LINUX COMMANDS

1. **Test drive and understand the usage of all the commands given in the 50 Most Frequently**
   Used UNIX / Linux Commands and linuxcommands.pdf
2. **Create a directory and create a file inside that directory.**
   Working:
   The *mkdir* command simply creates a subfolder in the directory path(s) given with the command. The creation of directories can be made to have file permissions to it, whether it can be changed or viewed using *-m=MODE* option. The MODE is the same as *chmod* permission format.
   The *touch* creates an empty file with the given name.
   There are some installed text editors, these may include: vim, emacs, nano, gedit, etc. all these are there in the form of executable commands and take the names of the files we create as the arguments. Here till these files are saved they aren't added to file system but they stay in RAM
   Code:
   mkdir dName
   nano ./dName/hello.txt
   #write and save the file
   Or
   mkdir dName
   touch ./dName/hello.txt
3. **List the files and directories that are empty in a working directory.**
   There is not any command for finding for looking for subfiles of all folder directly, so going around the problem, here what I did was find going to present working directory by '*.*' and inside these files if search for '*-type d*' that is a directory, particularly type of d I go for is the *-empty* type.
   Code:
   find ./ -empty
4. **Show commands to delete empty and non-empty directory.**
   The *rmdir* command is used to remove an empty directory and if the complete path leading from the working directory to the empty directory is empty then the complete path can be removed by *-p*. The directory the is only delete if it is empty i.e it only deletes one type of files from the memory- the directories.
   Empty directory removal:
   Code:
   rmdir directoryName
   The *rm -rf* options is used to recursively and forcefully delete the file. Forcefully in the sense that if there is no directory by the given name then don't show error message and recursively in the sense that if the file or in regular cases directory contains some other files or so components delete them, here we say recursively so we mean to go down to every directory and within it and within it and within its sub-directories and so on, hence recursively till we erase the files from memory.

Non-Empty directory removal:
Code:
rm -rf directoryName

5. **Find the location of the input files using locate and find command.**
   Code:
   *find * sample1.txt*
   *find * sample2.txt*
   locate sample1.txt
   locate sample2.txt
   The *find* command is used to find the existence of the file named, here '*sample1.txt or sample2.txt*', in the argument inside of the path given. The search can be further
6. **View the user permissions and ownership of the files in the current directory and change the ownership of some selected files to another user.**
   For viewing the permissions of files, along with relevant information like which permission belongs to whom and when the file was last modified, in the directory is done using *ls -l* shows the above details about files but only of those files that are not hidden, the option *-a* for *ls* does the job of showing all files.
   Understanding the output:
   Code:
   ls -al
   Output:
   total 16
   drwxr-xr-x 2 anant anant 4096 Aug 15 14:19 ./
   drwxrwxr-x 4 anant anant 4096 Aug 15 13:18 ../
   -rw-r--r-- 1 anant anant 4150 Aug 15 14:19 Ex2.txt

   Here instead of passing an argument for the path to some directory we pass nothing and by default it takes the present working directory as the path, we can also pass arguments as regular expression or multiple arguments such.
   The other information in the output like *'drwxr-xr-x'* can be understood as a information on the file d-directory, r-read permission, w-write permission and x- executable, these repeat due to them being told for the different users, here they are owner-group-others.
   The 'total' here is the number of blocks of memory given to the directory.
   2nd column: Number of link to this file
   3rd column: Name of the owner of the file/directory
   4th column: The group owner of the file/directory
   5th column: The size of the file/directory in the bytes unit.
   6th column: The last time and date the file is modified
   7th column: The filename or directory nam

7. **List all the files in the current directory and subdirectories.**
   Code:
   ls -R

The *-R* is nothing but the recursive, here the file as displayed recursively in the sense that the files for current path are shown then if these contain directories then these directory sub-files are shown and then the directories within these so on and so forth. To limit this to the level 2 (as in only subdirectories of current directory there sub-directory and no further) we can do this :

Code:

ls .//*

The above expression is a regular expression used to pass the names of the current subdirectories and their children.

8. **Concatenate the two input files: "sample1.txt" and "sample2.txt" and save it to a new file named "Input".**

   The *cat* command is the acronym for concatenate, the command deals with stream redirection. The default streams to redirect the input and output are stdin and stdout, respectively. The command can access the files as streams and concatenate, read, overwrite the file depending on usage.

   cat man page: 'With no FILE, or when FILE is -, read standard input.'

   The concatenation of these files can be done by:

   Code:

   cat sample1.txt sample2.txt > Input

   This will concatenate the files before the stream redirection operator > in the order they are written. Here the > operator is used to create a new stream (or rewrite the previous one) that we name 'Input'.

9. **Copy the contents of file 'sample2.txt' to 'sample.txt'**

   cp sample2.txt sample.txt

   OR

   cat sample2.txt > sample.txt

   These two will do the same work of copying the 'sample2.txt' to 'sample.txt'

10. **Append the file contents of input file 'sample2.txt' to the end of the first input file 'sample1.txt'.**

    cat sample2.txt >> sample1.txt

    To append to an existing file by accessing an existing stream using >> this will add the file contents from before to the end of the file of the other file.

11. **Remove the permission for the users to read, write and execute the file 'sample.txt'.**

    chmod can be expanded as 'change mode', here the mode will refer to the file permission given to different entities, those being user, group and other groups.

    This will give permission of rw for all three,, this can be understood by 3-bit representation of the number

    rwx= 3-bits of permission given to the respective user, one represents that the permission is given.

    We can specify the owner permission by

    u=wrx

    Group permissions by

go=wx

Other group permission by

og=w

Option using the same chmod command, we can change all of them together by

a=+w

Here the '=+' signifies addition of w permission to existing permission where as '=' is used for overwriting the permission.

Similarly there is =- for removing a permission r, w or,x.

Code

chmod u=-rwx sample.txt

12. **Display the current date with the day of week, month, time and the year.**

    The *date* command is used to find the current date, the options to the date command are all with respect to the current date.

    Code:

    date

13. **Show the calendar of previous, current and next month.**

    The *cal* command will show the calendar of the current month. There are options like showing the 12 month calendar for the year given in the argument and so on.

    For the given question the command:

    Code:

    cal -3

    would result in getting the previous, current and next month calendar.

14. **Sort the contents of the file 'sample1.txt' in alphabetical order.**

    The file's words can be sorted using *sort* command passing the filename as the argument.

    Code:

    sort sample.txt

    The options to sort vary greatly due to versatility of sort, the sort can be stable sort, dictionary ordering, ascending or descending order sort, etc. The sorting provides a variety but this in turn needs a proper inputs and options of sorting, like we give the input as space separated integers the output, the output will be the dictionary based sort on the elements, the sort for numerical values require option *-n* , thus there needs to be a proper understanding of the options and sort commands to get the output as required.

15. **Erase duplicate records in the file 'sample1.txt' and display only the unique records**

    The use of command *uniq* will return unique consecutive lines in the file. To get the unique lines out of all we need the sorted file and then use *uniq* on it. This can be done by creating a new file and then using *uniq* on this file or using pipes.

    Pipes are used to redirect a stream from one program to another.Though the functionality of the pipe appears to be similar to that stream redirection operators the distinction is that pipes redirect data from one command to another, while > and >> are used to redirect exclusively to files, or from files.

    Inappropriate usage:

sort sample1.txt > uniq

Here, using > or >> here will result in formation or appending (respectively) to the file named ‘uniq’ not the unique elements in file ‘sample1.txt’

The desired result would come from:

Code:

sort sample1.txt | *uniq*

16. **Add line numbers to the file ‘sample2.txt’**

    There are multiple ways to get the line number to an existing file, the easiest or rather a command solely made for this is the *nl* command. This command even provides us with options like leaving the blank lines and adding ‘.’ after every line number.

    Code:

    nl sample2.txt > newfile

    cat newfile > sample2.txt

    Here the creation of 'newfile' was necessary due to the fact we need to add to the same file.

    For this above to be justified there needs to be more explanation behind the working of ‘>’

    Say we do:

    cat file1 > file2

    The shell does is create an output stream for ‘file2’, even if ‘file2.txt’ already exists then the file, the shell does is it truncates it, and then starts ‘cat file1’ for reading, now the transfer from ‘file1’ is created. But in the case of ‘*cat file1 > file1’* the ‘file1’ is already lost after being truncated by ‘>’ and hence an empty ‘file1’.

    Opening of file for both read and write access by the same command may lead to corruption of file and is usually avoided in practice.

17. **Find out whether the two pairs of input files are identical or not.**

    Compare sample1.txt and sample2.txt

    Compare sample2.txt and sample.txt

    We can check manually the contents of files are same or different but this fails in case of large files, the linux command pack comes with a command especially for this i.e the *diff* command.

    The *diff* command is used to differentiate contents of the file. The output is not as direct as expected, the output contains symbols {*a, d, c,<, >*} . The meaning of these symbols is add, delete, change, line from first file and line from second file. The output comes in the format of line numbers, symbols and line strings.

    The *diff* command itself has many modes, these include *-q* where there will be no output if the command there is no different in the input files, *-s* report if identical

    Code:

    diff sample1.txt sample2.txt

    diff sample2.txt sample1.txt

18. **Show how the input file “sample1.txt” differs line by line from “sample2.txt” in context and unified mode.**

    diff -c sample1.txt sample2.txt

diff -c sample1.txt sample2.txt

The context mode is *-c* and unified mode *-u*.

I have not pasted the screenshot of the command as I did it in the end and I don't think the output expected will be there.

19. **Solve the arithmetic expression: ((8+12)*(5-3))/2 using linux commands,**

    Code:

    echo $((((8+12)(5-3))/2))*

    The above expression would result in outputting the value of the expression to stdout. The echo is to output a string or variable. Here '(( ))' is used to evaluate the expression inside the brackets, the $ is used to convert the output of the evaluated expression to string and pass it to echo.

    If we don't echo the result then the shell will treat it as a string like any other command, ls, cp, etc. This in this particular case will lead to the error: 'command not found'

20. **Cut and display the first 10 characters of every line of the file "Input.txt".**

    cat Input | cut -c1-10

    The cat here would give the text output of the complete, which through the pipe is passed to cut command as the input, now this in turn allows as to print only 10 characters or more like 10 bytes.

    The other way to do this is

    cat -c1 -10 Input

    This does the same thing as above but here we avoid using 2 commands and get it smoothly in one executable.

21. **Print the name of the current working directory.**

    pwd

    This command can be expanded as "Present Working Directory". It does exactly as said and returns a path from root to the current working directory.

22. **Process Status**

    a. List all the running processes with their corresponding PIDs.

    This can be done by command *ps* ("Process Status") which is used to get the list of all processes currently running.

    ps

    This returns a table of process and details related to them like process id, time of start, who is running the process (gnome, bash, etc), etc. All these are returned by the command *ps*. Further details like from where the process is actually run from the root directory, who started it, etc can from various options in *ps*, the one I talk of is *ps -aux*.

    b. List the processes that are not associated with the terminal.

    ps -a

    c. List the processes that are associated with the terminal.

    ps -T

23. **Print the number of characters, number of lines and number of words all the given input files.**

To do this we simply have to pass the input files as arguments to the *wc* command, this command returns lines, words and characters along with corresponding file name as output.

wc input

To particularly look for line *-l*, words *-w*, characters *-c* and bytes *-m*.

24. **Print the length of the longest line from all the input files.**

    wc -L input

    This option *-L* would return the length of the longest line

25. **Move the contents of the input file sample.txt to a new file.**

    mv sample.txt newfile.txt

    The above is actually moving the file from 'sample.txt' to 'newfile.txt' and this results in renaming of the file if the file is moved to the same directory.

26. **Copy the contents of one directory to another directory.**

    cp -r ./Practise/ ./dName/*

    This will copy the contents of the directory from folder 'Lab2' to 'Lab 2'. *-r* will make it be done recursively to ensure even the subdirectories get copied.

27. **Reverse the lines of the two input files and concatenate the file contents using a single command.**

    tac sample1.txt sample2.txt

    The can be considered the same as *cat* only here *tac* start reading from the last line instead of the first line. So as we would make a 'newfile' from previous inputs files in *cat* we do the same in *tac*.

28. **Delete all the files with *.txt extension from the working directory using yes command.**

    *yes | rm -i *.txt*

    The *yes* commands repeatedly prints a string that is passed to it, if there was no string passed then the default string is 'y', here we would need 'y' from user before every deletion as the mode of *rm* chosen is interactive *-i*.

    A better way to do the same is not using interactive *-i* mode at all.

29. **Given the input file "sample1.txt", print the number of the lines that match the pattern "system".**

    grep system sample1.txt | wc -l

    grep stands for Global Regular Expression Print, this allows the user to search for a pattern, usually regular expression or a string in the file(s) passed as arguments. Due to the wide use of *grep* command comes in with various options like *-i* for case insensitivity, etc. The output of *grep* is the lines in which the pattern appears so to get the number of lines we need to redirect the output to *wc -l*.

    The using option of grep like *-c* for counting the number of matches will do the same job but without the need of piping

    grep -c system sample1.txt

30. **Having sample1 file as input, print the matched lines that contain the pattern "Unix" as whole words.**

    grep -n Unix sample1.txt

The *-n* return the line number and corresponding lines that match the pattern.

31. **Print the lines from “sample1.txt” that do not match the pattern “OS”.**
    grep -v OS sample1.txt
    The *-v* can be thought of as inverting the *grep* output, here the lines that don’t contain the pattern are printed.
32. **Fetch the files that contain the word “OS”, “Operating System”, “Operating Systems” with its respective line number. (Ignore the case).**
    *grep -lei “OS” -ei “Operating System” -ei “Operating Systems” ./**
    Option *-e* for expression, *-i* for case insensitivity and *-l* in the end to get the files corresponding to the pattern rather than the lines in files.
33. **Having “sample1.txt” and “core” as the input and pattern respectively, along with the matched line print three lines before and after the pattern match.**
    grep -3 core sample1.txt
    The *-3 is* for getting the three lines above and below the pattern.
34. **Find and replace the string “OS” with “Operating System”.**
    sed -i ‘s/OS/Operating Systems/g’ FILENAME
    sed is the stream editor, *awk* and *sed* can mostly be used for the same operation in different ways.
    -i: inplace replacement (in this question it wasn’t specified whether to inplace replacement or not, to remove this remove option *-i*)
    ‘s/old_pattern/new_pattern.g’: **s**ubstitute **g**lobally the old_pattern to new_pattern.
35. **List only the text files in the current working directory with its corresponding disk space occupied.**
    stat returns the file statistics like memory used, date of file creation, date last modified, accesses, links, etc.
    *stat *.txt | awk '{if(($1=="Size:") || ($1 == "File:")) {print}}'*
36. **Show the last modification time of all the input text files.**
    If we want only the date last modified then we have to filter the output to of the *stat* command:
    *stat *.txt | awk '{if(($1=="Modify:") || ($1 == "File:")) {print}}'*
37. **Delete the line that has the word “Powerful” from the text file “sample2.txt”.**
    sed -i ‘/powerful/d’ sample2.txt
38. **Print the roll numbers that end with even numbers in the format (COE18B002) up to COE18B050.**
    for i in $(seq -w 2 2 50); do echo COE18B0$i; done
    The command *seq* is used to get a sequence, it is used as:
    seq [OPTION]... FIRST INCREMENT LAST
    The *-w* is for equal width here.
    for is the looping statement where the *seq* output is put in the *i*.
    echo is used to print expressions to stdout.
39. **Use filter commands like head, tail, more to view the file contents page by page.**
    head -<number_of_lines> <file>
    tail -<number_of_lines> <file>

The above two are used this way to display the <*number_of_lines*> in the <*file*>.

more <file>

less <file>

The command *more* is used to display the contents of <*file*> page-wise, here we get the output in stdout.

The command *less* is used to display the contents of <*file*> like the *more* command but the instead of displaying it the stdout this command displays it in a new interactive session in the terminal, much like the *man* page sessions.

less is being used more than *more.*

40. **Compress the current working directory contents to a tar file and extract those files from the compressed tar file.**

    tar cvf new.tar ./

    tar xvf new.tar

    The *c* to create the .tar and *x* to extract from the tar.

41. **Compress the files using zip command.**

    a. Zip the input file "sample1.txt" as samplezip.zip and remove the file from the current directory after zipping.

    zip -o samplezip.zip sample1.txt

    b. Add "sample2.txt" and update the zip archive.

    zip -u samplezip.zip sample2.txt

    c. Zip a directory with all its contents.

    zip <zip_name>.zip <directory_path>

    d. Remove a file from the zip archive

    zip -d <zip_file_name>.zip <file_to_remove>

    e. Unzip the contents from samplezip.zip

    unzip <zip_file_name>.zip