d                    update-motd.d
crontab                        iproute2         multipath.conf       resolv.conf              update-notifier
cryptsetup-initramfs           iscsi            nanorc               rmt                      vdpau_wrapper.cfg
```

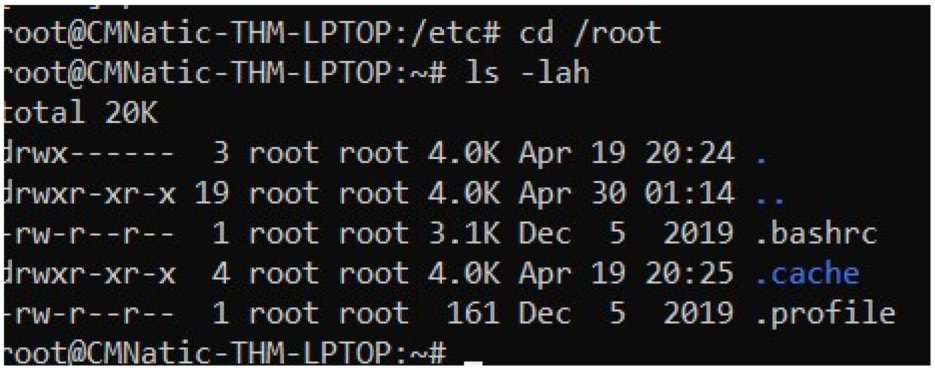
/var

The "/var" directory, with "var" being short for variable data, is one of the main root folders found on a Linux install. This folder stores data that is frequently accessed or written by services or applications running on the system. For example, log files from running services and applications are written here (**/var/log**), or other data that is not necessarily associated with a specific user (i.e., databases and the like).


```
cmnatic@CMNatic-THM-LPTOP:/var$ ls
backups  cache  crash  lib  local  lock  log  mail  opt  run  snap  spool  tmp
cmnatic@CMNatic-THM-LPTOP:/var$
```

/root

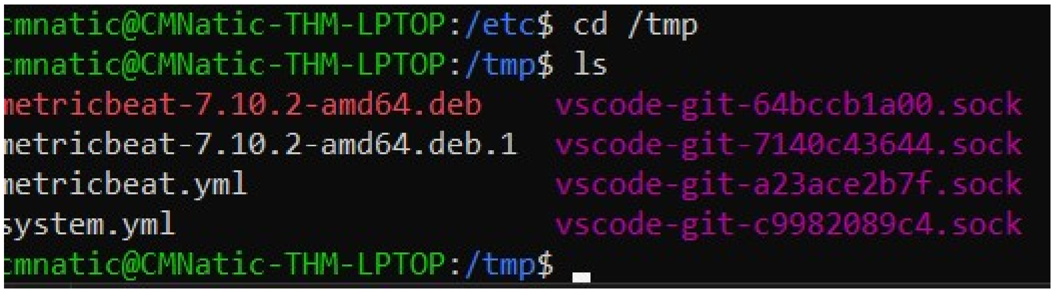
Unlike the **/home** directory, the **/root** folder is actually the home for the "root" system user. There isn't anything more to this folder other than just understanding that this is the home directory for the "root" user. But, it is worth a mention as the logical presumption is that this user would have their data in a directory such as "**/home/root**" by default.


```
oot@CMNatic-THM-LPTOP:/etc# cd /root
oot@CMNatic-THM-LPTOP:~# ls -lah
otal 20K
rwx------  3 root root 4.0K Apr 19 20:24 .
rwxr-xr-x 19 root root 4.0K Apr 30 01:14 ..
rw-r--r--  1 root root 3.1K Dec  5  2019 .bashrc
rwxr-xr-x  4 root root 4.0K Apr 19 20:25 .cache
rw-r--r--  1 root root  161 Dec  5  2019 .profile
oot@CMNatic-THM-LPTOP:~#
```

/tmp

This is a unique root directory found on a Linux install. Short for "**temporary**", the /tmp directory is volatile and is used to store data that is only needed to be accessed once or twice. Similar to the memory on your computer, once the computer is restarted, the contents of this folder are cleared out.

What's useful for us in pentesting is that any user can write to this folder by default. Meaning once we have access to a machine, it serves as a good place to store things like our enumeration scripts.

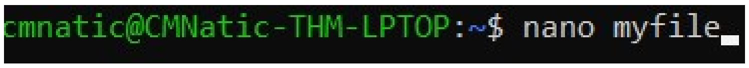

```
mnatic@CMNatic-THM-LPTOP:/etc$ cd /tmp
mnatic@CMNatic-THM-LPTOP:/tmp$ ls
etricbeat-7.10.2-amd64.deb     vscode-git-64bccb1a00.sock
etricbeat-7.10.2-amd64.deb.1   vscode-git-7140c43644.sock
etricbeat.yml                  vscode-git-a23ace2b7f.sock
ystem.yml                      vscode-git-c9982089c4.sock
mnatic@CMNatic-THM-LPTOP:/tmp$
```

Introducing terminal text editors

There are a few options that you can use, all with a variety of friendliness and utility. This task is going to introduce you to nano but also show you an alternative named VIM

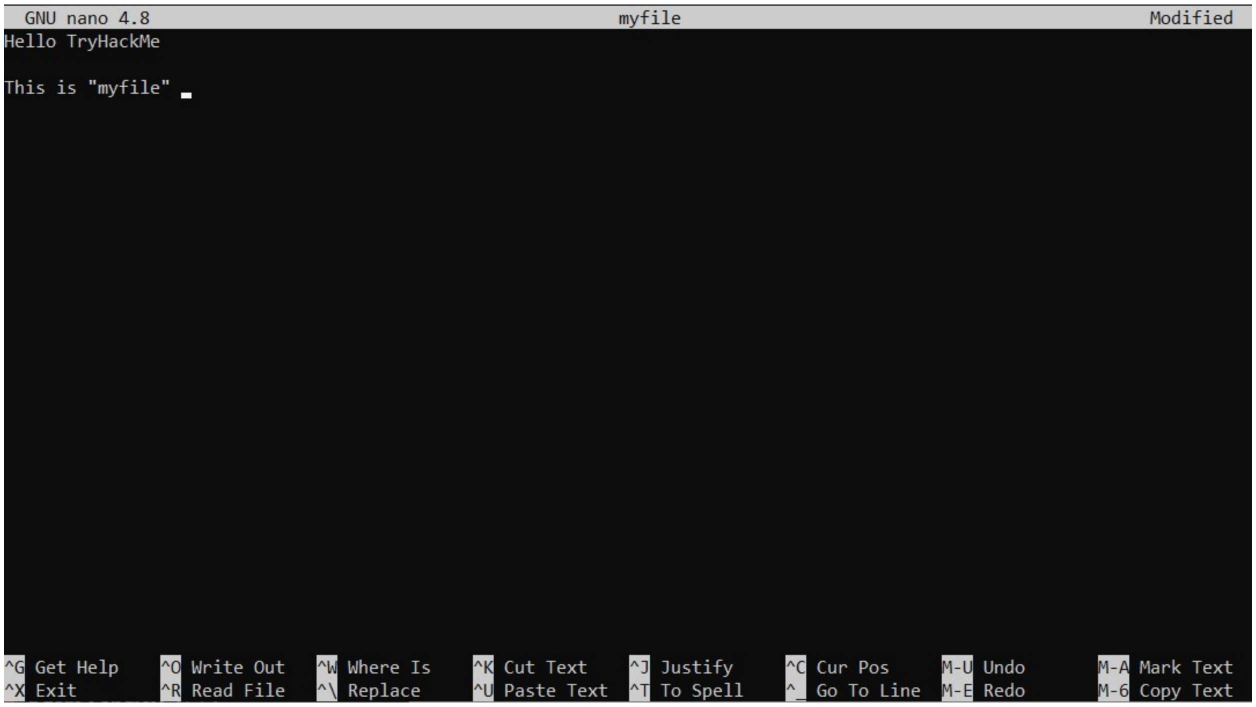
Nano

It is easy to get started with Nano! To create or edit a file using nano, we simply use nano filename -- replacing "filename" with the name of the file you wish to edit.



```
cmnatic@CMNatic-THM-LPTOP:~$ nano myfile
```

Once we press enter to execute the command, nano will launch! Where we can just begin to start entering or modifying our text. You can navigate each line using the "up" and "down" arrow keys or start a new line using the "Enter" key on your keyboard.



```
  GNU nano 4.8                              myfile                              Modified
Hello TryHackMe

This is "myfile"

^G Get Help   ^O Write Out  ^W Where Is   ^K Cut Text   ^J Justify    ^C Cur Pos    M-U Undo      M-A Mark Text
^X Exit       ^R Read File  ^\ Replace    ^U Paste Text ^T To Spell   ^_ Go To Line M-E Redo      M-6 Copy Text
```

Nano has a few features that are easy to remember & covers the most general things you would want out of a text editor, including:

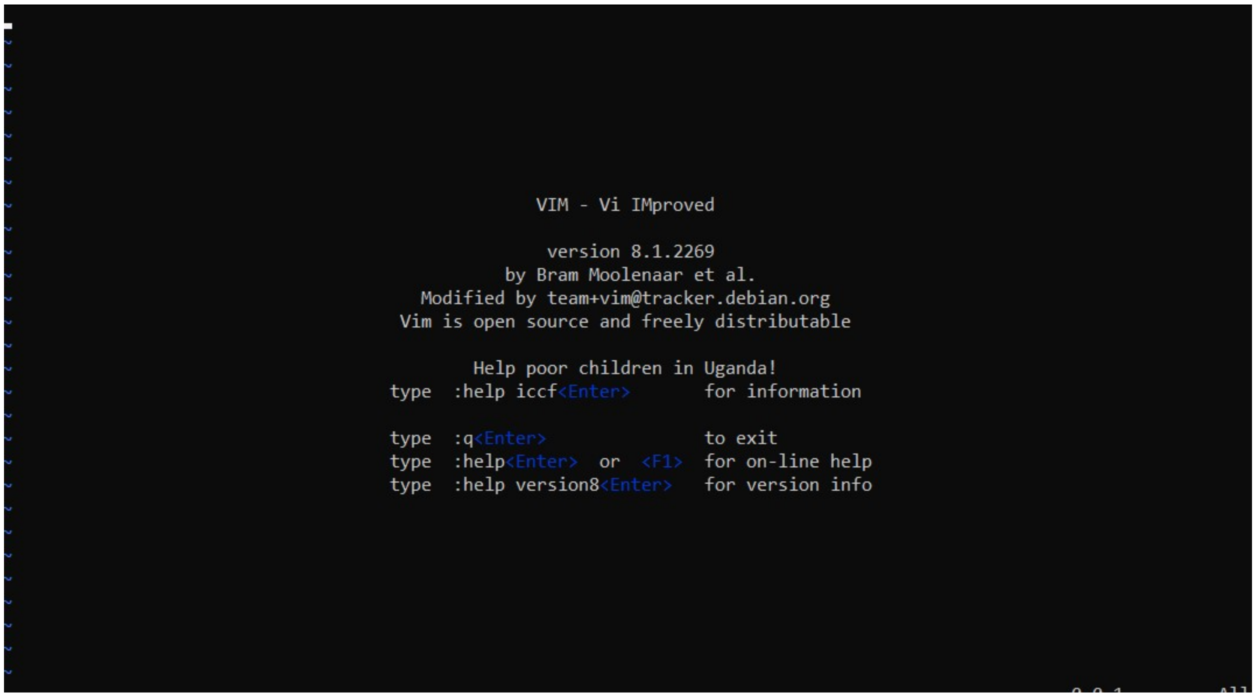
- Searching for text
- Copying and Pasting
- Jumping to a line number

- Finding out what line number you are on

You can use these features of nano by pressing the "**Ctrl**" key (which is represented as an ^ on Linux) and a corresponding letter. For example, to exit, we would want to press "**Ctrl**" and "**X**" to exit Nano.

VIM

VIM is a much more advanced text editor. Whilst you're not expected to know all advanced features, it's helpful to mention it for powering up your Linux skills.



```
                         VIM - Vi IMproved

                          version 8.1.2269
                      by Bram Moolenaar et al.
              Modified by team+vim@tracker.debian.org
             Vim is open source and freely distributable

                       Help poor children in Uganda!
            type  :help iccf<Enter>       for information

            type  :q<Enter>               to exit
            type  :help<Enter>  or  <F1>  for on-line help
            type  :help version8<Enter>   for version info
```

Some of VIM's benefits, albeit taking a much longer time to become familiar with, includes:

- Customisable - you can modify the keyboard shortcuts to be of your choosing
- Syntax Highlighting - this is useful if you are writing or maintaining code, making it a popular choice for software developers
- VIM works on all terminals where nano may not be installed
- There are a lot of resources such as cheatsheets, tutorials, and the sorts available to you use.

Downloading Files

A pretty fundamental feature of computing is the ability to transfer files. For example, you may want to download a program, a script, or even a picture. Thankfully for us, there are multiple ways in which we can retrieve these files.

We're going to cover the use of wget . This command allows us to download files from the web via HTTP -- as if you were accessing the file in your browser. We simply need to provide the address of the resource that we wish to download. For example, if I wanted to download a file named "myfile.txt" onto my machine, assuming I knew the web address it -- it would look something like this:

```
wget https://assets.com/myfile.txt
```

Transferring Files From Your Host - SCP (SSH)

Secure copy, or SCP, is just that -- a means of securely copying files. Unlike the regular cp command, this command allows you to transfer files between two computers using the SSH protocol to provide both authentication and encryption.

Working on a model of SOURCE and DESTINATION, SCP allows you to:

- Copy files & directories from your current system to a remote system
- Copy files & directories from a remote system to your current system

Provided that we know usernames and passwords for a user on your current system and a user on the remote system. For example, let's copy an example file from our machine to a remote machine, which I have neatly laid out in the table 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

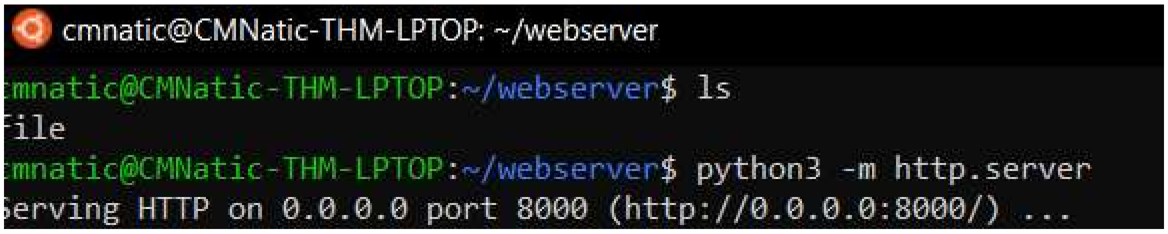
Variable	Value
IP address of the remote system	192.168.1.30
User on the remote system	ubuntu
Name of the file on the remote system	documents.txt
Name that we wish to store the file as on our system	notes.txt

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

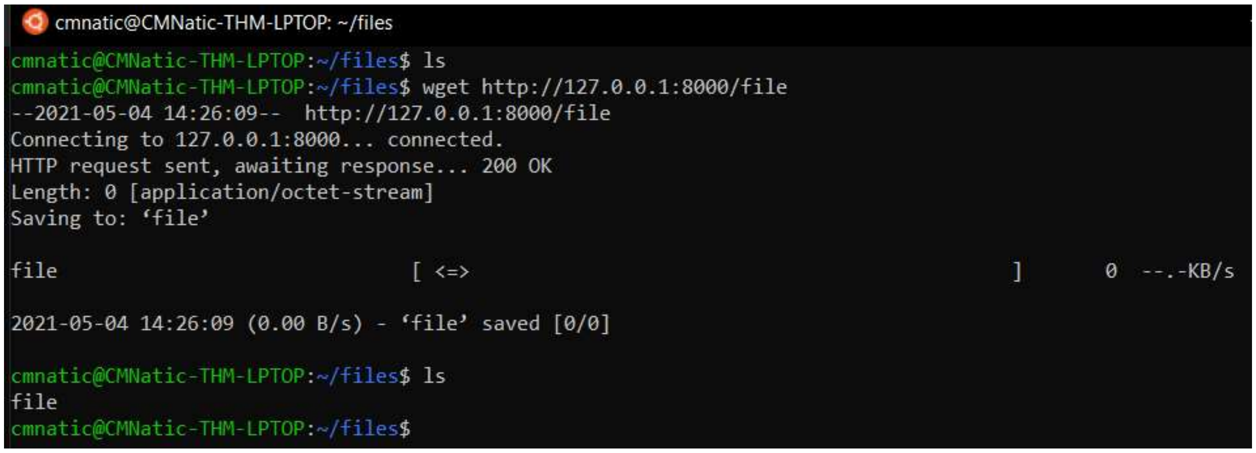
Ubuntu machines come pre-packaged with python3. Python helpfully provides a lightweight and easy-to-use module called "HTTPServer". This module turns your computer into a quick and easy web server that you can use to serve your own files, where they can then be downloaded by another computing using commands such as curl and wget.

Python3's "HTTPServer" will serve the files in the directory that you run the command, but this can be changed by providing options that can be found in the manual pages. Simply, all we need to do is run python3 -m http.server to start the module! In the screenshot below, we are serving from a directory called "webserver", which has a single named "file".



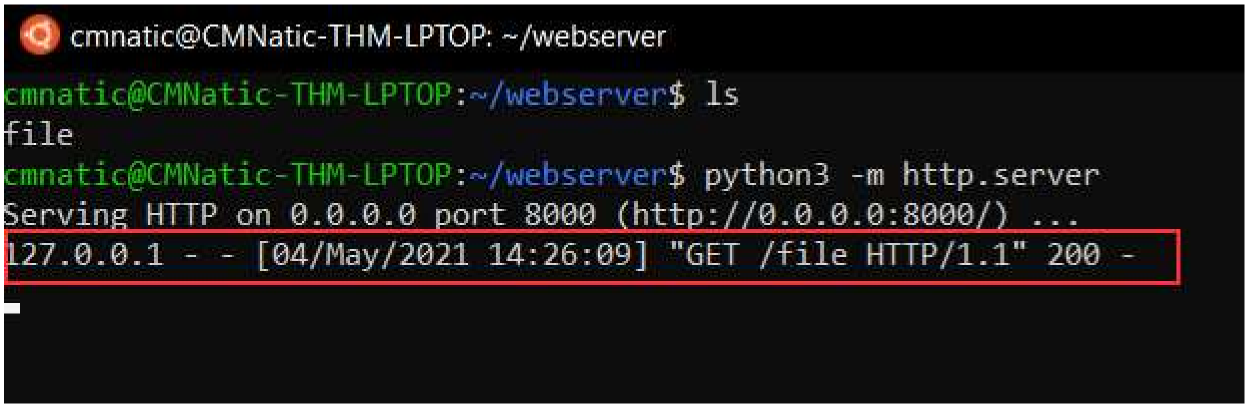
```
cmnatic@CMNatic-THM-LPTOP: ~/webserver
mnatic@CMNatic-THM-LPTOP:~/webserver$ ls
ile
mnatic@CMNatic-THM-LPTOP:~/webserver$ python3 -m http.server
erving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
```

Now, let's use wget to download the file using the computer's IP address and the name of the file. One flaw with this module is that you have no way of indexing, so you must know the exact name and location of the file that you wish to use. This is why I prefer to use Updog. What's Updog? A more advanced yet lightweight webserver. But for now, let's stick to using Python's "HTTP Server".



```
cmnatic@CMNatic-THM-LPTOP: ~/files
cmnatic@CMNatic-THM-LPTOP:~/files$ ls
cmnatic@CMNatic-THM-LPTOP:~/files$ wget http://127.0.0.1:8000/file
--2021-05-04 14:26:09--  http://127.0.0.1:8000/file
Connecting to 127.0.0.1:8000... connected.
HTTP request sent, awaiting response... 200 OK
Length: 0 [application/octet-stream]
Saving to: ‘file’

file                             [ <=>                                              ]       0  --.-KB/s

2021-05-04 14:26:09 (0.00 B/s) - ‘file’ saved [0/0]

cmnatic@CMNatic-THM-LPTOP:~/files$ ls
file
cmnatic@CMNatic-THM-LPTOP:~/files$
```

In the screenshot above, we can see that wget has successfully downloaded the file named "file" to our machine. This request is logged by SimpleHTTPServer much as any web server would, which I have captured in the screenshot below.

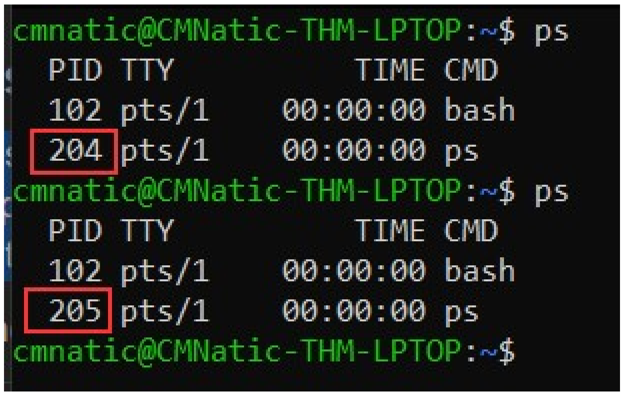


```
cmnatic@CMNatic-THM-LPTOP: ~/webserver
cmnatic@CMNatic-THM-LPTOP:~/webserver$ ls
file
cmnatic@CMNatic-THM-LPTOP:~/webserver$ python3 -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
127.0.0.1 - - [04/May/2021 14:26:09] "GET /file HTTP/1.1" 200 -
```

Processes are the programs that are running on your machine. They are managed by the kernel, where each process will have an ID associated with it, also known as its PID. The PID increments for the order In which the process starts. I.e. the 60th process will have a PID of 60.

Viewing Processes

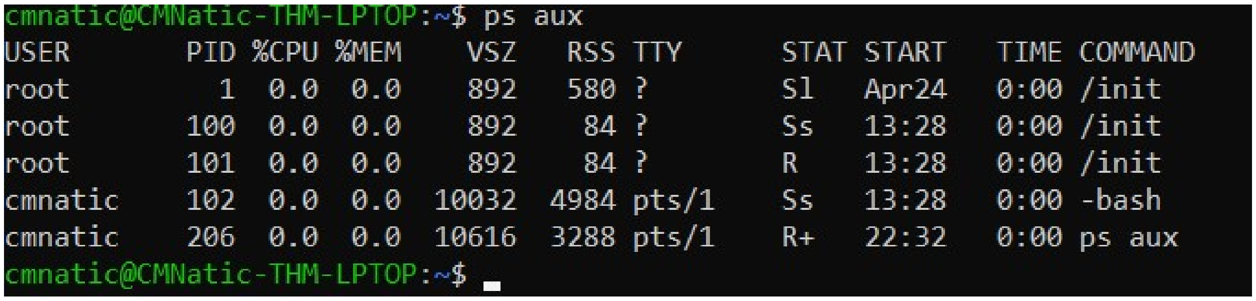
We can use the friendly ps command to provide a list of the running processes as our user's session and some additional information such as its status code, the session that is running it, how much usage time of the CPU it is using, and the name of the actual program or command that is being executed:



```
cmnatic@CMNatic-THM-LPTOP:~$ ps
  PID TTY          TIME CMD
  102 pts/1    00:00:00 bash
  204 pts/1    00:00:00 ps
cmnatic@CMNatic-THM-LPTOP:~$ ps
  PID TTY          TIME CMD
  102 pts/1    00:00:00 bash
  205 pts/1    00:00:00 ps
cmnatic@CMNatic-THM-LPTOP:~$
```

Note how in the screenshot above, the second process ps has a PID of 204, and then in the command below it, this is then incremented to 205.

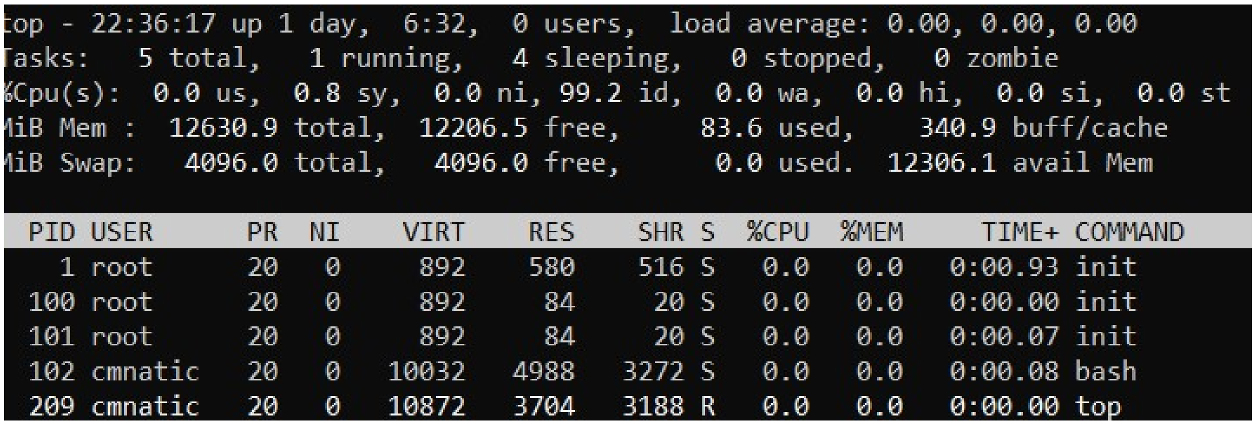
To see the processes run by other users and those that don't run from a session (i.e. system processes), we need to provide **aux** to the ps command like so: ps aux



```
cmnatic@CMNatic-THM-LPTOP:~$ ps aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0    892   580 ?        Sl   Apr24   0:00 /init
root       100  0.0  0.0    892    84 ?        Ss   13:28   0:00 /init
root       101  0.0  0.0    892    84 ?        R    13:28   0:00 /init
cmnatic    102  0.0  0.0  10032  4984 pts/1    Ss   13:28   0:00 -bash
cmnatic    206  0.0  0.0  10616  3288 pts/1    R+   22:32   0:00 ps aux
cmnatic@CMNatic-THM-LPTOP:~$ _
```

Note we can see a total of 5 processes -- note how we now have "root" and "cmnatic"

Another very useful command is the top command; top gives you real-time statistics about the processes running on your system instead of a one-time view. These statistics will refresh every 10 seconds, but will also refresh when you use the arrow keys to browse the various rows. Another great command to gain insight into your system is via the top command



```
top - 22:36:17 up 1 day,  6:32,  0 users,  load average: 0.00, 0.00, 0.00
Tasks:   5 total,   1 running,   4 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.0 us,  0.8 sy,  0.0 ni, 99.2 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :  12630.9 total,  12206.5 free,     83.6 used,    340.9 buff/cache
MiB Swap:   4096.0 total,   4096.0 free,      0.0 used.  12306.1 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
    1 root      20   0     892    580    516 S   0.0   0.0   0:00.93 init
  100 root      20   0     892     84     20 S   0.0   0.0   0:00.00 init
  101 root      20   0     892     84     20 S   0.0   0.0   0:00.07 init
  102 cmnatic   20   0   10032   4988   3272 S   0.0   0.0   0:00.08 bash
  209 cmnatic   20   0   10872   3704   3188 R   0.0   0.0   0:00.00 top
```

Managing Processes

You can send signals that terminate processes; there are a variety of types of signals that correlate to exactly how "cleanly" the process is dealt with by the kernel. To kill a command, we can use the appropriately named kill command and the associated PID that we wish to kill. i.e., to kill PID 1337, we'd use kill 1337.

Below are some of the signals that we can send to a process when it is killed:

- SIGTERM - Kill the process, but allow it to do some cleanup tasks beforehand
- SIGKILL - Kill the process - doesn't do any cleanup after the fact
- SIGSTOP - Stop/suspend a process

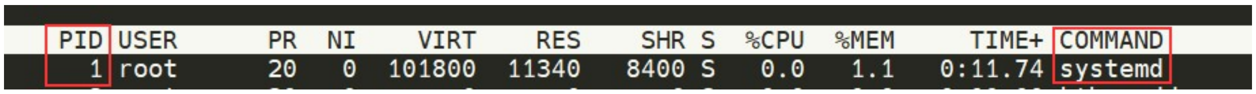
How do Processes Start?

Let's start off by talking about namespaces. The Operating System (OS) uses namespaces to ultimately split up the resources available on the computer to (such as CPU, RAM and priority) processes. Think of it as splitting your computer up into slices -- similar to a cake. Processes within that slice will have access to a certain amount of computing power, however, it will be a small portion of what is actually available to every process overall.

Namespaces are great for security as it is a way of isolating processes from another -- only those that are in the same namespace will be able to see each other.

We previously talked about how PID works, and this is where it comes into play. The process with an ID of 0 is a process that is started when the system boots. This process is the system's init on Ubuntu, such as **systemd**, which is used to provide a way of managing a user's processes and sits in between the operating system and the user.

For example, once a system boots and it initialises, **systemd** is one of the first processes that are started. Any program or piece of software that we want to start will start as what's known as a child process of **systemd**. This means that it is controlled by **systemd**, but will run as its own process (although sharing the resources from **systemd**) to make it easier for us to identify and the likes.



```
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
    1 root      20   0  101800  11340   8400 S   0.0   1.1   0:11.74 systemd
```

Getting Processes/Services to Start on Boot

Some applications can be started on the boot of the system that we own. For example, web servers, database servers or file transfer servers. This software is often critical and is often told to start during the boot-up of the system by administrators.

In this example, we're going to be telling the apache web server to be starting apache manually and then telling the system to launch apache2 on boot.

Enter the use of `systemctl` -- this command allows us to interact with the **systemd** process/daemon. Continuing on with our example, systemctl is an easy to use command that takes the following formatting: `systemctl [option] [service]`

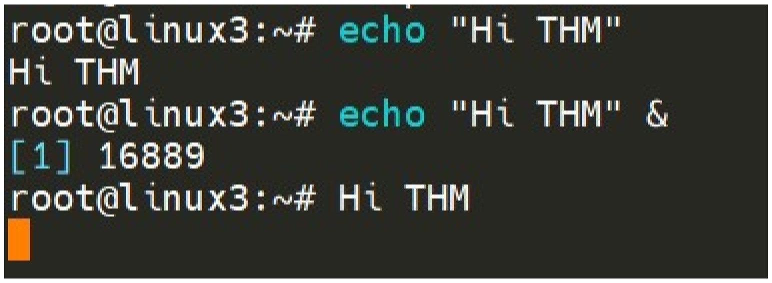
For example, to tell apache to start up, we'll use `systemctl start apache2`. Seems simple enough, right? Same with if we wanted to stop apache, we'd just replace the `[option]` with stop (instead of start like we provided)

We can do four options with `systemctl`:

- Start
- Stop
- Enable
- Disable

An Introduction to Backgrounding and Foregrounding in Linux

Processes can run in two states: In the background and in the foreground. For example, commands that you run in your terminal such as "echo" or things of that sort will run in the foreground of your terminal as it is the only command provided that hasn't been told to run in the background. "Echo" is a great example as the output of echo will return to you in the foreground, but wouldn't in the background - take the screenshot below, for example.

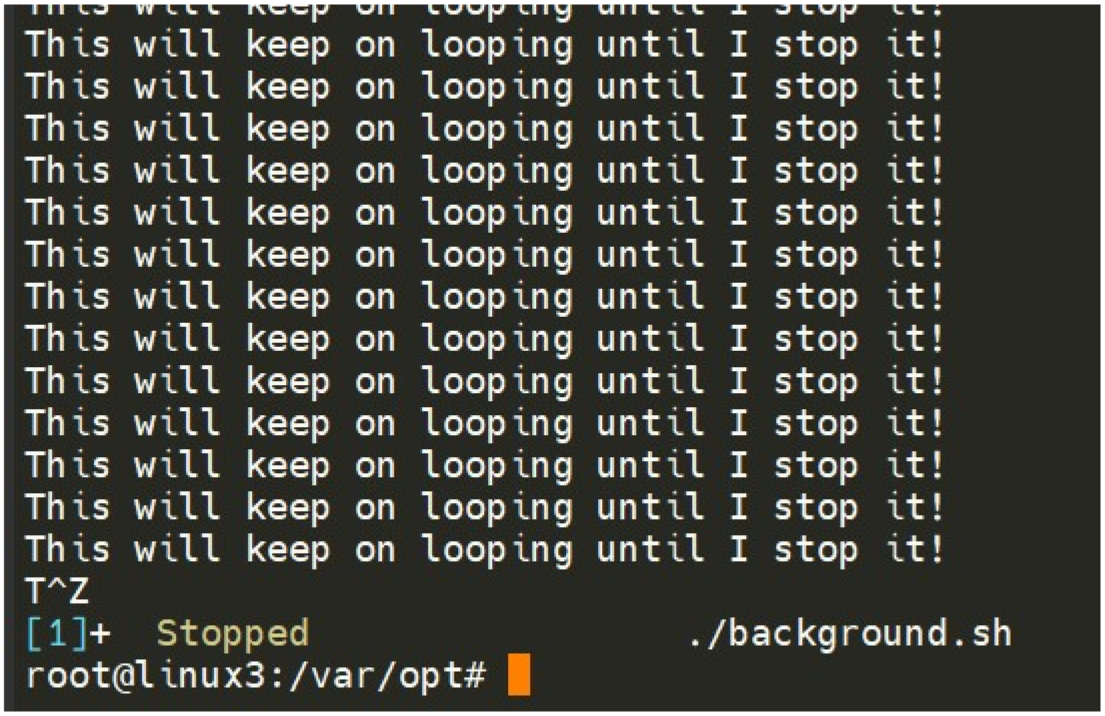


```
root@linux3:~# echo "Hi THM"
Hi THM
root@linux3:~# echo "Hi THM" &
[1] 16889
root@linux3:~# Hi THM
```

Here we're running `echo "Hi THM"` , where we expect the output to be returned to us like it is at the start. But after adding the `&` operator to the command, we're instead just given the ID of the echo process rather than the actual output -- as it is running in the background.

This is great for commands such as copying files because it means that we can run the command in the background and continue on with whatever further commands we wish to execute (without having to wait for the file copy to finish first)

We can do the exact same when executing things like scripts -- rather than relying on the & operator, we can use Ctrl + Z on our keyboard to background a process. It is also an effective way of "pausing" the execution of a script or command like in the example below:

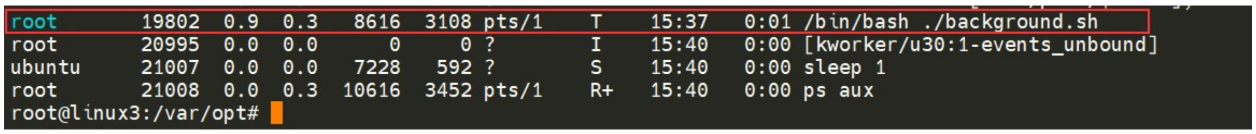


```
This will keep on looping until I stop it!
This will keep on looping until I stop it!
This will keep on looping until I stop it!
This will keep on looping until I stop it!
This will keep on looping until I stop it!
This will keep on looping until I stop it!
This will keep on looping until I stop it!
This will keep on looping until I stop it!
This will keep on looping until I stop it!
This will keep on looping until I stop it!
This will keep on looping until I stop it!
This will keep on looping until I stop it!
This will keep on looping until I stop it!
T^Z
[1]+  Stopped                 ./background.sh
root@linux3:/var/opt# 
```

This script will keep on repeating "This will keep on looping until I stop!" until I stop or suspend the process. By using Ctrl + Z (as indicated by **T^Z**). Now our terminal is no longer filled up with messages -- until we foreground it, which we will discuss below.

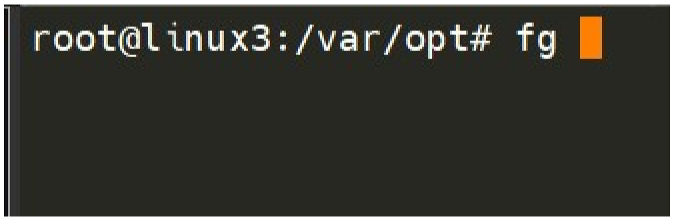
Foregrounding a process

Now that we have a process running in the background, for example, our script "background.sh" which can be confirmed by using the ps aux command, we can back-pedal and bring this process back to the foreground to interact with.

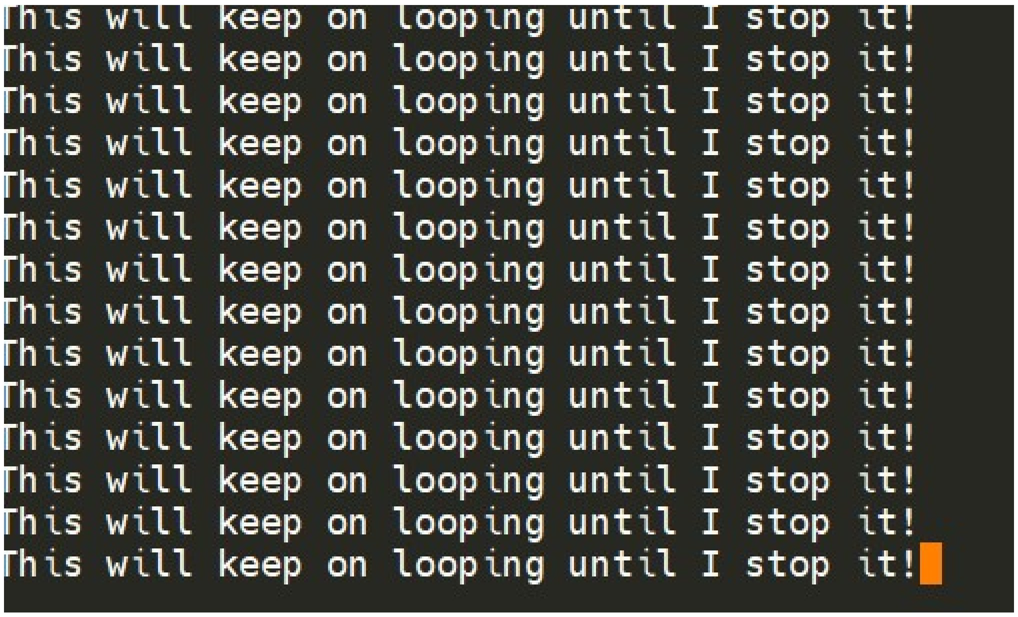


```
root       19802  0.9  0.3   8616  3108 pts/1    T    15:37   0:01 /bin/bash ./background.sh
root       20995  0.0  0.0      0     0 ?        I    15:40   0:00 [kworker/u30:1-events_unbound]
ubuntu     21007  0.0  0.0   7228   592 ?        S    15:40   0:00 sleep 1
root       21008  0.0  0.3  10616  3452 pts/1    R+   15:40   0:00 ps aux
root@linux3:/var/opt# 
```

With our process backgrounded using either Ctrl + Z or the & operator, we can use fg to bring this back to focus like below, where we can see the fg command is being used to bring the background process back into use on the terminal, where the output of the script is now returned to us.



```
root@linux3:/var/opt# fg
```

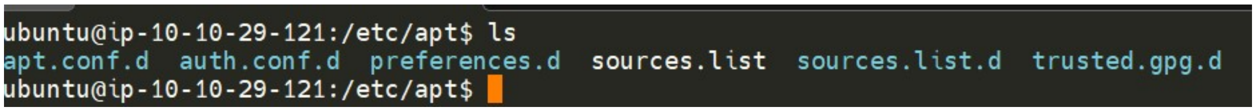


```
This will keep on looping until I stop it!
This will keep on looping until I stop it!
This will keep on looping until I stop it!
This will keep on looping until I stop it!
This will keep on looping until I stop it!
This will keep on looping until I stop it!
This will keep on looping until I stop it!
This will keep on looping until I stop it!
This will keep on looping until I stop it!
This will keep on looping until I stop it!
This will keep on looping until I stop it!
This will keep on looping until I stop it!
This will keep on looping until I stop it!
This will keep on looping until I stop it!
```

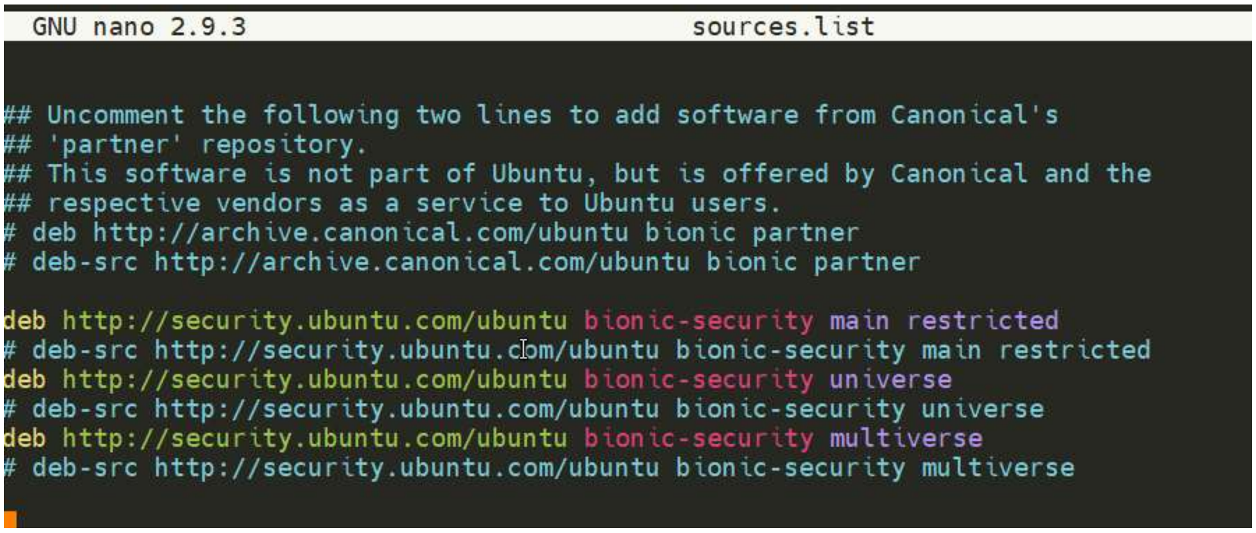
Introducing Packages & Software Repos

When developers wish to submit software to the community, they will submit it to an "apt" repository. If approved, their programs and tools will be released into the wild. Two of the most redeeming features of Linux shine to light here: User accessibility and the merit of open source tools.

When using the ls command on a Ubuntu 20.04 Linux machine, these files serve as the gateway/registry.



```
ubuntu@ip-10-10-29-121:/etc/apt$ ls
apt.conf.d  auth.conf.d  preferences.d  sources.list  sources.list.d  trusted.gpg.d
ubuntu@ip-10-10-29-121:/etc/apt$
```



```
  GNU nano 2.9.3                                    sources.list


## Uncomment the following two lines to add software from Canonical's
## 'partner' repository.
## This software is not part of Ubuntu, but is offered by Canonical and the
## respective vendors as a service to Ubuntu users.
# deb http://archive.canonical.com/ubuntu bionic partner
# deb-src http://archive.canonical.com/ubuntu bionic partner

deb http://security.ubuntu.com/ubuntu bionic-security main restricted
# deb-src http://security.ubuntu.com/ubuntu bionic-security main restricted
deb http://security.ubuntu.com/ubuntu bionic-security universe
# deb-src http://security.ubuntu.com/ubuntu bionic-security universe
deb http://security.ubuntu.com/ubuntu bionic-security multiverse
# deb-src http://security.ubuntu.com/ubuntu bionic-security multiverse
```

Whilst Operating System vendors will maintain their own repositories, you can also add community repositories to your list! This allows you to extend the capabilities of your OS. Additional repositories can be added by using the add-apt-repositorycommand or by listing another provider! For example, some vendors will have a repository that is closer to their geographical location.

Managing Your Repositories (Adding and Removing)

Normally we use the apt command to install software onto our Ubuntu system. The apt command is a part of the package management software also named apt. Apt contains a whole suite of tools that allows us to manage the packages and sources of our software, and to install or remove software at the same time.

Let's walk through adding and removing a repository using the add-apt-repository command we illustrated above. Whilst you can install software through the use of package installers such as dpkg, the benefits of apt means that whenever we update our system -- the repository that contains the pieces of software that we add also gets checked for updates.

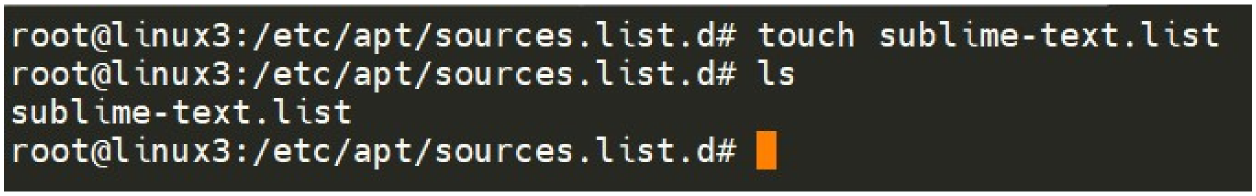
In this example, we're going to add the text editor Sublime Text to our Ubuntu machine as a repository as it is not a part of the default Ubuntu repositories. When adding software, the integrity of what we download is guaranteed by the use of what is called GPG (Gnu Privacy Guard) keys. These keys are essentially a safety check from the developers saying, "here's our software". If the keys do not match up to what your system trusts and what the developers used, then the software will not be downloaded.

So, to start, we need to add the GPG key for the developers of Sublime Text 3.

1. Let's download the GPG key and use apt-key to trust it: wget -qO - https://download.sublimetext.com/sublimehq-pub.gpg | sudo apt-key add -

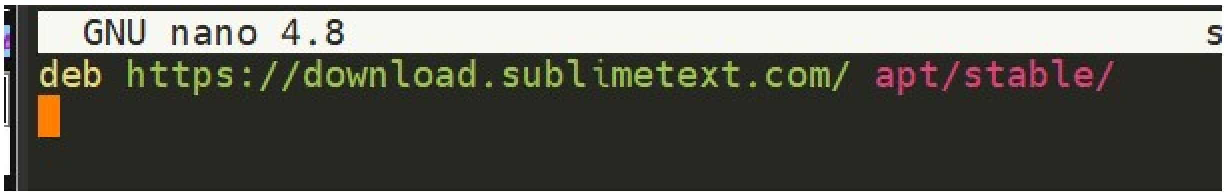
2. Now that we have added this key to our trusted list, we can now add Sublime Text 3's repository to our apt sources list. A good practice is to have a separate file for every different community/3rd party repository that we add.

2.1. Let's create a file named **sublime-text.list** in **/etc/apt/sources.list.d** and enter the repository information like so:



```
root@linux3:/etc/apt/sources.list.d# touch sublime-text.list
root@linux3:/etc/apt/sources.list.d# ls
sublime-text.list
root@linux3:/etc/apt/sources.list.d# 
```

2.2. And now use Nano or a text editor of your choice to add & save the Sublime Text 3 repository into this newly created file:



```
  GNU nano 4.8                                                s
deb https://download.sublimetext.com/ apt/stable/
```

2.3. After we have added this entry, we need to update apt to recognise this new entry -- this is done using the apt update command

2.4. Once successfully updated, we can now proceed to install the software that we have trusted and added to apt using apt install sublime-text

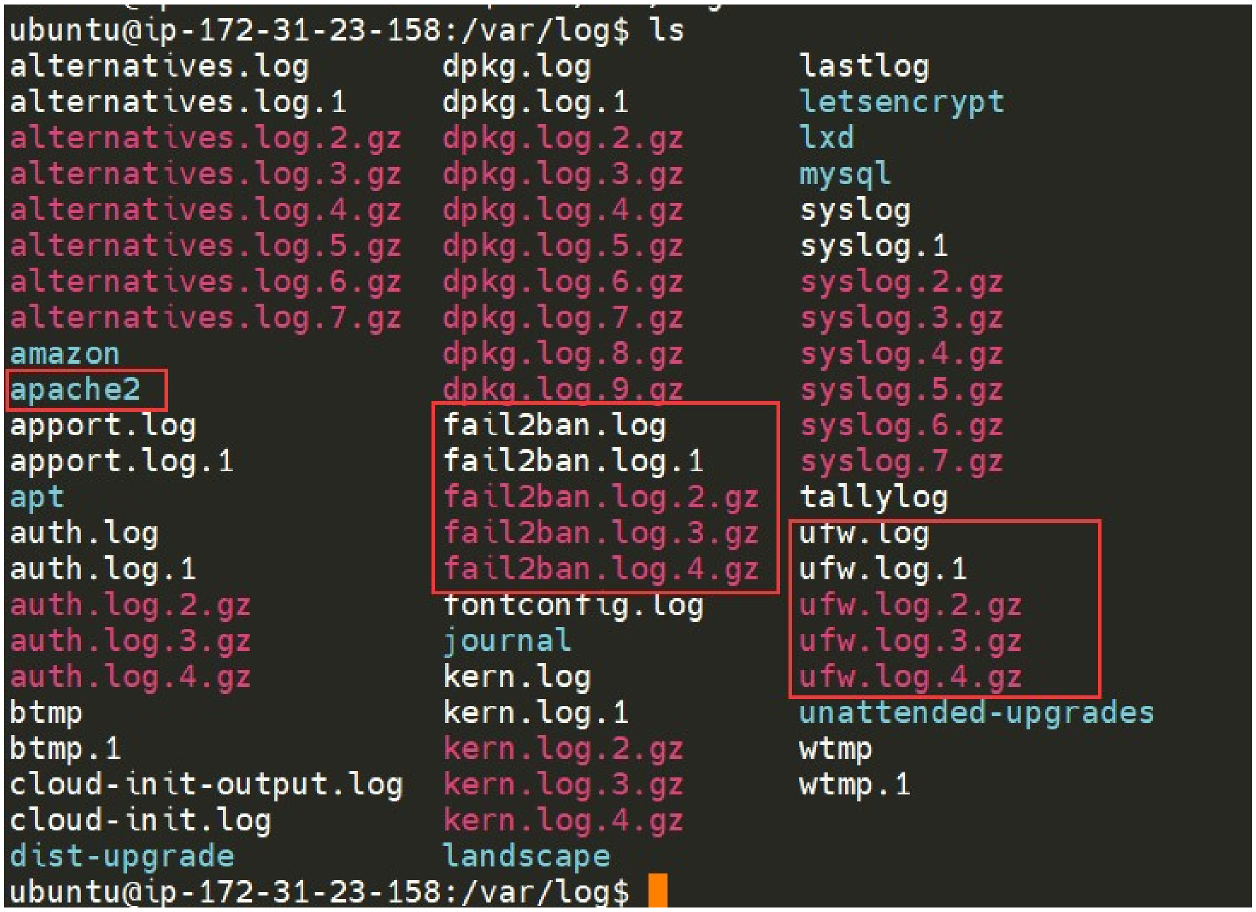
Removing packages is as easy as reversing. This process is done by using the add-apt-repository --remove ppa:PPA_Name/ppa command or by manually deleting the file that we previously fulfilled. Once removed, we can just use apt remove [software-name-here] i.e. apt remove sublime-text

Located in the /var/log directory, these files and folders contain logging information for applications and services running on your system. The Operating System (OS) has become pretty good at automatically managing these logs in a process that is known as "rotating".

I have highlighted some logs from three services running on a Ubuntu machine:

- An Apache2 web server
- Logs for the fail2ban service, which is used to monitor attempted brute forces, for example

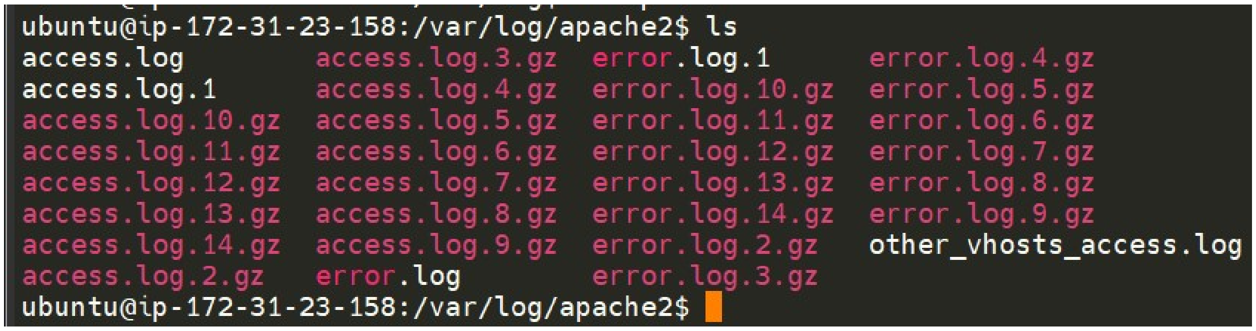
- The UFW service which is used as a firewall



```
ubuntu@ip-172-31-23-158:/var/log$ ls
alternatives.log       dpkg.log           lastlog
alternatives.log.1     dpkg.log.1         letsencrypt
alternatives.log.2.gz  dpkg.log.2.gz      lxd
alternatives.log.3.gz  dpkg.log.3.gz      mysql
alternatives.log.4.gz  dpkg.log.4.gz      syslog
alternatives.log.5.gz  dpkg.log.5.gz      syslog.1
alternatives.log.6.gz  dpkg.log.6.gz      syslog.2.gz
alternatives.log.7.gz  dpkg.log.7.gz      syslog.3.gz
amazon                 dpkg.log.8.gz      syslog.4.gz
apache2                dpkg.log.9.gz      syslog.5.gz
apport.log             fail2ban.log       syslog.6.gz
apport.log.1           fail2ban.log.1     syslog.7.gz
apt                    fail2ban.log.2.gz  tallylog
auth.log               fail2ban.log.3.gz  ufw.log
auth.log.1             fail2ban.log.4.gz  ufw.log.1
auth.log.2.gz          fontconfig.log     ufw.log.2.gz
auth.log.3.gz          journal            ufw.log.3.gz
auth.log.4.gz          kern.log           ufw.log.4.gz
btmp                   kern.log.1         unattended-upgrades
btmp.1                 kern.log.2.gz      wtmp
cloud-init-output.log  kern.log.3.gz      wtmp.1
cloud-init.log         kern.log.4.gz
dist-upgrade           landscape
ubuntu@ip-172-31-23-158:/var/log$
```

These services and logs are a great way in monitoring the health of your system and protecting it. Not only that, but the logs for services such as a web server contain information about every single request - allowing developers or administrators to diagnose performance issues or investigate an intruder's activity. For example, the two types of log files below that are of interest:

- access log
- error log



```
ubuntu@ip-172-31-23-158:/var/log/apache2$ ls
access.log        access.log.3.gz  error.log.1      error.log.4.gz
access.log.1      access.log.4.gz  error.log.10.gz  error.log.5.gz
access.log.10.gz  access.log.5.gz  error.log.11.gz  error.log.6.gz
access.log.11.gz  access.log.6.gz  error.log.12.gz  error.log.7.gz
access.log.12.gz  access.log.7.gz  error.log.13.gz  error.log.8.gz
access.log.13.gz  access.log.8.gz  error.log.14.gz  error.log.9.gz
access.log.14.gz  access.log.9.gz  error.log.2.gz   other_vhosts_access.log
access.log.2.gz   error.log        error.log.3.gz
ubuntu@ip-172-31-23-158:/var/log/apache2$
```

There are, of course, logs that store information about how the OS is running itself and actions that are performed by users, such as authentication attempts.