**Computer Security I Lab 4**

This is a continuation of Lab 3. I felt like there was too much information to squeeze into the third lab so I created another lab based off of it. In this lab we will discuss user permissions/ password policies, memory management/ memory addresses and some advanced commands.

**The chmod & chown command**

The chmod command is used to change the permissions of files and directories. This command at first can be a little tricky but once we explain it and go through a couple examples you should understand it. Permissions in Ubuntu from what we have seen are understood as “Read”, “Write” and “Execute” these permissions are given based on numeric values automatically provided. In most cases we want to edit the permissions for what users have access to. (It wouldn’t make sense for a software developer or penetration tester have access to the control of a database.)

The typical syntax for the chmod command is the following (we will get into it later):

“chmod 3\_Digit\_Permission\_Number filename”

Let’s discuss what I just said “permissions are given based on numeric values” this means that we have numbers 0-7 (these are called octal digits). The read, write and executable permissions all have their own respective value. This value doesn’t change for the 3 types of users “Owner”, “Group” and “Other”. This means that since we only have 3 types of users then we only need 3 digits of permissions. These 3 digits will dictate to us who has access to read, write and execute privileges.

The read privilege for the Owner, Group and Other users carry a 4.

The write privilege for the Owner, Group and Other users are 2.

The execute privilege for the Owner, Group and other user carry a 1.

The more permissions we add more a specific category the higher the number can go hence if the Owner needed to have write and execute privileges they would need a 5 in their respective permission spot. In the table below I will go through an example of how this works:

Let’s say for this example I want the owner of the directory called “Quantum\_Physics” to be able to read, write and execute files. Then I want the Group users to be able to just execute and read within the directory and lastly I want the other people to be able to execute to the directory. Understanding this we can build a little table which shows the permissions we want for each user:

|  |  |
| --- | --- |
| Owner | rwx |
| Group | rx |
| Other | x |

In the table above we illustrate the different user permissions for our example we are going through. The next step we need is to match the numeric permission values for each user. In order to do I have provided you with a chart of which numbers correlate to which permissions (this is what was in that blurb above). Check out the image below:

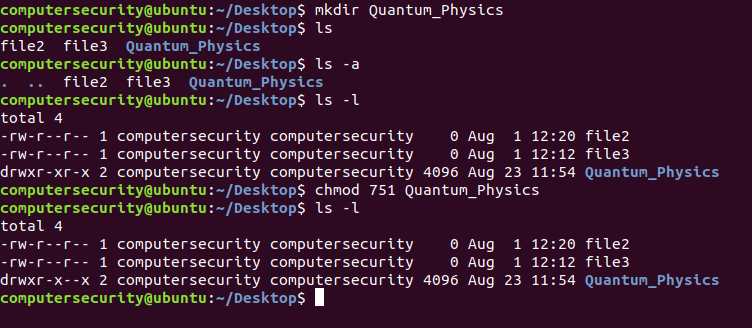
|  |  |  |  |
| --- | --- | --- | --- |
|  | Owner | Group | Other |
| Read | 4 | 4 | 4 |
| Write | 2 | 2 | 2 |
| Execute | 1 | 1 | 1 |
| User permission values(form example above) | 7 | 5 | 1 |

This means that for our permission command we would do the following:

“chmod 751 Quantum\_Physics” this would change the user permissions from what they are currently to

“drwxr-x—x” (which is permission 751)

Check out the image below for the actual example of this and feel free to enter the commands along with me: (the most important thing about every command you enter into the terminal is that you need to 100% understand what you enter).

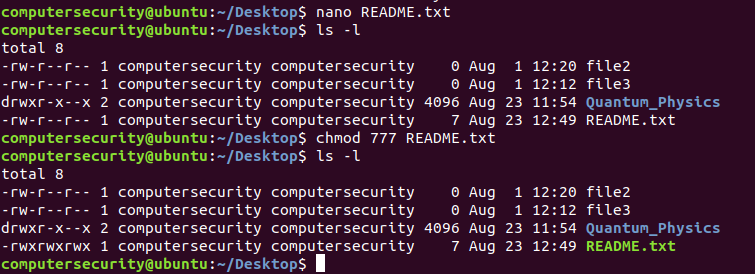


**Figure 4.1:** In the figure above we are viewing the current user permissions for the directory quantum physics. After we do that we are then changing the user permissions for the file.

In this example let’s say we have a file called README.txt and we want everyone to be able to read it, view and execute it. In this example I will show you the proper number in order to do this. (I know some of you might have the answer but I’m going to walk through it again):

The first step we are going to do is list out our user permissions (in our case everyone needs read, write and execute privileges)

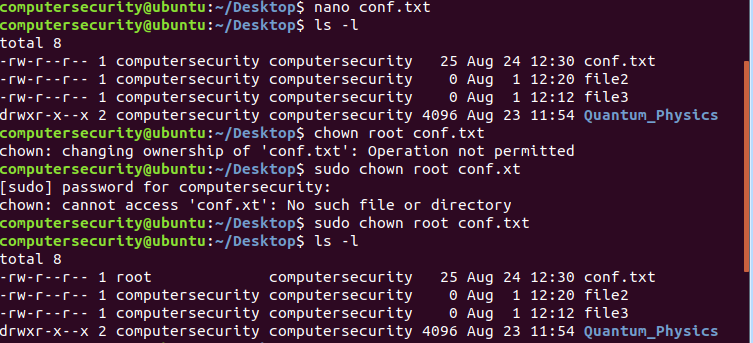
|  |  |
| --- | --- |
| Owner | rwx |
| Group | rwx |
| Other | rwx |

The second step we can do is look at the table above and see that since we need all users to have access to this README.txt file we can just add up the 3 columns and get “777” which means all users have access to read, write and execute files. Hence the command would be “chmod 777 README.txt”. Check out the screen shot below for an example: 

**Figure 4.2:** We are changing the readme.txt file to chmod 777 anyone can do anything to this file.

The next part of this section is the **chown** command. The chown command allows you to change the owner and group information of a related file or directory. For a more detailed explanation of this feel free to type in the “man chown” command which as always will bring up the flags and a more in depth look into the command.

Let’s go through a couple basic commands for the first example I’m going to take a file and change the owner of it. In my example below I will make a text file called conf. After I display this I will show you that the user who owns this file is computersecurity. Then after using the chown command I will show you that it now belongs to whichever user I made it belong to who in this case is root. Check out the image below:



**Figure 4.3:** Changing the ownership of the file conf.txt

In the first command using the nano text editor I created a text file called conf.txt there is nothing special in here except for some text.

In the second command ls –l we know what this does it lists out the read, write and execute properties along with some other attributes such as users who have access to the file ad when the file was made and such. While using the second we can see that the conf.txt file belongs to the user computersecurity.

In the next couple lines we go through some common mistakes while using the chown command the first is the use the command “chown root conf.txt” which doesn’t work because we always need to be root or sudo to change ownership on files. This to prevent files from getting in the wrong hands accidently. Second is if we spell the file name wrong. After those two we need to break down the correct command which is “sudo chown root conf.txt” breaking this command down line by line means the following:

sudo: we need sudo because we are non- privileged users or non-sudo users. To use the chown command a user must have privileges of the target user. So since we were giving to root and don’t have root privileges it wouldn’t allow us to give the file to root without sudo

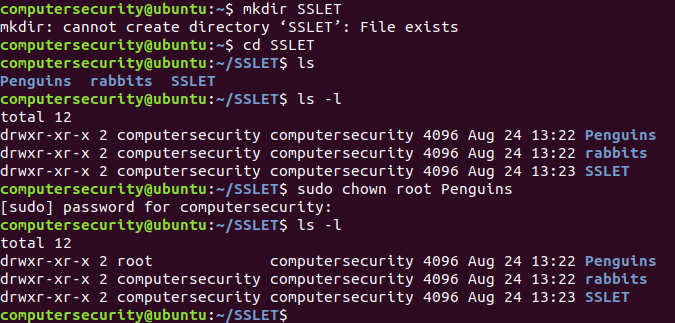
chown: Alter the user who has privileges to a file or directory

root: this could be any user I just happened to pick root because I didn’t want to make another user.

conf.txt: this is the file that we are changing user ownership of. It is going to go from computersecurity to root.

After the success of the command we use the ls –l again which shows the new owner is root. Hence the chown command was successful.

In the next example we go through how to change the directory owner. In this case I create a directory named “SSLET” and then within that directory I make 3 more directories (SSLET, Penguins and rabbits. The commands aren’t shown for space purposes). After I do this I change the Penguins directory to be owned by root using the chown command which is the same process as above. Check out my screenshot below and feel free to follow the commands:

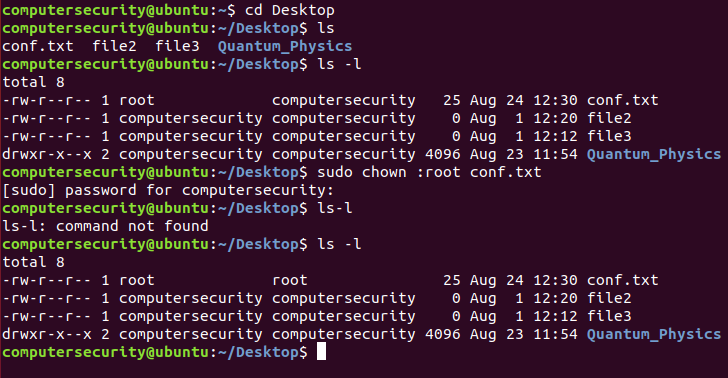


**Figure 4.4:**  The figure above shows how to change ownership of a directory

Since we have been using the chown command we have only been altering who the owner is of a file or directory. While using the chown command we can also change the permission of who owns the group of a file. In the next few commands we will go through how this works:

Let’s say for our example we wanted to alter the group in which the conf.txt file belongs to.(This file should be in our Desktop directory if you are following along if it isn’t I highly encourage you to make this file to follow along.)

In the terminal snapshot below we are going to take the file conf.txt as described above and make the group belong to root:



**Figure 4.5:**  Changing group ownership to root instead of computersecurity

In the first line we go into the Desktop directory by using the “cd” command.

In the second command we use the ls command to view all files and directories within the current directory which we are in. (I did this to make sure the conf.txt file is in there)

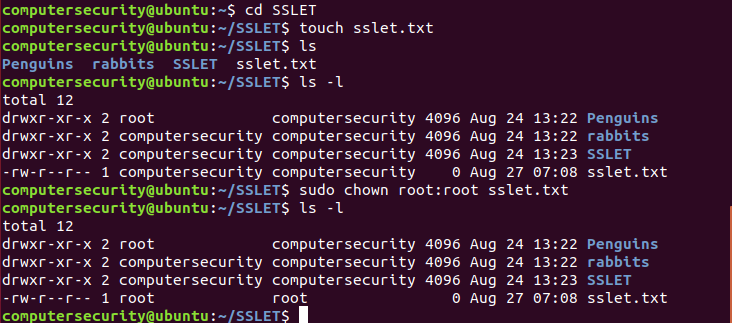
In the third command we use the ls -l command for a more in detail look of the files and directories within our current directory. By using the ls –l command as we know we are able to see the user permissions, the owner/group the file/directory belongs to and much more.

In the fourth command we are going to use the chown command to change the group from one user to another. The architecture of this command is the following: “sudo chown :[group-name] [file-name]”. In the terminal window you can see that we use the command “sudo chown :root conf.txt”. Since this command involves **changing ownership of a group** we need to use the sudo command. The next part of the command is the chown command which we know by now is used to change ownership of files/directories from one user/group to another. The next part of the command is the “:root” command this part of the statement means that we want the group to be changed to root everything after the colon in this command is reference to the group. (This will make more sense later on in the lab). Then the last part of the command is the filename conf.txt in which we are changing ownership of. Simply in English this statement changes the file conf.txt group user from computersecurity to root.

In the fifth command we use the “ls-l” command which fails because we need a space between our main statement and the flag which we are using.

In the sixth command we use the “s ls –l” (which has the space in between) and this shows us that the group user is now root and our transfer was successful.

As you can see the process to changing isn’t really all that difficult it’s pretty straightforward we are going to go through one more example then there will be questions following this section. The example is in one line changing both the owner and group of a file. The architecture for how this command is used is the following: “sudo chown [New-Owner]:[New-Group] [File-Name]” In our example we are going to create a new file named sslet.txt within our SSLET directory and we will change that file’s owner and group to root then we will have a brand new set of permissions for said file. Let’s check it out in the screenshot below:



**Figure 4.6:** Illustrating how we change both the user and group at the same time.

In the first command we went into our SSLET directory which should be made in our home directory and if it isn’t there please make one.

In the second command we create a text file using the touch command. This text file is called sslet.txt

In the third command we view the contents of our directory with the ls command to verify that our file was made.

In the fourth we command we use the ls –l command to check the user permissions, group and user etc…

In the fifth command we use the command “sudo chown root:root sslet.txt” this command must use **sudo** since we are transferring the file user/group to someone else, the second part of the command “chown root:root” is saying that we are transferring the file to user root and group root. The last part of the command is the file which we are changing to root(in both user/group) this is sslet.txt. In summary we are taking the file sslet.txt and making the user to root and group to root and we need to use sudo because we are transferring the file to another user.

Then in the last command we use ls –l to show that the new user and group is verified and it is:

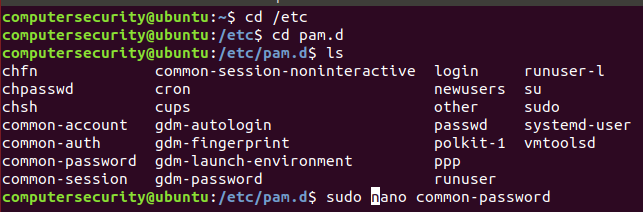
“root | root”.

**Password Policy**

A password is a very fundamentally and important aspect to security. A password is a form of authentication. It is part of the three factor authentication process: (1) “Something that the user knows” (this is typically a password),(2) “something that the user has” such as a pre- defined code or a fingerprint etc… and (3) third factor authentication is called “something a user is” this is usually a biometric like a voice recognition, iris recognition, fingerprint, retina scan or hand configuration.

In this case we are going to be dealing with one factor authentication and it is going to be passwords. A strong password policy and in general strong password is one of the most important aspects of your security. As you will learn some of the successful security breaches involve simple brute force dictionary attacks against weak passwords. Enforcing a password requirement as we will see is a great idea to make sure your attacker has a very difficult accessing the data/system.

Typically 99.99% of the time Ubuntu requires you to have a password their minimum character values are at a minimum 6 but can be edited to be longer which we will see. This is all controlled in a file within the pam.d directory which is in the etc directory. Check out my screen shot below to see how we get there:



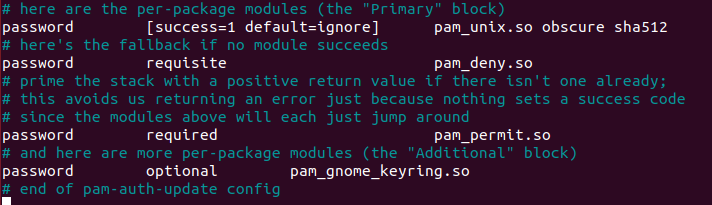
**Figure 4.7:** Accessing the files in the pam.d directory

In the first line I go through the /etc directory which is where the .config/ configuration files are and all the properties for the commands in which we use.

In the second command we go to the pam.d directory which is where we have password, username properties and much more located. For our purposes now we are just interested in the common-password file.

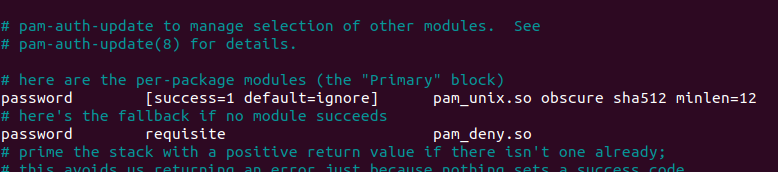
In the third command we list all the files and directories within the pam.d directory.

In the fourth command we access the file using sudo. “sudo nano common-password” this will call the nano text editor for the file “common-password”. If we scroll to the bottom of the file (using our down arrow key) we are able to see the following:



**Figure 4.8:** Inside the common-password file

The comments within the file do an excellent job of what is going within there. But I want to show in case you need to alter the minimum length for a password for a user we can go within this file and at the end of the first line where it says “pam\_unix.so obsure sha512” we can add the following “pam\_unix.so obsure sha512 minlen=12” where I have 12 can be any number. Check out the screenshot below:

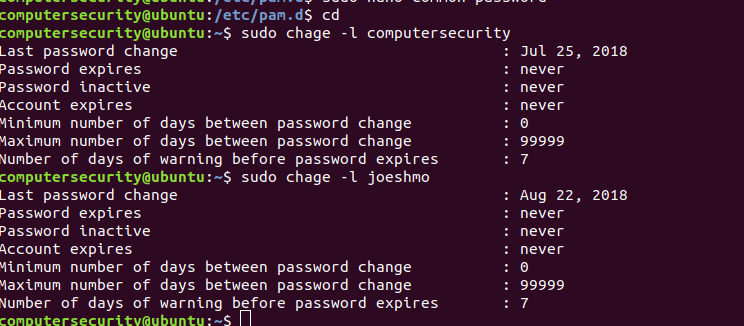


**Figure 4.9:** Enforcing Password entropy checks

These requirements which we are integrating into our System are called **password entropy checks**. These are checks within our passwords to make sure that the user passwords are at a secure level. The administrator (AKA the Systems administrator) isn’t limited to this they are not tied to these requirements.

We can view the status of different user accounts by utilizing the following command:

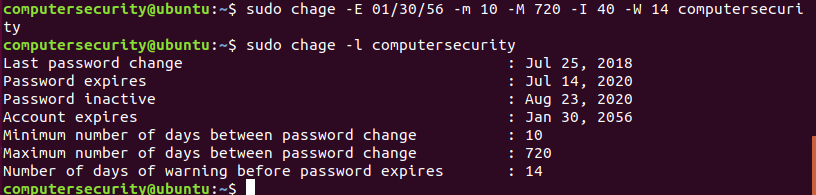
“sudo chage –l SomeUserName” this command allows you (as the administrator) to check out a user. In my example below my account “computersecurity” is the system administrator account and I created an extra user (which you can feel free to do) I named this user “joeshmo”. In the screenshot below we show how this command works:



**Figure 4.10:** Viewing the breakdown of a user in the Ubuntu system

In the first command I went to the home directory just to make sure I’m there.

In the second and third command we utilize the chage command which stands for “changing user password expiry info” I know it doesn’t look like it but it is. Secondly since we are changing passwords we need to use the sudo command, thirdly we use the –l is a flag which stands for list as in list out the contents which we want displayed, and lastly we write in the username of the user in first case it is “computersecurity” and in my third case it is “joehsmo”. Hence the command “sudo chage –l computersecurity lists all the details about the computersecurity account. Similarly the joehsmo account does the same.

We can also alter these properties which are being displayed and configure them to correctly show the proper system settings for each user. For this example I am going to alter the credentials of the user “computersecurity”. I will explain what I did when below the screen shot:

**Figure 4.11:** How to change the user settings for each account.

In the first part of the command I used the administrator command in order to change the password. The –E flag stands for the expiration date of the account which in my case is 01/30/56

The second part of the command is –m 10 which stands for the minimum number of days between password changes.

The third part of the command is the –M which is the maximum number of days between password changes.

The fourth part is the inactivity period or locking an account if not used for some x amount of time in this case it is 40 days.

The fifth part of the command is a warning message which displays 14 days before your password expires.

The last part is the user which is being impacted by these attributes.

Check out man chage to learn more about the command!

The second command shows the newly configured settings.

**Memory**

This is going to be a very brief intro to memory. We will not be getting super technical in here. That is to be elaborated on with your instructor.

Remember in the last lab we learned all that Binary and hexadecimal jazz we also learned getting the numbers in and out of different number systems. Well we learned that for two reasons: (1) when we deal with digital electronics/ Boolean algebra binary is commonly used and (2) when we deal with memory in a computer we use hexadecimal(your professor should touch upon this if they don’t you will get an extra sheet which explains all about memory).

In this section we are going to go over some different file types to just have in the back of your mind while you’re at the computer. This is imperative to have a basic understanding in because when we have that understanding it allows us to fully grasp what’s happening internally to the computer. So when we perform a certain action to a computer we understand on a hardware and software level what is actually happening.

Let’s start off with some basic memory information which you all hopefully know. A single bit is either a 1 or a 0. A bit is just a fancy name for a 1 or 0. When we refer 8 bits we call it 1 byte. Below is a table full of common memory sizes which we might know about:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Memory abbreviation | Actual memory name | Bytes | Bits | Power of 2 |
| 1KB | Kilobyte | 1,024 bytes | 8,192 |  |
| 1MB | Megabyte | 1,024 KB or 1,048,576 bytes | 8,388,608 |  |
| 1GB | Gigabyte | 1,024 MB or 1,048,576 KB or  1,073,741,824 bytes | 8,589,934,592 |  |
| 1TB | Terabyte | 1024 GB or 1,048,576 MB or 1,073,741,824 KB or 1,099,511,627,776 bytes | 8,796,093,022,208 |  |

A couple of things I want to touch upon is that I understand that there is more to this such yottabytes, petabytes, Exabyte and zettabyte but we will not be getting into them. One could also argue that I should be using 1,000 but since Math is our friend we will be using 1024.

People get easily get confused while converting numbers from Kilo Bytes to Giga Bytes or even Giga Bytes to megabytes the aim of this little blurb is to combat this with a popular technique. **The technique is as follows if we have a small unit such as a Byte or a Kilobyte and we want it to become a Gigabyte or Terabyte then we need to divide it by 1024. If we have a large unit and we want it to become a smaller unit we multiply the number by 1024.** Let’s go through a couple examples of this method:

For our next couple of examples we are going to take different values and change their units around. The first example is the value 100,230,600 bytes and we will convert it to Gigabytes:

. This yields us with a result that 100,230,600,200 bytes equals to 93.34GB (We are using 1024 **NOT** 1000). The reason to divide the number 100,230,600,200 by 1024 3 times is because we are moving from different units. The **first** being Byte -> Kilobyte, the **second** Kilobyte-> Megabyte and **third** Megabyte to Gigabyte.

The second problem we are going to do is convert 6,646,232 Megabytes to Terabytes. We are going to divide twice since we need to go: (1) Megabytes -> Gigabytes and then (2) Gigabytes->Terabytes. This means when we perform our division we get the following result 6.33TB. The work is shown below:

The third problem we are going to do is convert a large unit to a small unit. In this case we multiply our large number by 1024 ad we are able to get our accomplished value. For this example we are to going to show how to convert 89 Terabytes to bytes.

.

As you can see from the above example we had to multiply through 4 times by 1024 because we needed to go from Terabytes to Bytes. (We went from Terabyte-> Gigabyte, Gigabyte->Megabyte, ->Megabyte-> Kilobyte and then lastly Kilobytes -> Bytes.). This means 89 Terabytes is 97,856,534,872,064 bytes.

The fourth problem is we are going to convert 82 Gigabytes to Megabytes. In order to do this we are only going to move through one hop (Gigabyte -> Megabyte). This means that the computation is the following:

Below are a couple of questions which you can answer and if you need help below there will be a chart:

**Converting Smaller Units to Larger units:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Terabyte | Gigabyte | Megabyte | Kilobyte | Byte |

Divide by 1024 while moving through the sections (going to the left)

**Converting larger units to smaller units**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Terabyte | Gigabyte | Megabyte | Kilobyte | Byte |

Multiply by 1024 while moving through this section to the right

Where does hexadecimal come into this. Well deep deep down within the computer the computer understands values by binary ad we see them in a sectors within memory which are stored as hexadecimal. So we as humans see hexadecimals but the computer sees binary bits flowing through out it.

**The Grep command**

The grep command (grep stands for global regular expression print) this is a slightly advanced command but is very important and useful when you know how to use it. The grep command is used to search for text from a file or another command’s output. The grep command can return the lines it finds matches and the lines where it doesn’t.

In our example for this section we are going to create a python file. This python file will have some code which I will paste below. In order to complete this section you will not need to know python all you will have to do is read what I’m writing. The code is as follows:

class rectangle(): # we have a class called rectangle

def \_\_init\_\_(self,width,length): # we have a constructor with self which relates to our objects and width and length for our program

self.width = width # create an instance of name width

self.length=length # create an instance of name length

def area(self): # have the method area

return self.width \*self.length # multiply length and width get area of rectangle

a=int(input("Enter length of rectangle: ")) # get input for side a

b=int(input("Enter breadth of rectangle: ")) # get input for side b

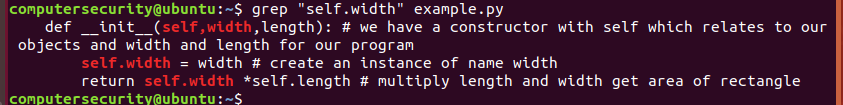
obj=rectangle(a,b) # pass parameters to the rectangle class for init

print('Area of rectangle:',obj.area()) # get the area

In order to copy & paste this code into a file we do the following:

1. We will create a new file I’m going to make mine called example.py
2. Right click inside the file and then exit and save from the file

Let’s say within this python file I want to search for the word “self.width”. The architecture for how the grep command functions is the following “***grep [OptionalFlag] SomePattern [file]”***. I am able to enter in the following command as illustrated in the screenshot:



**Figure 4.12:** Finding the pattern within the python file.

The first line of this and only line says “grep “self.width” example.py”. This is saying we are going to search the entire file example.py for the key word self.width.

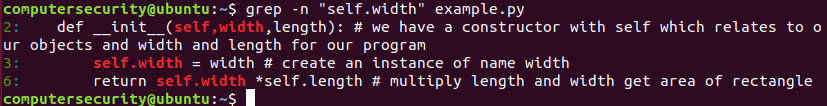
grep: this is the search command which we use to find patterns

“self.width”: the keyword we are looking for

example.py: this is the file which we are searching from.

If we analyze the output above we can see that we have multiple returns these are all the lines which have our desired patter hence they all get returned unless otherwise stated.

In the next example we are going to use grep to view line numbers of successful matches. In order to run this command we need to use the “-n flag”. The –n flag will display the line number for the matching keyword.

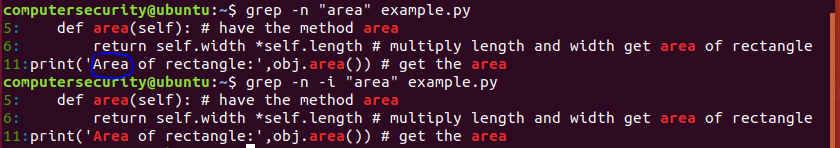


**Figure 4.13:** Showing the line numbers where the grep command falls on the python file.

As we analyze the output we can see that it is the same thing as above except in our second we have line numbers 2: 3:, 6: this indicates line numbers for our given key word.

The line “grep –n “self.width” example.py” will allow us to search for the keyword self.width in the file example.py and when it returns a match it will return the line number and the line of the match.

The grep command looks for the exact match of the given pattern in our command. That means if for some reason one of our words are uppercase grep will not pick up on it **unless** we perform a case-insensitive search. In order to do this we add the “-I” flag. In order to prove this to you check out my screen shot below:



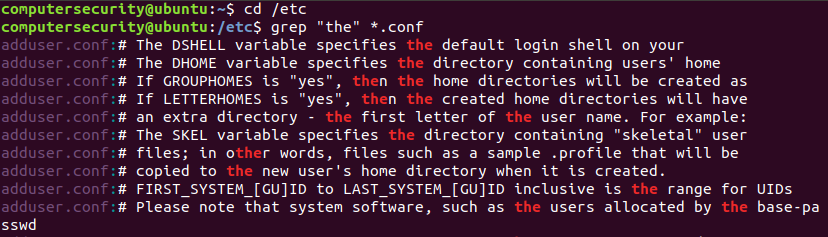
**Figure 4.14:** Ignoring he case sensitivity within the python file

As you can see in my screenshot above I ran two commands both which have the same result of line numbers but they both have one difference as we will see.

In the first command it highlighted in all red the places it saw the word “area”. Except it didn’t pick up on “Area” on line 11 because that has an uppercase “A” and different then “area”.

In the second command we added the “-I” flag which runs the case insensitive grep search so this will now highlight red “Area” on line 11 regardless if the pattern word has a uppercase/lowercase.

Hopefully you are starting to see that grep is a pretty useful command well it is about to get a whole lot cooler with it being able to search multiple file extensions. For this example we are going to maneuver to the /etc directory. Once in there we will run a grep search for the word “the” in all the .conf files. Check out my screen shot below (it is a partial screen shot the entire screen shot would be too much to take in)

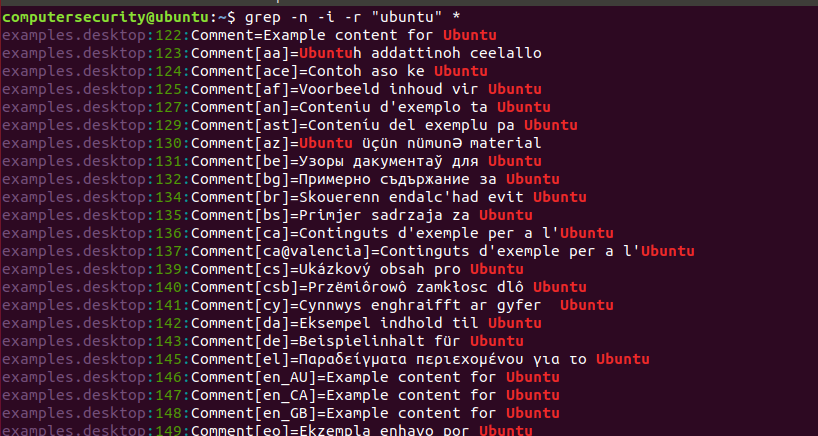


**Figure 4.15:** Using the grep command in another directory to show how it can be used.

In the first command we go into the /etc directory

In the second command while in the /etc directory we run the grep command. The grep command is run as the following: “grep “the” \*.conf”. This means that we will search in all the files that end in “.conf” for the word “the”. The “\*” asterisks symbol is denoted as a wildcard which means that it will take the name of any file which ends in our desired file extension and list out the matching pattern.

One of the next important things we can do with this is recursively search directories/subdirectories. We are able to recursively cycle through these using the –r flag. In order to do this we will follow the screenshot below:



**Figure 4.16:** The figure above recursively looks through all thedirectories for the word “ubuntu”

In the screenshot above we use the grep command but add the –r flag and the asterisks symbol. These are used to recursively search through all the directories and sub directories within our current directory. Hence we run the command “ grep –n –I –r “ubuntu” \* ”

This command can be broken down into the following:

grep: used to find patterns and searches within files.

The – n flag is used for displaying number lines

The –i flag is used to neglect uppercases and lowercases and just match the word.

The –r flag is used for recursion to cycle through all the directories/ sub directories

“ubuntu” is the word we are looking for and since we have the –I flag it doesn’t matter if it has uppercase or lowercase but as long as the word is spelled correctly.

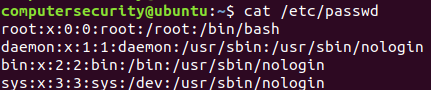
The “\*” symbol is the wild card symbol this will cycle through all files and search for the word “ubuntu”

In summary grep is a very powerful command used to find key words within different files without having to immediately access them but as we will see within the next command the “cat” command can do the same thing with a twist.

**Cat Command**

The cat command short for concatenate is one of the most frequently used command within the Linux Operating System. The cat command allows us to create single or multiple files, view file contents and redirect output in the terminal. This is the last section we will go through in this lab.

1. The first command to get us familiar with the cat command is just displaying the contents of a file. We can use this to display the text in a file. The screen shot below illustrates a sample snippet of the use of the command:



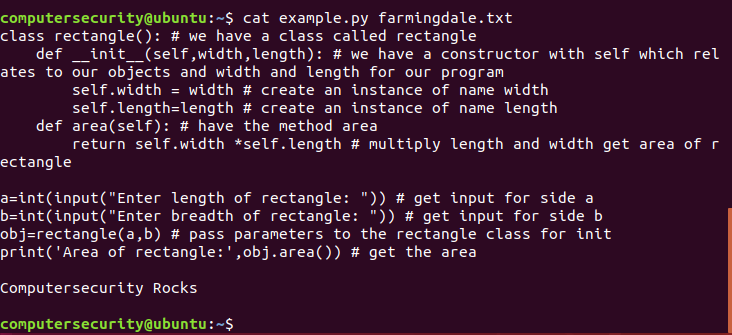
**Figure 4.17:** Showing some of a file using the cat command

The cat /etc/passwd command as stated in the screenshot allows for the text within the file passwd to be printed. If we break this command down we get the following:

cat: this is the command which is used to display contents within a file

/etc/passwd: this is the directory which we are trying to view the contents of.(this is the password directory)

1. The next command with cat is going to be viewing the output of multiple files. This can be done by using the file example.py and a new text file which we can call farmingdale.txt(within the text file we can put the text “Computersecurity Rocks”). After we have the two files “example.py” and “farmingdale.txt” we can use the cat command in order to view the output of both files. In order to do this we call the cat command and then the two files. Check out the screen shot below to see how we do this:



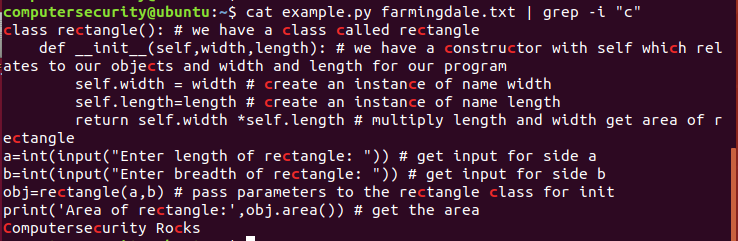
**Figure 4.18:** Showing two files using the cat command

In the command “cat example.py farmingdale.txt” is used to display the contents within both

the files. A breakdown of the commands is below:

The “cat” command as we know is used to print out files

The two files “example.py and farmingdale.txt” are both files with contents in them. These two files are what we are outputting to the terminal.

1. The third thing we are going to do with the cat command is something called **piping**. We use piping in order to direct the output and use multiple commands along a single command. In order to use piping we use the “|” icon or the straight line icon. So for instance if I want to run the cat command on both files and search for the letter “C” I would use the cat and grep command on both the files. In order to do this the output would look like the following: 

**Figure 4.19:** Learning how to pipe in Linux

In the output above we are using the cat command to view the output of the two files. Then we are piping (or re directing) the output of this command to be the input of the next command which is “grep –I “c”. Think of the idea of piping as almost a chain we do one part of the chain and the next input is the output of the prior command. Still confused? Check out the image below:

grep –I “c”

cat example.py farmingdale.txt

The output from the cat command is the input to the grep command

If we want we can break down this command “cat example.py farmingdale.txt | grep –I “c”” if we did this we would have the following:

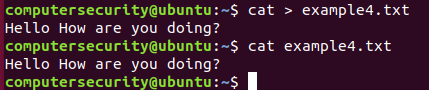
The first part “cat example.py” farmingdale.txt: this part of the statement we know outputs the contents from the files.

The arrow above in my example is supposed to denote the pipe symbol or “|” is the pipe which we used to tunnel through the output of one command as the input into another command.

The last part of the command “grep –I “c”” which we know will find the given pattern which in this case is “c ” and we know the –i flag negates the case sensitivity search.

Hence the full command “cat example.py farmingdale.txt | grep –i “c”” displays the output of the two file and then finds the pattern of “c” within the two outputted files.

1. As we did with the nano and touch command we can also create files with the cat command. In this section when we create the file we are going to use the symbol “>” this indicates to the terminal that we are going to be awaiting user input. Once we enter in our desired input we will hit ctrl+d in order to terminate the session. Let’s check out the screenshot below to see how we do this:



**Figure 4.20:** In the first command we create a text file and then put the contents “Hello How are you doing” inside the file. Then we use the cat command to view the contents.

In the first command we used the following:

“cat > example4.txt”

When we break this command down we are able to see that the cat command is awaiting user input for the file example4.txt.

The cat command is used to print out or await user input from the user.

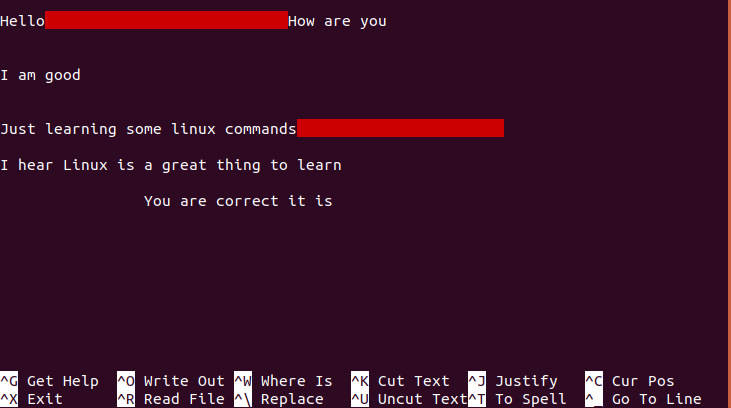
The “>” operator is used for user input.

The file example4.txt is what we are inputting our input contents into.

Once we finish with our inputting we hit ctrl + d and it will stop inputting the terminal.

On the second command we use the cat command as normal and are able to view the contents within the file.

1. For the fifth example we are going to create another text file which will have empty lines and tabs in it. This is to check other flags which are utilized within the Ubuntu Operating system for the cat command. The text file is as is:

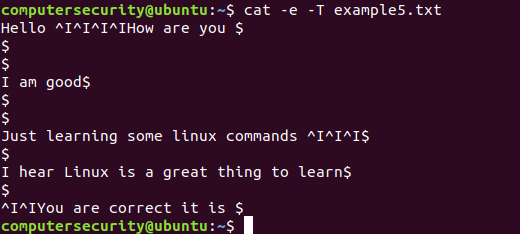


**Figure 4.21:** A tabbed text file.

\*Note the red just means I tabbed in the text file.

I made the file above through nano text editor and named it example5.txt

After I do this we are going to run a command “cat –e –T example5.txt”. It is shown below:



**Figure 4.22:** Understanding some of the weird syntax output. (Hint look up about regular expressions)

This output might look confusing so we are going to break it down. The flag “-e” and “-T”

The normal cat command as we know is for searching through files essentially

The –e flag displays an end of a new line or when I entered the “enter” button. This option is useful for squeezing multiple lines into one single output. (This is additionally useful so our terminal isn’t cluttered with randomness)

The –T flag is the tab operator this is used for someone performs a tab operation with in a file instead of you seeing a bunch of space tabs you will instead see “^I” the amount of “^I” you see determines how many indents I performed. For example the first line is below:

“Hello ^I^I^I^I How are you $”

Remember ^I is indentation

Remember $ is a new line

This is saying we have four indents in-between “Hello” and “How”

Then after “How are you” we have the $ symbol this represents that we are going to a new line (so basically it means you entered the enter button)

These two flags are very important if you are ever confused about these flags please feel free to contact me or look it up. We can also use the man command for the flags.