**Computer Security I Lab 5**

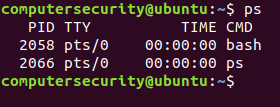
Congratulations on getting to the fifth lab within the Computer Security course curriculum. I understand that some of these labs are tricky and require an incredible attention to detail so I applaud you on getting this far. If you have not yet created Github repo for the class it might be best for you to and have the different Microsoft word documents uploaded in there. In this lab we are going to go through some more important commands which in general are great to be exposed to. These commands will make you seem like a wizard while in an Ubuntu/Linux environment.

**The ps command**

This ps section is going to be a semi long section so please get comfortable and get ready to read.

The ps command it is a very powerful command it allows for the user to see their current running processes on their computer. The ps command is commonly used in the real world with grep which we will touch upon. These commands and flags as we will see will help us filter through the output.

The ps command by itself “ps” shows the running processes running within the terminal. When we run the “ps” command we get the following output below:



**Figure 5.1:** Using the ps command to see how it functions

If we break down this output for the ps command we see that we have an output of 4 columns. These 4 columns are labeled the following: PID, TTY, TIME and CMD.

The PID column is the number of the process. We will use this later it is basically a reference number for a process which is running.

The TTY column is the name of the console that the user is logged into and executed said command. Along with the TTY command we might see a keyword “pts” which stands for pseudo terminal slave it just means that we are in a certain terminal number this will make more sense when we analyze the output.

The time column is the cumulative CPU time in which the operation was being used for.

The fourth column is the CMD or the argument which you entered into the terminal.

The ps command output from our terminal shows that the process id for our entered commands is the following:

2058 and 2066

These numbers above are used in case we need to interact with a given command perhaps we need to kill the process because it is frozen. (We will get into this later)

The TTY column is the console or terminal number that user is logged into so for my case we have pts/0 which means that you are all logged onto the system through the 0th terminal.

The time indicates the CPU time this command was quicker than the allotted time frame we are given HH:MM:SS. That means the command was run quicker than a second which is very likely.

Then the fourth column the CMD is the commands which we were activated when using the bash command the bash shell was executed when we entered the ps command and then the ps command was executed as well. Hence the fourth column dictates the command operated within the shell.

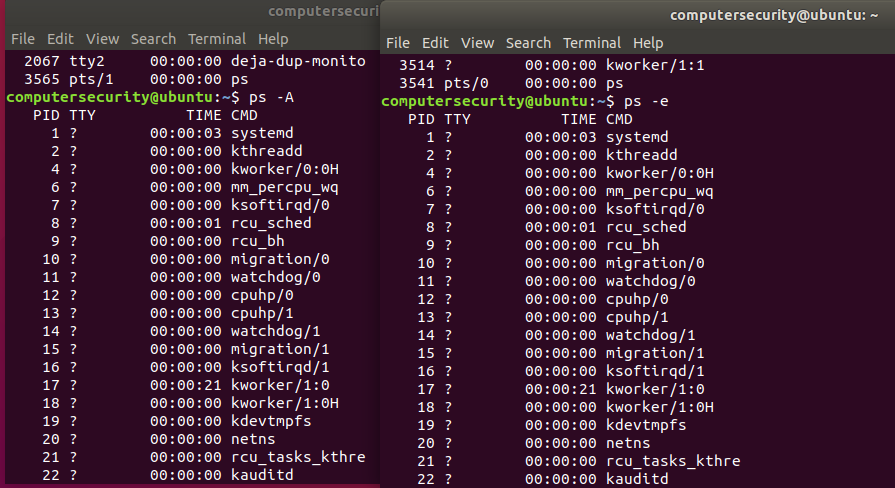
This I the most basic form is the ps command. As you can see it isn’t quite useful solely like this but if we add flags and other commands to it becomes a very powerful command. As we go through this terminal we will see multiple commands which can achieve the same solution. This differentiation is due to the fact that the ps command is adopted within the BSD syntax and Linux syntax (we will get into this later).

In order to run a command which shows our current running processes except for **session leaders. (A session leader is where every process id equals a session id or essentially a session leader is when one process kicks off other processes that process is now the session leader. For example if we have 5 processes “A, B, C, D and E”. We can tell process “A” to kick off “B” and “D”. Then tell process “C “to kick off “B”, “D” and “E”. This means that the following displayed are not session leaders “B”, “C”, “D” and “E”. The only session leader remaining is “A” because it kicked everyone off it is the root.** We can run the following command(s) which will allow us the processes except session leaders(don’t worry if you don’t understand this now it will make sense the more you read it):

“ps –A” or “ps –e”

Let’s see a couple examples with the following command being illustrated:

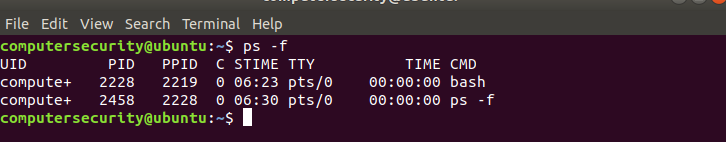
Below is an output of two terminals one running the “ps –A” command and one running the “ps –e” command. Don’t forget these commands show all the active running processes except for session leaders:



**Figure 5.2:** Two terminal output of the ps –A and ps –e command to prove that they are the same.

As we can see form the above screenshot the output is the same this is just a snippet feel free to look through the whole thing though.

1.) The next flag we are going to mention is the –f flag. This flag provides a more detailed output in regards to the information on the processes running. If we look at the screenshot below we should something like the following:



**Figure 5.3:**  Running ps –f to see the output columns

The first column in the above output states “UID”. The UID is the User Identification. We can verify this by typing “man ps” and seeing the following:



**Figure 5.4:**  When we run the man ps command and look for uid this comes up

I know what you’re thinking this tells us nothing except to see euid. Once we scroll to euid within the man command we get the following output:



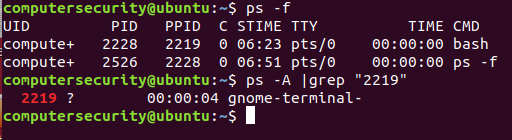
**Figure 5.5:**  When we run the man ps command and look for euid this comes up

This output above states that EUID and UID do the same thing which is show the “effective User ID”

The second column is the PID or the process ID this number as we know represents the process which is running.

The third column is the PPID or the parent process ID. The Parent Process ID tells us which process ID started that given process. Hence if we look at the second row we can see the process ID of the current task is “2458” and the parent process ID is “2228” which is row 1 the bash cmd. We then see that row 1 has a process ID of “2228” which makes sense and then after we can see that there is a parent process ID of “2219” for the bash command. The output here provides us with a wealth of information except for what is the parent process ID of the bash command. In order to do find out what the parent process ID is we can use our ps command and grep. We are able to use the “ps –A” command to display all the current processes then we are able to pipe the output of the ps command into a grep command which will match the output of what we are looking for (which in this case is the parent process ID). If we look at the output below we will see the output of the following:

* ps –f
* ps –A |grep “2219”



**Figure 5.6:** Using the grep and ps command jointly to find a running process.

According to output row 1 we are able to see the Parent Process ID is 2219. After running the second command with grep we are able to see that the parent process ID is the “gnome-terminal-“. The gnome-terminal is the current terminal in which we are using. That means from an architecture point of view we have the terminal up ->after the terminal is running we run the ps –f command which activates the bash terminal so that process starts up. Lastly after this the ps –f command runs. Hence we go gnome-terminal -> bash -> ps -f.

The fourth Column just says “C” this is the CPU utilization currently of the given process. The CPU utilization % is 0 here because there is no %CPU\_Utilization being used.

The fifth column is “STIME” or the start time of the given process.

The sixth column is the TTY or the terminal associated with the process.

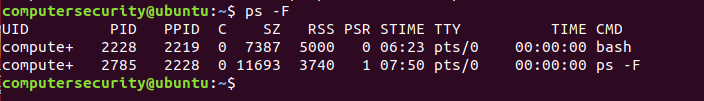
The seventh column is the TIME command which is the total CPU Usage or a given time.

The eighth column is the CMD or the name of the given process.

These columns are very important to understand how certain functions and resources run with the Linux Operating System.

1. The ps –F command is the next command which we will go through. This command allows us to have an extra full format opposed to just the –f flag.

Instead of me explaining to you what the additions are in this command opposed to the –f flag allow me to show you the output of the screenshot so you can see for yourself:



**Figure 5.7:** A more detailed output of the ps command is shown above

As we did with the prior command we will go through every column to make sure you understand what is happening in the output

The first column as we know is the UID or the User Identification.

The second column as we know is the process ID number which we know is the numerical number associated with the entered command.

The third column as we know is the parent process ID which is the process that allows that current process to work, (Kind of like a parent child relationship).

The fourth column is the “C” which we know stands for the current CPU Utilization. Of the given command.

The fifth column is “SZ” which according to the man command shows us the size in physical pages of the core image of the process. This is just fancy talk… The SZ command shows the virtual memory in which the command utilizes to perform its action(s). The main difference between this command and other memory commands is that it is giving us units in terms of pages. (We assume 1 page size is 4096 bytes)

The sixth column is the “RSS” column which is called the Resident set size and is used to show how much memory is allocated to a certain process. This is an actual number in kilobytes of how much RAM the current process is using.

The seventh column is the PSR or the processor that the process is currently assigned to.

The eighth column is the STIME or the start time of the process.

The ninth column TTY is the current terminal session.

The tenth column is the TIME or the total CPU usage used for a process.

The eleventh command is the CMD which is the command entered into the terminal.

1. The third command we are going to talk about is the ps aux command. The “ps aux” command is similar to the other commands except with a twist. Before we go into it let me explain what we are doing:

The ps command as we know will show all the current running active processes. When added with certain flags we get all the current running process and other stuff within our terminal. (It is important to note when we get a long terminal output we can also use grep to find what we are looking for). The aux in the ps aux command as you might see doesn’t have any flags naturally but when entered into the terminal an output is displayed. This doesn’t need to be “aux” it can be “ax” or “ux” or “ua” etc… so it is important to understand what the “a u x” stands for in ps aux.

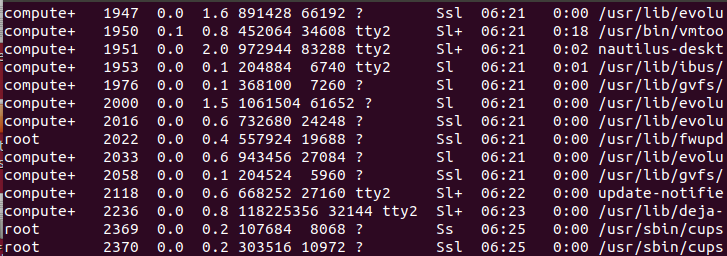
The “aux” stands for the following:

“a” = show’s processes for all users.

“u” = display the processes of the user/owner

“x”= shows the processes not attached to the terminal.

The ps aux by itself will show all processes for all users, display the processes of the user/owner and show the processes not attached to the terminal. A snippet of the output of the command is show below:

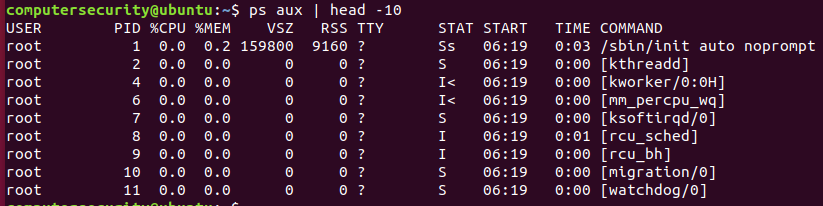


**Figure 5.8:** A snippet of the ps aux command is above

As we can see as great as this command is it has a downside. Our entire terminal is filled up with the output and we can’t even see the top row which tells us what is happening in each column. In the next paragraph we discuss how to fix these issues.

The first thing we are going to tackle is the first 10 rows and the last 10 rows from the ps aux output. In order to do this we are going to introduce two news commands the **head** and the **tail commands**. The head command allows the user to see some specified number -n of the output. When I say “n” I mean the user picks the number.

The command below which I’m going to show you is going to take the output of ps aux but only display the first 10 rows. The command is “ps aux | head -10” and is shown below:



**Figure 5.9:** Using the head command to show the first 10 processes being run from ps aux.

This will display the first 10 rows as stated above and is a very detailed look at what is happening within the user services. Below this will be a breakdown of what is happening with the columns:

The first column is the USER which is the textual user ID

The second column is the PID or the process ID for the given process.

The third column is %CPU which is the CPU utilization of the given process. The way this is calculated is taking the CPU Time and dividing it by the time the process has been running.

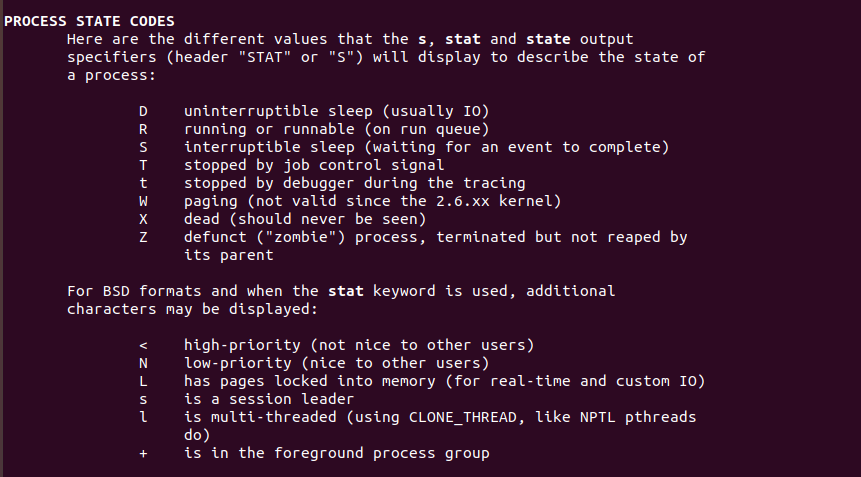
The fourth column is the %MEM which is the ratio of the process’s Resident set size (RSS) to the Physical memory on the machine.

The fifth column is “VSZ” which is virtual memory size of the process that is being run it is currently expressed in Kilo bytes.

The sixth column is RSS or the resident set size which is the memory that is allocated for the process and is in RAM.

The seventh column is TTY or the terminal interface if there is no number and there is a “?” it means there is no terminal interface.

The eighth column is the STAT which represent different file functions the best way to understand this is to understand the screen shot below which is taken directly from the “man ps” command:



**Figure 5.10:** The screen shot above shows the process state codes.

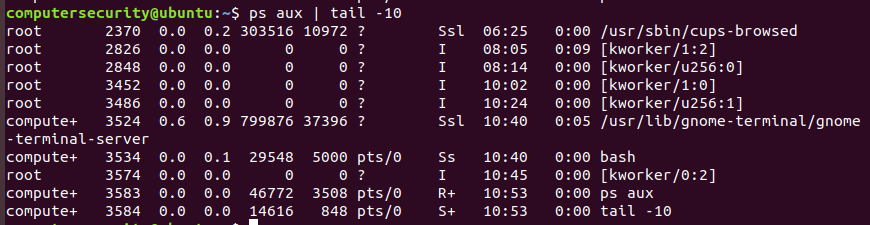
The ninth column is the START which is the start time or date of the given process

The tenth column is the total sum of the CPU Time of the command.

The eleventh column is the command which was run in order to get the row set up.

These eleven rows encompass the way the ps aux function works. I understand that this is a lot and you might be very confused so feel free to re-read sections and ask **QUESTIONS**.

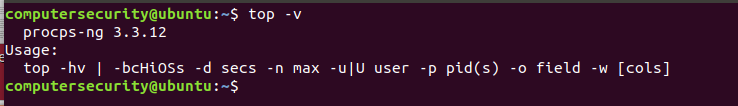
Alternatively to the head command we can also run the tail command which displays in reverse the order of ps aux. The image of the command “ps aux | tail -10” is below:



**Figure 5.11:** Showing how ps aux commands runs with the tail command

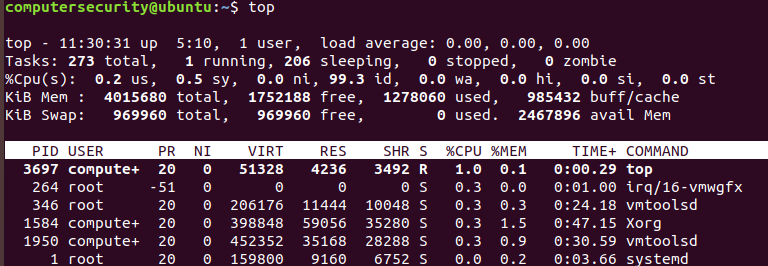
In this command everything is displayed as normal just in reverse which could be helpful if we were running a bunch of commands and needed to see the last few processes. (It is also important to note that the head and tail commands can also be used with files.).

**The top command**

An alternate to running the ps command is the top command which enables us to get a glimpse of running processes. In order to do this we just run the command “top” into the terminal. Before we do this though it is important to check the version of top which we are using. **In order to do this we use the “-v” flag** this allows for us to get the version of top. The top command along with the version flag is displayed below: 

**Figure 5.12:** Checking the current version of top which we are using.

After we know which version of the top command we are using we can enter just the command “top” into the terminal. Once we do this we will get something illustrated below: (my screenshot is a small code snippet of the command):



**Figure 5.13:** Running the top command to see the current active running processes

I understand that when you pull it up on your terminal instance there will be a lot going on within the screen. Some of the headers though do make sense when you read them. Below I will go through the headers row by row:

In the **first** row of the top command we have the following:



**Figure 5.14:** Breaking down the first row of the top command.

The first row indicates the following starting from left to right:

“top” – 11:42:31 current time

“up” – stands for the uptime of the machine in my case the uptime or the length the machine has been running for is 5:10 or 5 hours 10 minutes.

“1 user” this is the users sessions has logged currently there is 1 user active and it’s me.

“load average” refers to the last minute of activity (the left most value), the past 5 minutes (this is the middle number) and the past 15 minutes (the most right number). The “load” is a measure of the amount of computational work a system performs. This is very important when trying to see how fast a given system is.

Let us consider an example to understand load averages. If we have a single core system with a load average of 0.6 (this means if we multiply this number by 100 we will see that the System is doing 60% of work it can do). Another thing to note is a load average of 1 means that the system is exactly at capacity because if we multiply 1 by 100 we got 100% (hence the system is at full capacity). If we attempt to add even more to a system we will be overloading the load average which means we will overload the system. For instance if we have a load average of 1.86 that is 186% above the System capacity hence we are 86% above the capacity (which isn’t good. In order to get combat this we can download the “htop” command we will not be getting into that here currently)

In the **second** row has the following information:



**Figure 5.15:** Breaking down the second row of the top command

The second row indicates the following starting from left to right:

“Tasks: 273 total” this means there are 273 processes running in total.

“1 running”: there is 1 process running

“206 sleeping”: There are 206 processes sleeping (when a process is sleeping it is waiting for a signal. The signal will tell one of the sleeping functions to wake up and perform a task)

“0 stopped”: means that there are 0 processes stopped

“0 zombie”: processes waiting to be stopped form the parent process. Another definition for a zombie process is a process that has completed execution but still has an entry in memory.

The last two things I want to talk about briefly which are not located in here are called “orphan processes” and “dameon processes” The orphan process is a computer process whose parent process has finished or terminated. A daemon process is a process which runs in the background of the computer (it usually ends in a “d” letter to indicate that it is a program which will run in the background).

In the **third** row we have the following information:



**Figure 5.16:** Breaking down the third row of the top command

The third line here indicates how the CPU is being used. If we sum up all the percentages total will be 100% of the CPU or close to it. Let’s go through this:

“%Cpu(s): 1.7us”Percentage of the time spent in user space 1.7% of the time the user has spent in user space.

“1.0 sy”: percentage of the CPU for the system processes. Also known as the time spent in kernel space.

“0.0 ni”: This is the time spent running user defined processes.

“97.1 id”: This is the time spent in idle operations or processes not being used.

“0.0 wa”: Time spent on waiting on IO peripherals like USB or Disks

“0.0 hi”: This is time spent from handling hardware interrupt routines. Before we get into what a **hardware interrupt is?** Let’s talk about a regular **interrupt.** A regular **interrupt** is the way for external devices to get the attention of a piece of software. **Hardware interrupts** are generated by hardware devices (keyboards, controllers, network cards, sensors etc...) they are essentially signals sent out from the hardware devices which signal something to the CPU (for instance maybe some form of data has arrived)

For example when we discuss **hardware interrupts** think of an external peripheral. When we press keys on the keyboard or move a mouse it triggers these hardware interrupts which cause the processor within the keyboard to read the keystroke or the mouse to read the position on the screen.

“0.2si”: this represents the time spent handling **software interrupts**. **The software interrupts** are generally used to make system calls such as a request to an operating system to perform an Input/output operating or run a new program.

For an example of a **software** **interrupt** think when someone tries to divide by zero on the computer an exception will be thrown by the computers ALU or Arithmetic Logic Unit (the thing in the computer that does all the math for you). The operating system will catch this exception and decide what to do with it generally it displays an error message in regards to the error.

“0.0 st” This is the amount of CPU “stolen” from the virtual machine by the hypervisor for other tasks. This represents time when the real CPU (local host) was not available to the current virtual machine AKA it was “stolen”. Usually to run another virtual machine. For instance maybe you want to run a virtual machine within a virtual machine then this number would change.

In the **fourth and fifth** row there is the following information:



**Figure 5.17:** Breaking down the fourth and fifth row of the top command.

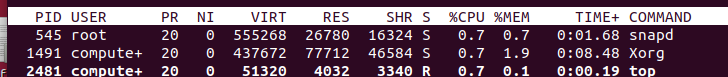
This fourth row is pretty self-explanatory there might be one thing which is confusing which we will tackle. The memory discussion for this row is talked about in kilobytes and shows how much memory is being used in total, how much is free, how much is being used the Ubuntu Virtual Machine and lastly buff/cache.

The buff/cache is a little tricky and confusing in order to understand this we need to understand buffers and cache. Buffers are generally used by the computer’s memory and is set aside as a temporary holding space for the data that is being sent (generally received from an external device such as a Hard disk drive). Alongside the buffer there is cache which is also a temporary storage area where frequently accessed data can be stored for quick access.

So in order to break this down buff/cache counts the cumulative sum of the buffers and cache. The buff/cache additionally counts memory used for the data that’s on a disk or should end up there soon and as a result is potentially usable for memory. Row 4 all has to do with physical memory (RAM, and ROM and Hard disk drives)

The **fifth row** is what we call swapped memory. Swapped memory is a disk partition the system uses as an extension to your memory. Also known in simple terms as data that was not used and can be swapped to the disk to free up your RAM. Swapped memory is virtual memory in this case which can be a file or partition on your hard drive that is essentially used as extra RAM. **IT IS NOT A SEPARATE RAM CHIP** it resides on your hard drive this partition.

The last part of this is that row which indicates what each attribute is for top. This is displayed in the following image below:



**Figure 5.18:** Breaking down the column headers from the top command.

The columns are very important to understand while using Linux. As you are able to see throughout out the various commands we have been running just understanding the command line interface is imperative to the key to Linux.

**PID:** This is the process ID which is the unique number used to identify a process.

**USER:** The username of whoever launched the process.

**PR:** The priority of the process. Processes with higher priority will be favored by the Linux Kernel. That means these processes will be given more CPU time than processes with lower priority. The only weird thing with this command is the highest priority is actually “-20” mean while the lowest priority is “20”. Hence the negative values are a higher priority than the positive values.

**NI:** This stands for Nice value. The nice keyword is a way of setting your process’s priority.

**VIRT:** this stands for Virtual Memory size (KiB). The total amount of virtual memory being used by a process.

**RES:** The Resident Memory Size (KiB) is non-swapped physical memory a certain task has used.

**SHR:** This is shared Memory size (KiB) this is the amount of memory that reflects memory that could be potentially shared with other processes.

**S:** The process status of the given process. There are multiple values for the process status and they are below:

* “R” stands for Running the process is running.
* “D” stands for uninterruptible sleep (there is no way to get out of this except for rebooting so try not to get any application stuck in the process status D mode.
* “S” stands for sleeping: This means the process is sleeping in a simple sense it is waiting for an action to complete or another external signal to wake it up
* “T” traced or stopped process this means a process has completed a trace or stopped due to a signal.
* “Z” this stands for a zombie process which as we know is when the process completes an execution but still has an entry in memory.

**%CPU-** This stands for the %CPU Usage which is being used by the process. (It is important to note that these are all for single core values not multi core)

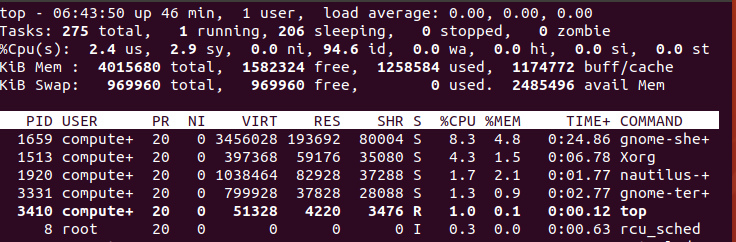
**“%MEM”**- This stands for memory usage which is a tasks available physical memory left (RAM).

“**TIME+”**: This is the total CPU Time to the hundredths of the given task.

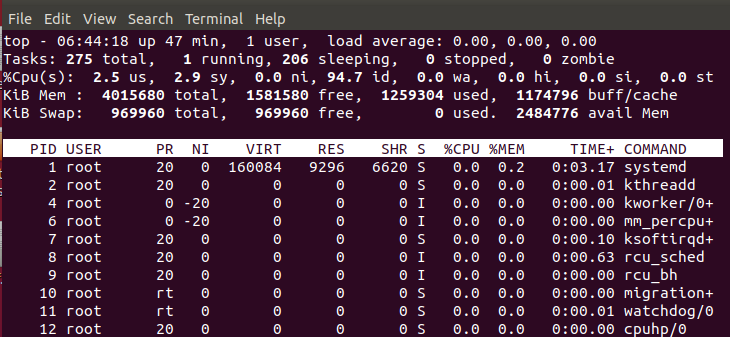
“**Command”:** The command name on the command line interface.

I understand that this whole breakdown of the top command is a lot so definitely feel free to re-read through it and ask questions. After we go through some more with the top command we will answer some questions on it.

After all this talk about the top command we need to go in how to tame the top command. This is done through different **symbols/letters** on your keyboard. For instance if I type **“R”** (note that it is uppercase and not lowercase) within the terminal top instance The top command will sort the output in ascending order of the CPU usage. This means that processes which consumes the least amount of CPU will be shown first. A snippet of it is shown below. The first instance is the top command then the second is after I tapped “R” on the keyboard:

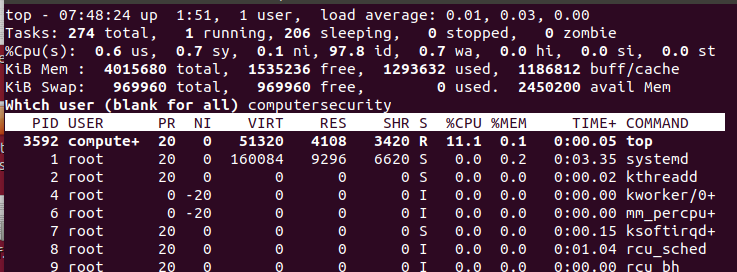


**Figure 5.19:**  The output of the top command

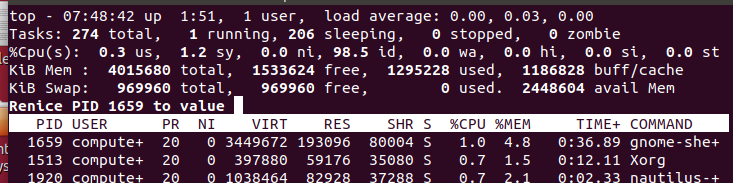


**Figure 5.20:** The output if I type the R keyword.

The next three commands I want to get into are three capital sorting commands for the top command it is using the following capital letters: “P”, “N”, and “T”. If you press the capital “P” on your keyboard it will sort the process list by CPU Usage, if you press the capital “N” it will sort the list by process ID and lastly if we sort by capital “T” then we will sort by running time.

Another helpful command with the top command is being able to view processes of a user for this we can hit the u/U button. For instance in the screen shot’s below I hit the “U” button and then entered in computersecurity. Then in the second screenshot I hit the enter button which shows all the processes being run by the user computersecurity. 

**Figure 5.21:** The output after I tapped “U” on the keyboard while the top command is running.



**Figure 5.22:** After I hit the enter key the processes being run by the user computersecurity**.**

These couple of commands and hints I showed you are the core fundamentals to the top command, the further you get in the command you might realize that this command is suitable for your liking and might want to try **“htop”**. Please do some research on the htop command to see how it works.

**The tar command**

The tar command is used for compressing and decompressing files within Linux. The tar command stands for tape archive which is used by a large number of Linux System administrators to deal with backups. When we compress files from the tar command we get the following files: tar, gzip and bzip. The tar command is one of the most widely used commands to create compressed archive files. This is due to its flexibility in moving files from one disk to another disk or a machine to machine. For more information on the tar command feel free to check out “**man tar”**

The **first** exercise we are going to do is create a tar archive file. We are going to create a tar archive file of our home directory (this is just for simplicity). In order to do this we are going to go through the command and explain how it works part by part.



**Figure 5.23:** Creating a tar file of your entire directory. So in case something happens you have the file all backed up.

The first part of the command is the tar command which as we know allows for a backup of all the files and directories which we specify.

The –cvf flags are the flags which we will commonly use to create tar files (think of a tar file as a zip file except it is in Linux). The three flags addressed here are –c, v and f flag these flags do the following:

The “c” flag creates a new .tar archive file usually this flag will be in most commands when making a .tar

The “v” flag allows us to see the progress or the files in which is going to the .tar file.

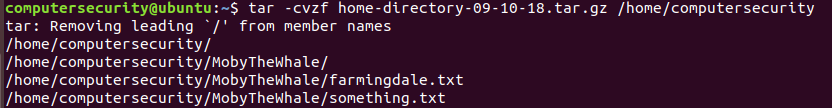
The “f” flag allows us to specify the file name type of the archive file.

After the tar –cvf I wrote “computersecurity-18-0-4-1.tar” this is what we are labeling the .tar file as.

Lastly I wrote /home/computersecurity which is saying that the .tar file we are making “computersecurity18-0-4-1.tar” is going to have all the contents of the /home/computersecurity directory.

Hence the whole command is saying that we are going name a file called “computersecurity-18-0-4-1.tar” and all the contents within the /home/computersecurity directory will go in there.

The **second** thing we are going to do is create a tar.gz archive file. The only real difference between creating a .tar and a .tar.gz file is the file extension and the extra –z flag which creates a compressed gzip fie. For example check out the snippet below:



**Figure 5.24:** Creating a tar file of our home directory but showing which files we are tarring.

Everything is the same in the following image as prior except for a couple things but we will go through the entire command again to re-enforce how the command works.

The tar command allows us to compress and decompress files into certain formats.

The –cvzf flags are illustrated below what they do:

The “c” flag creates a new .tar archive file usually this flag will be in most commands when making a .tar

The “v” flag allows us to see the progress or the files in which is going to the .tar file.

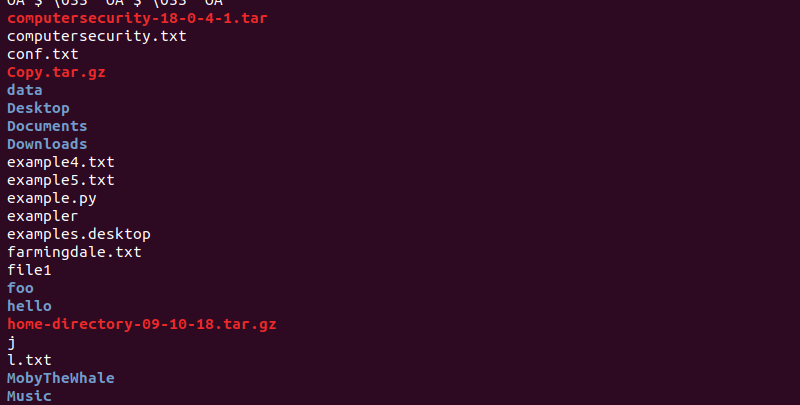
The “z” flag allows us to create the gzip(.gz) extension on our file.

The “f” flag allows us to specify the file name type of the archive file.

The next part of the command is telling us the file name which we are creating “home-directory-09-10-18.tar.gz” this file name will have all the contents of the /home/computersecurity directory which is the next parameter.

Hence in this command we are creating a .tar.gz file (gzip file) which contains the contents of the /home/computersecurity directory.

The **third thing** we are going to discuss is how to untar a .tar file (basically how are we going to extract a compressed file. First we can verify that files are within our home directory by running the ls command. We should seeing something like the following:



**Figure5.25:** Verifying that our computersecurity file is located within the home directory.

For our first example we are going to untar the “computersecurity-18-0-4-1.tar” file into our Downloads directory. I know what you’re thinking that you will have to copy this file into another directory first then untar it. Well you are in luck there is a flag “**-C**” which allows us to untar the file in a different directory.

The command below will show us how to extract the tar file into the Downloads directory:



**Figure 5.26** How to use the –C flag to extract the contents and place them in a directory.

We are going to break down the above command and allow us to below:

The tar command as we know will compress and decompress files and directories.

The following flags –xvf do the following:

The **–x** flag allows the user to extract the files

The **–v** flag allows the user to have a verbose output for the command (note this isn’t always needed and if you get an error thrown it is probably better to leave out the –v)

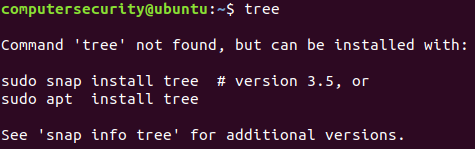
The **–f** flag allows us to specify a file name for the archive.

The next parameter is the name of the file as we know which is “computersecurity-18-0-4-1.tar”

Then second we use the “-C” parameter which enables us to output the contents of the tar folder into another directory and not in our present working directory. After we specify the “-C” flag we then put the parameter of the directory where we are unzipping this and in our case it is into the Downloads directory which is “/home/computersecurity/Downloads”

**The tree command**

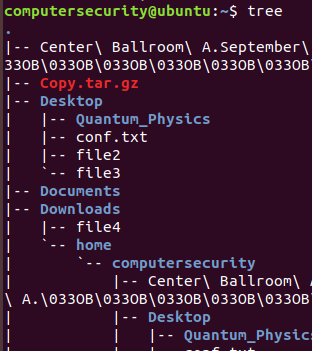
Before we even get into what the tree command is type “tree” into the terminal and see what the output is if you get the following then it is not installed and just follow the on screen instructions:



**Figure 5.27:** Installing tree within the Ubuntu machine

Hopefully you entered in “sudo snap install tree” and entered in your sudo password instance now that you have this all taken care of it is time to define what is the tree command. The tree command is a program which shows a recursive directory listing of different files within different directories. Upon listing out all directories and files within the current directory and respective recursive directories it lists at the bottom the number of directories and files that the command has listed.

For the first command we are going to just type the tree command into our terminal instance nothing else just the “tree” command in our home directory instance. A partial screenshot is listed below of the current output of the command:



**Figure 5.28:**  The output of three command.

**Sort**

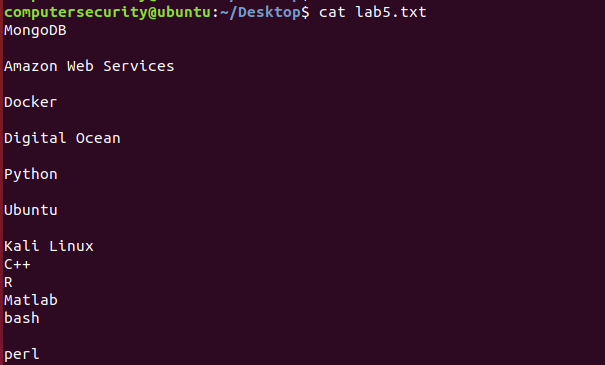
The sort command is a built in command in Linux which allows us to print lines of text files and concatenate different files in a sorted manner. The sort command usually takes in blank space as a field separator and the entire input file as some form of a sort key. The sort command doesn’t actually sort the file for you but only prints the sorted output in order to get the sorted output into a text file we will have to use piping symbol or redirect the output using the “>” symbol that we learned earlier on.

The first thing we are going to do is go into our **Desktop** directory and create a text file named lab5.txt and in this file we will have the following information exactly as I have it to illustrate all the points:



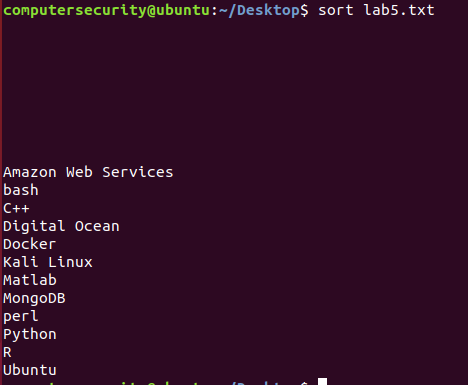
**Figure 5.29:** A text file to use the cat command on.

After we exit and save the file then we can run the cat command on the file lab5.txt in order for us to see the contents of it. When we run the cat command we get the below output and as we can see it is not sorted and needs to be sorted for our case:



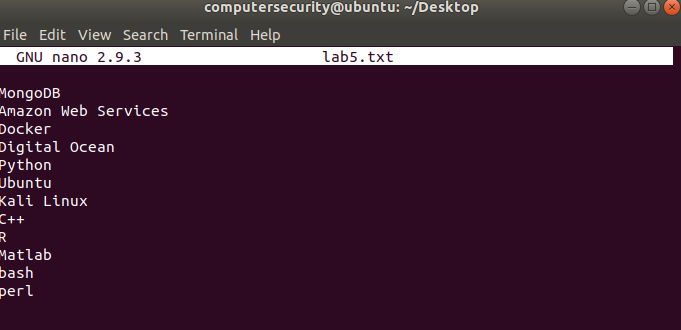
**Figure 5.30:** The output of the cat command

If we run the sort command with the lab5.txt file we will get the following output (note remember that this does not change the file around just the output):



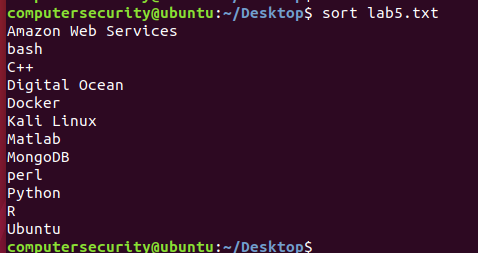
**Figure 5.31:** The sorted file lab5.txt but with a lot of unnecessary white space.

As you can see in the above figure we have everything sorted which needs to be sorted the only downside to this output is the white space which is included in the file output. In order to minimize this we need to get rid of the extra white spaces (the extra lines in between the data). This means that we are going to make our text file look like so:

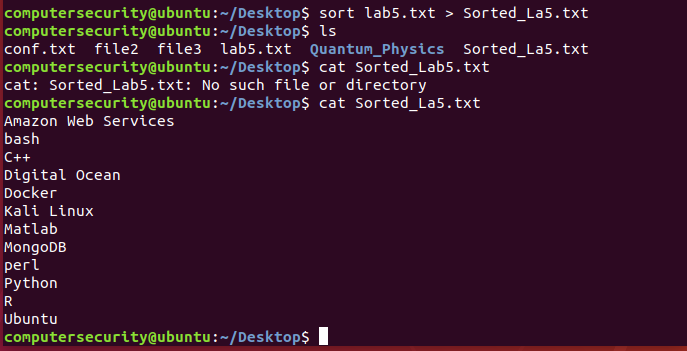


**Figure 5.32:** Edited the file lab5.txt to get rid of white space.

As we can see in the Figure above there is no extra white spaces no extra lines so when we run the sort command again we should get the following output:



**Figure 5.33** Called the sort command on the file again. This was used to show that there is no more white space.

As a note remember the above command will not sort the contents of the text file but just the contents that are being outputted to the terminal. In order to fix this we are going to use for our case the “>” symbol and this will redirect our output to a text file. Let’s take a look at this below:  


**Figure 5.34**: The figure above illustrates how to put sorted contents in another file.

In the above snippet we have the following snippet:

The first line which is “sort lab5.txt >\_La5.txt” what this does is it sorts the data within the lab5.txt file then it outputs to the Sorted\_La5.txt file.

The second line we use the ls command which shows the current files in the directory which we are in.

The third line I made an error and mis typed.

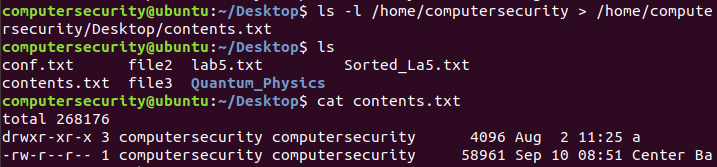
In the fourth line we used the cat command to view our new text file “Sorted\_La5.txt”

In order to summarize the above snippet we sort the data within the text file then we output to a new text file it’s that simple it can be done in two commands.

The next thing we are going to do with the sort command is take a break down of the ls command and put the output into a text file and then sort the ls file. This is useful because maybe for some reason we have so many projects in our directory that we can’t find one and we need to quickly sort through the files we can alphabetically sort the files. As we go through this command you will see that we will be using all syntax and commands we have talked about it but now we are bringing it all together.

In the next command we are going to create a new file called contents.txt this file will be located in the Desktop. Once this file is created we will perform different sorting operations on it in order to see how organized we can really get our text file.

In the snippet below we see how we are able to take an extended **ls** output and dump its contents into a file on our Desktop (contents.txt):



**Figure 5.35:** The first step to sorting through our ls command is dumping the contents in a file.

In the first line we call the ls command and formulate it into a extended output of our home directory. Then after we do this we dump the output of the ls command into a text file called contets.txt which is in our Desktop directory.

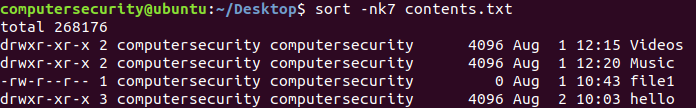
The second line I am already in my Desktop directory I type **ls** to view the files within the directory and I notice the contents.txt file has been made which is good.

Then third line I call the cat command just to see the contents within the file and make sure everything is intact.

Now that we have the contents of this file with in the file contents.txt we are going to try to sort through the file (this might seem tricky at first but you will get the hang of it and if at any point you get confused just remember all I did was use “**man sort”**). Check out the snippets below to see how we are going to sort through this file:

The first two sort flags we are going to look at is the –n flag and the –k flag. These two flags allow for us to sort our files accordingly. For instance the –n flag performs a numeric a string numerical value comparison (AKA comparing integers). The –k flag is a key based sort as well which is based off of KEYDEF. The KEYDEF explanation within the manual page is a little bit confusing so in order to simplify it think of the following. The KEYDEF or the –k flag specifies what column the file is sorted by. For example if the column number contains –k2 then we know we are sorting by the second column. Another usage of the –k flag can be –k2.3 (note the period within the k flag) this means we will sort by the 3rd character within the second column.

That’s enough theory for now let’s execute the command and make sure we understand what is happening when we call the sort command with its respective flags. In order to do this we are going to call the sort command along with the file which we are sorting. The command is below with a partial output following the screenshot we will go into detail about and have some questions. (The best way to learn is by doing and hands on exercises.):



**Figure 5.36:** Sorting the 7th column in the file colntents.txt

The command within the Desktop directory “sort –nk7 contents.txt” sorts the file contents.txt based off of the 7 column and the numbers within the 7th column. (Columns are usually determined by commas or tabs it is dependent on how the user or programmer makes the files). Hence the organization of these files are by the day they were last used.