**PRACTICAL FILE**

**BE (CSE) 4th Semester**

**OPERATING SYSTEM**

**Jan 2024 – May 2024**

**Submitted By**

**Aarzoo Khunger**

**Roll Number: UEM223116**

**Submitted To**

**Dr. Mamta Juneja**

**Professor, Computer Science and Engineering**



**Computer Science and Engineering**

**University Institute of Engineering and Technology**

**Panjab University, Chandigarh – 160014, INDIA**

**INDEX**

|  |  |  |
| --- | --- | --- |
| **S.NO** | **PRACTICAL** | **DATE** |
| 1 | To implement the commands:  cat,man,echo,touch,ls,mkdir,cd,cp, pwd,tty,who,wc,mv, rmdir,rm,script | 18/01/2024 |
| 2 | To implement the commands:  whatis , whereis,find,type,bc,expr, diff,cmp,comm,sort,tee,cut,tr, grep,head,tail,free,df,du,ulimit, cal,ncal,pipe(|) | 25/01/2024 |
| 3 | To implement the commands:  umask , chmod, adduser,su, deluser,gzip/gunzip,tar, split,sleep,shutdown,chown,semicolon(;) | 01/02/2024 |
| 4 | To implement the commands:  ps,nohup,kill,nice,batch, crontab,wall,write | 08/02/2024 |
| 5 | Shell scripting  → Introduction to vim editor  → Syntax of control structure | 15/02/2024 |
| 6 | Programs:  Addition of two numbers  Check whether the number entered is odd/even  Check whether the year is leap or not  Calculate the average salaries of employees  Calculate the average marks of students  To find circumference and area of circle  To find the sum of digits of a number.  Find the factorial of a number  Find the fibonacci series  Check whether a number is prime or not  Check whether the number is palindrome or not  Reverse of a given number  To create a binary calculator using switch case  Linear Search  Binary search  Bubble sort | 22/02/2024 |
| 7 | To implement the following String Operations programs in vim editor:  i) Find length of string  ii) Compare two strings  iii) Reverse of a string  iv) String is palindrome or not  v) Occurrence of a pattern  vi)Concatenate strings  vii) Convert lowercase letters into uppercase. | 29/02/2024 |
| 8. | To implement the following File Operations programs in vim editor :  i) To see permissions of a file after checking whether it is file or directory  ii) Rename file  iii) Take backup of a directory  iv) Compare directories and delete duplicate files  v) To check the number of files through command  line arguments  vi) If number of files is even, then copy first argument in second and so on, else display odd  vii) Number of arguments.  viii) Convert lowercase letters into uppercase.  ix) To calculate length of a file | 07/03/2024 |
| 9. | Write a C++ program to implement the following CPU scheduling algorithms :  First Come First Serve(FCFS)  Shortest Job First(SJF) (P, NP)  Priority Scheduling (P, NP)  Round Robin(RR) | 14/03/2024 |
| 10 | C++ programming to implement Banker’s Algorithm:    Safety Algorithm-  Request and Resource Allocation | 21/03/2024 |
| 11 | Write C++ programs to implement the following Memory management algorithms (MVT) :    First Fit (FF)  Best Fit (BF)  Worst Fit (WF) | 21/03/2024 |
|  |  |  |

**PRACTICAL – 1**

**Commands :-**

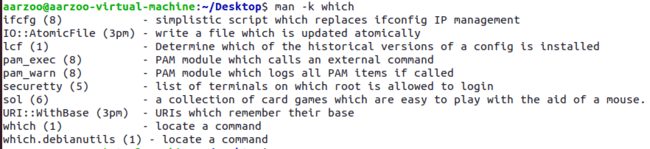
1. **man (manual):**

It is used to display the user manual of any command that we can run on the terminal. It provides a detailed view of the command which includes NAME, SYNOPSIS, DESCRIPTION, OPTIONS, EXIT STATUS, RETURN VALUES, ERRORS, FILES, VERSIONS, EXAMPLES, AUTHORS and SEE ALSO.



**COMMON OPTIONS IN man COMMAND**

* **-k :** This option searches the given command as a regular expression in all the manuals and it returns the manual pages with the section number in which it is found.



* **-f :** One may not be able to remember the sections in which a command is present. So this option gives the section in which the given command is present.



* **-a :** This option helps us to display all the available intro manual pages in succession.



* **-w:** This option returns the location in which the manual page of a given command is present.



* **Section-num :** Since a manual is divided into multiple sections so this option is used to display only a specific section of a manual.

****

1. **echo Command**

The echo command can be used for displaying a line of string/text that is passed as the **arguments**

echo [option] [string]



**COMMON OPTIONS IN echo COMMAND**

* **\b :** It removes all the spaces in between the text



* **\c :** It suppress trailing new line with backspace interpretor ‘-e‘ to continue without emitting new line



* **\n :** his option creates new line from where it is used.



* **\t :** This option is used to create horizontal tab spaces.



* **\v :** his option is used to create vertical tab spaces.



* **\a :** Alert return with backspace interpretor ‘-e‘ to have sound alert.



* **-n :** This option is used to omit echoing trailing newline



* **echo\* :** This command will print all files/folders, similar to ls command .



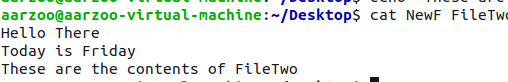
1. **cat Command**

Cat(concatenate) command is very frequently used in Linux. It reads data from the file and gives their content as output. It helps us to create, view, concatenate files. So let us see some frequently used cat commands.

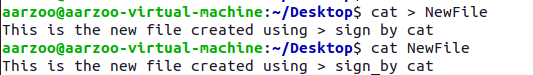
cat fileName



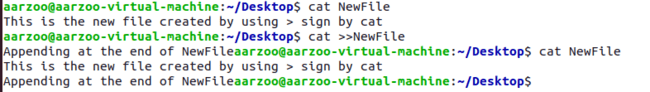
To **display the content of multiple files** at once, type file names in one single line like **"cat file1 file2 file3... fileN.**



The 'cat' command can be used to **create a new file** with greater than sign **(>)**.



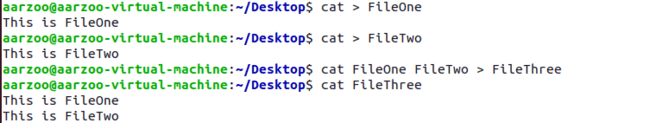
The 'cat' command with double greater than sign **(>>)** **append** (add something in the last of a file) something in your already existing file.



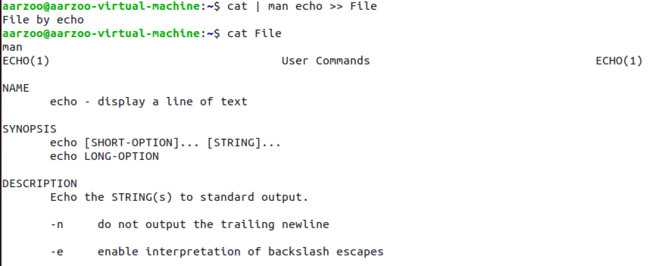
The 'cat' command can be used **to copy the content of a file into another file**.



The 'cat' command can be used **to concatenate** the contents of multiple files in a single new file.

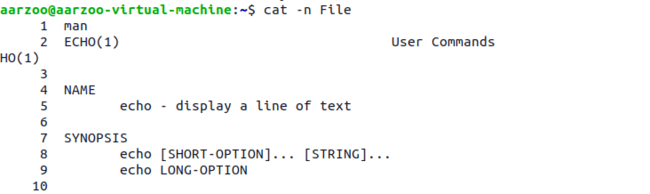


Re-directing output of one command to file using | & >> operator.

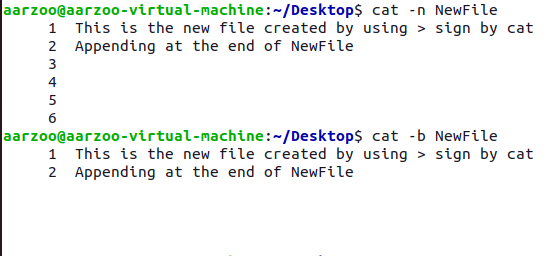


**COMMON OPTIONS IN cat COMMAND**

* **-n :** Number all output lines.



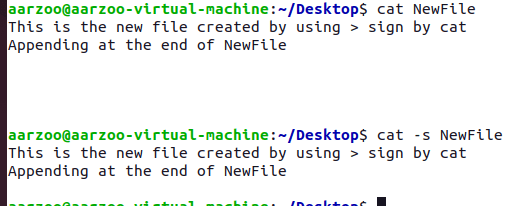
* **-b :** The 'cat -b' option removes the empty lines.



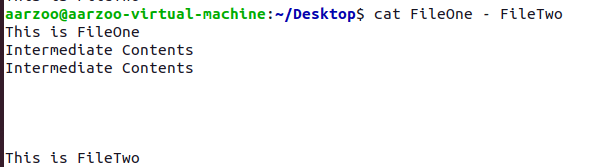
* **-e :** The 'cat-e' option displays a **'$'** sign at the end of every line.



* **-s :** Suppress repeated empty output lines. The -s option will ensure that if there are multiple consecutive empty lines in the file, only one empty line will be displayed.



* **Cat f - g :** Outputs f’s contents, then standard input, then g’s contents. Here, the contents of ‘hello’ will be displayed first, then any input you type will be shown, and finally, the contents of ‘world’ will be displayed.

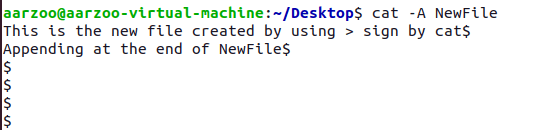


* **-A :** The -A option in the cat command is equivalent to combining the -v, -E, and -T options

-v: Displays non-printing characters (control characters) using the ^ notation.

-E: Appends a $ character at the end of each line.

-T: Displays tab characters as ^I



1. **touch Command**

The touch command is a standard command used in UNIX/Linux operating system which is used to create, change and modify timestamps of a file.



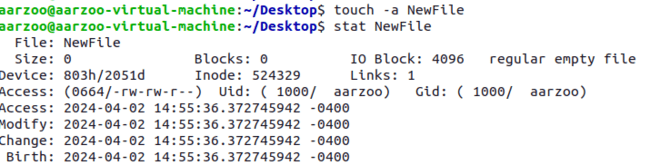
If the file already exists, it opens the file in write mode, and the timestamp of the file is updated.

The Command below creates three empty files at once using the touch command and you can create as many files you like.

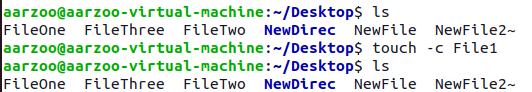


**COMMON OPTIONS IN touch COMMAND**

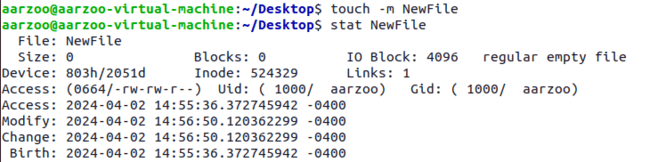
* **-a :** This command is used to change access time only. To change or update the last access or modification times of a file touch -a command is used.



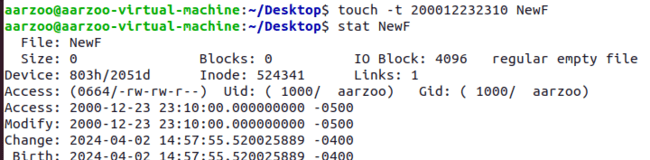
* **-c :** This command is used to check whether a file is created or not. If not created then don’t create it. This command avoids creating files. Here, we wanted to create file 'movie' but with 'c' option no file has been created



* **-m :** This is used to change the modification time only. It only updates last modification time.



* **-t YYYYMMDDHHMM :** This is used to create a file using a specified time.



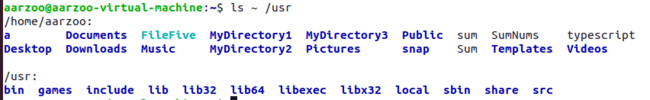
1. **ls Command**

To see a list of files and subdirectories contained in the current working directory. 

Besides the current working directory, we can specify the directory to list, like so:



Or even specify multiple directories. In this,we will list both the user's home directory (symbolized by the “~” character) and the /usr directory.

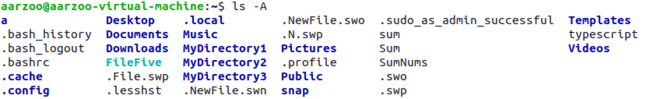


**COMMON OPTIONS IN ls COMMAND**

* **-a** (--all) **:** List all files, even those with names that begin with a period, which are normally not listed (i.e., hidden).

****

* **-A** (--almost-all) **:** Like the -a option above except it does not list . (current directory) and .. (parent directory).

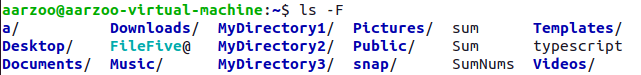


* **-d** (--directory) **:** Ordinarily, if a directory is specified, ls will list the contents of the directory, not the directory itself. Use this option in conjunction with the -l option to see details about the directory rather than its contents.



* **-F** (--classify) **:** This option will append an indicator character to the end of each listed name.

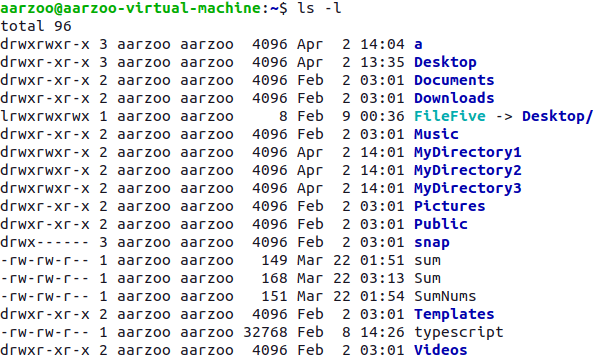
For example, a “/” if the name is a directory



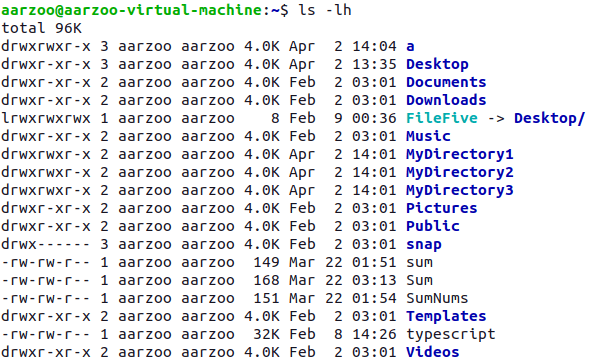
* -**h** (--human-readable) **:** In long format listings, display file sizes in human readable format rather than in bytes



* **-l :** Display results in long format

****

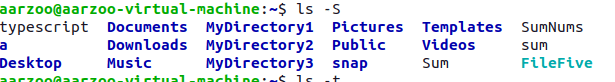
* -l**h:** With combination of -lh option, shows sizes in human readable format.

****

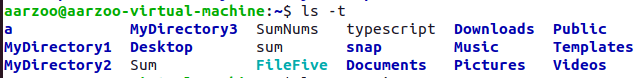
* **-r** (--reverse) **:** Display the results in reverse order. Normally, ls displays its results in ascending alphabetical order.



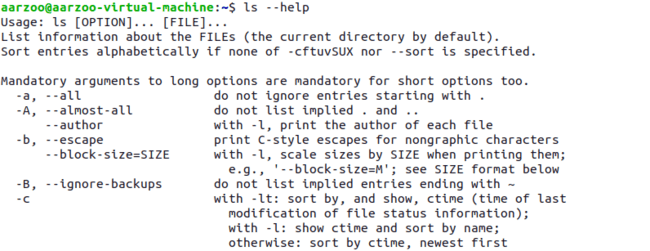
* **-S :** Sort results by file size



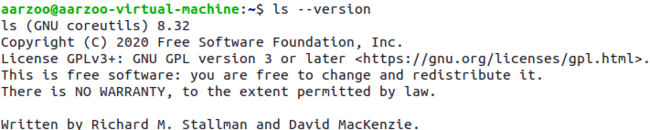
* **-t :** Sort by modification time

****

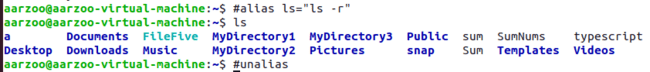
* **--help :** List help page of ls command with their option.



* **--version :** Check version of ls command.



* **# alias ls=”ls -r” :** We have made alias for ls command, when we execute ls command it’ll take -r option by default and display directories in reverse order.



1. **mkdir Command**

The mkdir command is used to create directories. It works like this:

Creates a single directory with the name MyDirectory1.

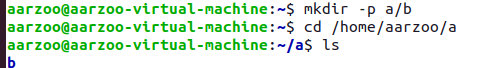


Creates 3 directories with the name MyDirectory1, MyDirectory2, MyDirectory3.



**COMMON OPTIONS IN mkdir COMMAND**

* **-p :** parents or path, will also create all directories leading up to the given directory that do not exist already. For example, mkdir -p a/b will create directory a if it doesn't exist, then will create directory b inside directory a. If the given directory already exists, ignore the error.



1. **cd Command (Changing the current working Directory)**

cd command in linux known as change directory command. It is used to change current working directory.



cd.. – Changing to the directory’s parent directory.

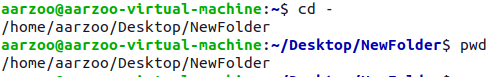
The "." symbol refers to the working directory and the ".." symbol refers to the working directory's parent directory.



**cd** Changes the working directory to your home directory .



**cd -** Changes the working directory to the previous working directory.



**COMMON OPTIONS IN cd COMMAND**

* **-L :** If you tell cd to move into a "directory", which is actually a symbolic link to a directory, it moves into the directory the symbolic link points to.



* **-P :** Use the physical directory structure without following symbolic links. In other words, only change into the specified directory if it actually exists as named; symbolic links will not be followed. This option is the opposite of the -L option, and if they are both specified, this option will be ignored.



* **-e** **:** If the -P option is specified, and the current working directory cannot be determined, this option tells cd to exit with an error. If -P is not specified along with this option, this option has no function.



1. **cp Command**

cp stands for copy.

This command is used to copy files or group of files or directory. It creates an exact image of a file on a disk with different filename.cp command require at least two filenames in its arguments.

**SYNOPSIS**

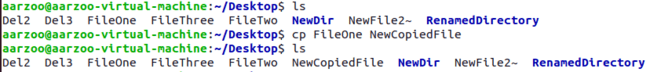
cp [OPTION] Source Destination

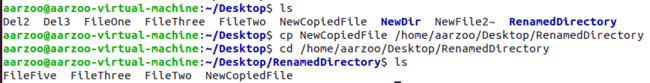
cp [OPTION] Source Directory

cp [OPTION] Source-1 Source-2 Source-3 Source-n Directory.

Using first and second syntax,we can copy files from source to destination,while third

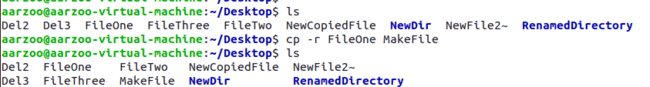
syntax copies multiple files into the directory.



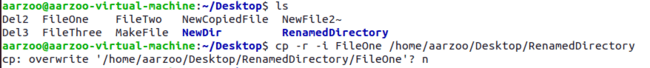


**COMMON OPTIONS IN cp COMMAND**

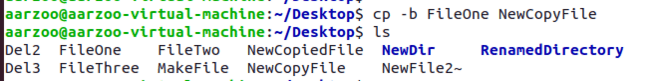
* **-r :** With this option cp command shows its recursive behavior by copying the entire directory structure recursively.



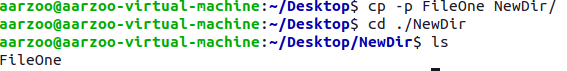
* **-i :** With this option system first warns the user before overwriting the destination file. Cp prompts for a response, if you press y then it overwrites the file and than y other option leave it uncopied..



* **-b :** With this option cp command creates the backup of the destination file in the same folder with the different name and in different format.



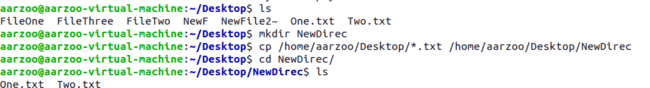
* **-p :** preserves the following characteristics of each source file in the corresponding destination file: the time of the last data modification and the time of the last access, the ownership (only if it has permissions to do this), and the file permission-bits.

****

* **-v , --verbose :** Explain what is being done.

****

* **Wildcard,\* :** It is used to copy all the files of the same syntax, recursively.



1. **pwd Command**

‘pwd’ stands for ‘Print Working Directory’.

It prints the current working directory or simply the directory user is, at present.

It prints the current directory name with the complete path starting from root(/).

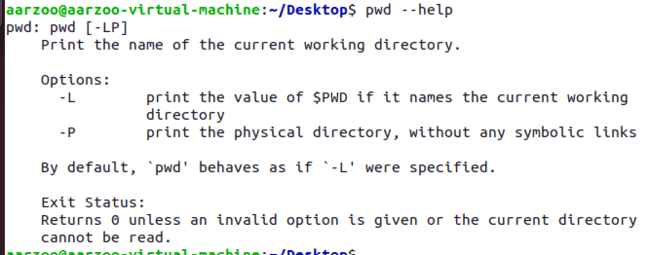


**COMMON OPTIONS IN pwd COMMAND**

* **-L:** Use PWD from environment, even if it contains symbolic links. ****
* **-P:** Avoid all symbolic links.

****

* **--help:** Display the help and exit.

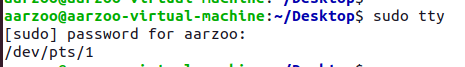
****

1. **tty Command**

The tty command of terminal basically prints the filename of the terminal connected to standard input.

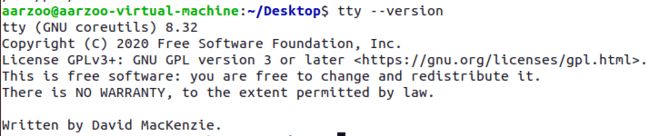
Tty is short of teletype,but popularly known as a terminal it allows you to interact with the system by passing on the data(youinput) to the system,and displaying the output produced by the system.

tty [OPTION]...

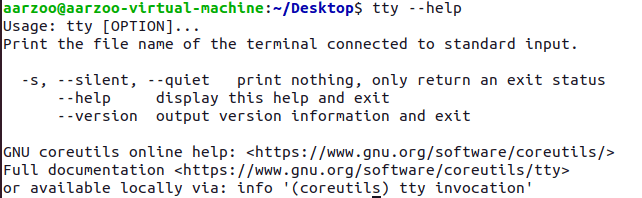


**COMMON OPTIONS IN tty COMMAND**

1. **--version :** Prints the version of tty



* **--help :** It will display the help message and exit.



1. **-s, --silent , -quiet :** Prints nothing, only returns an exit status.



1. **who Command**

Whenever a user needs to know about how many users are using or are logged-in into a particular Linux-based operating system, he/she can use the "who" command to get that information

who [options] [filename]



* who command is used to find out the following information :
  + Time of last system boot
  + Current run level of the system
  + List of logged in users and more.
* The who command is used to get information about currently logged in user on to system.

1. **whoami Command**

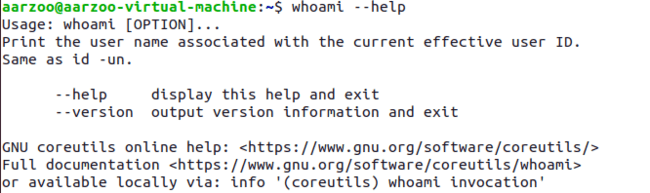
print effective userid

whoami [options]

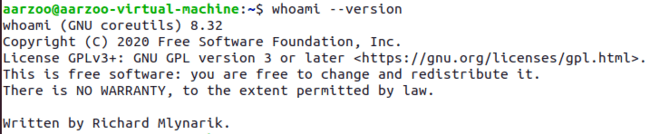


**COMMON OPTIONS IN whoami COMMAND**

* **--help :** Display help and exit

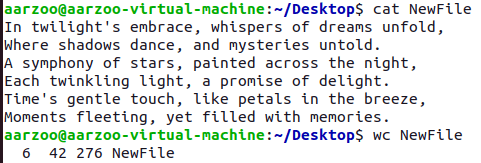


* **--version :** output version information and exit

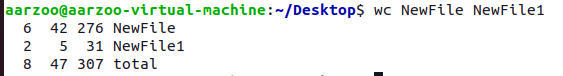


1. **wc Command**

* wc stands for word count.
* It in counting the lines, words, and characters in a file. It displays the number of lines, number of characters, and the number of words in a file. Mostly, it is used with pipes for counting operation.
* By default it displays four-columnar output.
* First column shows number of lines present in a file specified, second column shows number of words present in the file, third column shows number of characters present in file and fourth column itself is the file name which are given as argument.



To display the complete count information of multiple files at once, specify the file names after space.



**COMMON OPTIONS IN wc COMMAND**

* **-l :** This option prints the number of lines present in a file.With this option wc command displays two-columnar output,1st columns hows number of lines present in a file and 2nd itself represent the file name.



* **-w :** This option prints the number of words present in a file.With this option wc command displays two-columnar output,1st column shows number of words present in a file and 2nd is the file name.



* **-c :** This option displays count of bytes present in a file. With this option it display two-columnar output,1st columns hows number of bytes present in a file and 2nd is the file name.



* **-m :** Using -m option ‘wc’ command displays count of characters from a file.



* **-L :** It can be used to print out the length of longest (number of characters) line in a file.

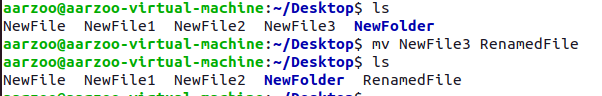


1. **mv Command**

mv stands for move.

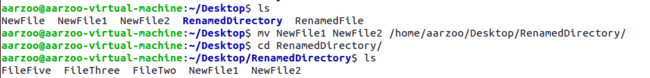
mv is used to move one or more files or directories from one place to another in file system like UNIX. It has two distinct functions:

It rename a file or folder.



It moves group of files to different directory. This command normally works silently means no prompt for confirmation.

* Moving Multiple Files to another directory.

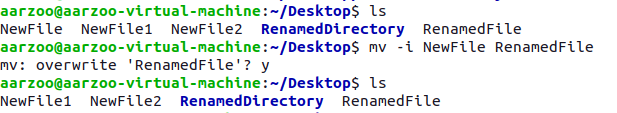


* Renaming a Directory

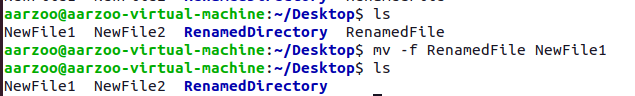


**COMMON OPTIONS IN mv COMMAND**

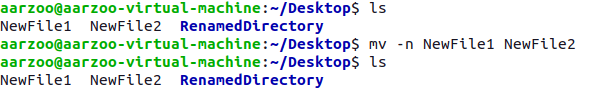
* **-i :** -i option makes the command ask the user for confirmation before moving a file that would overwrite an existing file, you have to press y for confirm moving, any other key leaves the file as it is.



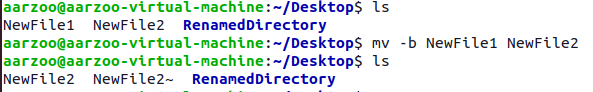
* **-f :** -f option overrides this minor protection and overwrite the destination file forcefully and delete the source file.



* **-n :** mv prevent an existing file from being overwritten.

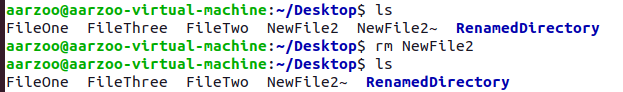


* **-b :** With this option it is easier to take a backup of an existing file that will be overwritten as a result of the mv command. This will create a backup file with the tilde character(~) appended to it.



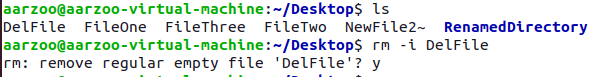
1. **rm Command**

It remove files or directories. Rm removes each specified file.By default, it does not remove directories.

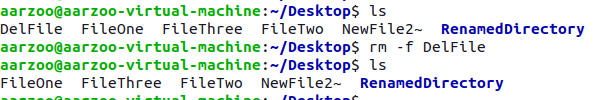


**COMMON OPTIONS IN rm COMMAND**

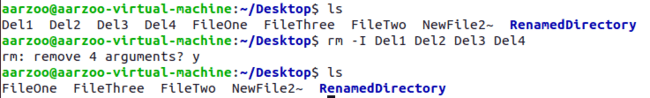
* **-i :** prompt before every removal



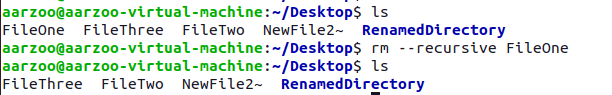
* **-f :** -f option overrides this minor protection and overwrite the destination file forcefully and delete the source file.



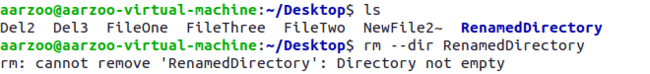
* **-I :** prompt once before removing more than three files, or when removing recursively; less intrusive than-i,while still giving protection against most mistakes



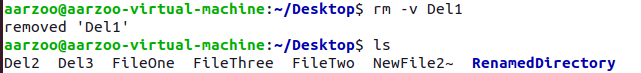
* **-r,-R,--recursive :** Remove directories and their contents recursively.



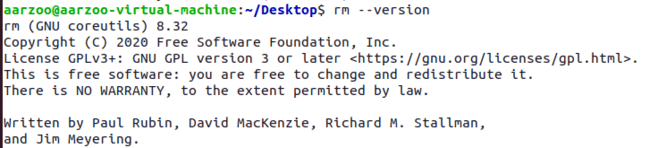
* **-d,--dir:** Remove empty directories.



* **-v,--verbose :** Explain what is being done.



* **--help :** Display help and exit.
* **--version :** Output version information and exit.

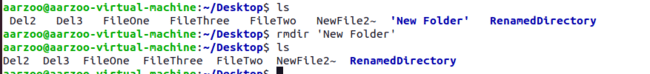


1. **rmdir Command**

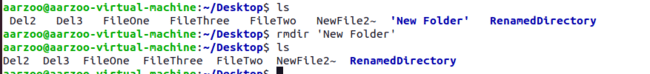
It removes an empty directory on various operating systems..

rmdir name\_of\_directory

where name\_of\_directory corresponds with the name of the directory one wishes to delete.



* **-p :** removes parent directories if they are also empty.



**PRACTICAL - 3**

**Commands**

* + - 1. **whatis Command**

This command is helpful to get brief information about Linux commands or functions. Whatis command displays man page single line description for command that matches string passed as a command line argument to whatis command. Whatis command searches for string in its index databases which is maintained by mandb program. Whatis command picks short description of NAME section of man page of command that matches to input given to the whatis command. One liner for manual pages.



**COMMON OPTIONS IN whatis COMMAND**

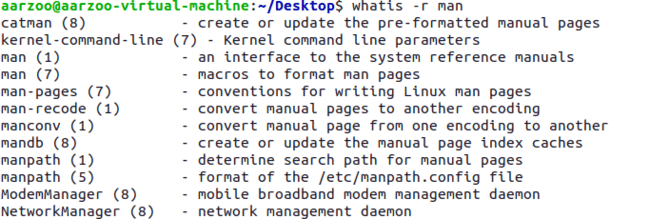
* **-s :** It is used to get Linux command information from specific section of man pages.



* **-w:** It is used to search Linux commands or functions information using wild card.



* **-r :** t is used to search Linux commands or functions information using regular expressions.



* **-l :** It trims long output of Linux commands or functions information to avoid “Not good” output display on terminal that is going beyond screen.



* + - 1. **whereis Command**

whereis command is used to find the location of souce/binary file of a command.

Find and whereis command are similar but whereis command produces the result more accurately by consuming less time comparatively.



**COMMON OPTIONS IN whereis COMMAND**

* -**b:** This option is used when we only want to search for binaries.



* -**m:** This option is used when we only want to search for manual sections.



* -**u:** This option search for unusual entries.



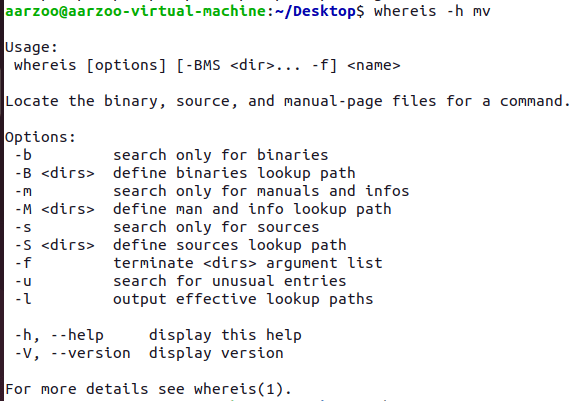
* -**V:** Displays the version information and exit.



* -**f:** This option simply terminate the last directory list and signals the start of file names.



* -**h:** Displays this help and exit.



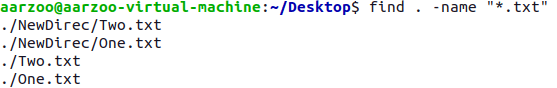
1. **find Command**

The find command in UNIX is a command line utility for walking a file hierarchy. It can be used to find files and directories and perform subsequent operations on them. It supports searching by file, folder, name, creation date, modification date,owner and permissions. By using the ‘-exec’ other UNIX commands can be executed on files or folders found.

**SYNOPSIS**

**find** [where to start searching from] [expression determines what to find] [-options] [what to find]

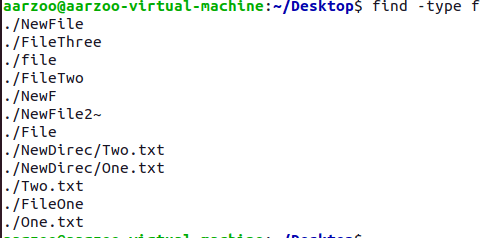
* Find files by name : Searches file with .txt configuration.



* Finding files by type

The '-type' parameter is used to specify the file type. Types are:

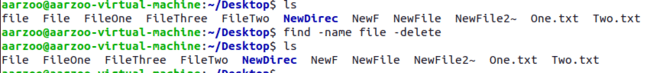
f: regular file



d: directory

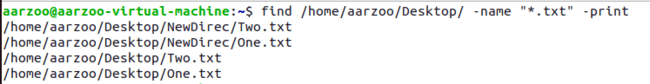


* Find and delete a file

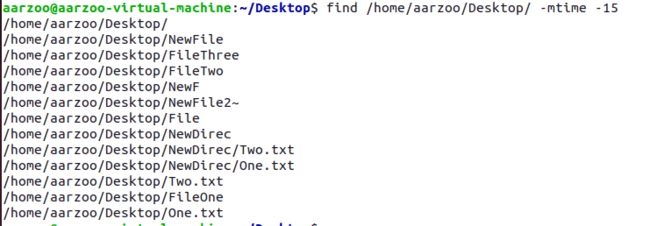
The '-delete' option is used to delete a specific file. We need to be very careful while using this command because there is no undo option if it is once executed 

**COMMON OPTIONS IN find COMMAND**

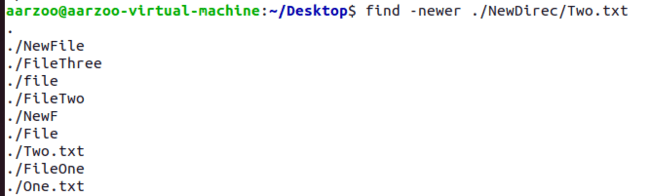
* **-print:** Display the path name of the files found by using the rest of the criteria.



* **-mtime, -atime, -ctime**: Searches for files based on modification, access, or status change times.

****

* **-newer file:** Search for files that were modified/created after ‘file’.



* **-user name:** Search for files owned by user name or ID ‘name’.

****

1. **type Command**

This command is used to describe how its argument would be translated if used as commands. It is also used to find out whether it is built-in or external binary file.

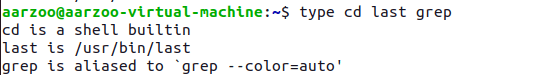
It shows a command’s type and location

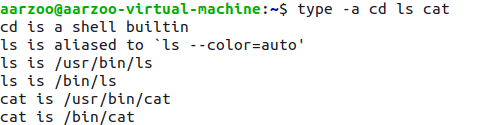
**type [OPTIONS] command names**



**COMMON OPTIONS IN type COMMAND**

* -**a:** This option is used to find out whether it is an alias, keyword or a function and it also displays the path of an executable, if available.



* -**t:** This option will display a single word as an output.
* alias – if command is a shell alias.
* keyword – if command is a shell reserved word.
* builtin – if command is a shell builtin.
* function – if command is a shell function
* file – if command is a disk file.
* **-p:** This option displays the name of the disk file which would be executed by the shell. It will return nothing if the command is not a disk file

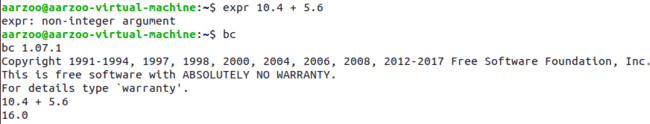
****

1. **bc(basic calculator) Command**

bc command is used for command line calculator. It is similar to basic calculator by using which we can do basic mathematical calculations.

echo and expr can also be used for calculations but don’t support floating point numbers.



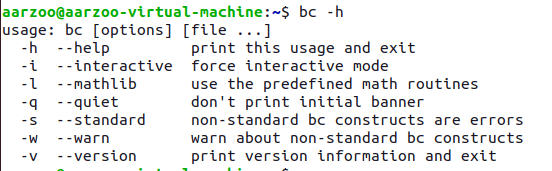


**SYNOPSIS**

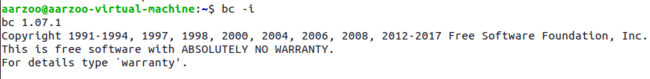
bc [ -hlwsqv ] [long-options] [ file ... ]

**COMMON OPTIONS IN bc COMMAND**

* **-h{--help}:** Print the usage and exit.



* **-i{--interactive}:** Force interactive mode.



* **-v{--version}** : Print the version number and copyright and quit.

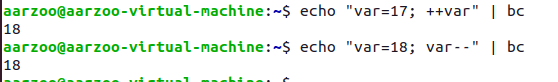


**The bc command supports the following features:**

* Arithmetic operators



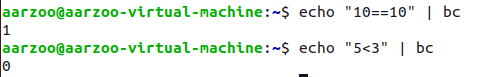
* Increment or Decrement operators



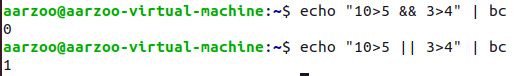
* Assignment operators



* Comparison or Relational operators



* Logical or Boolean operators



* Precision setting : setting the scale of division as per requirement.



1. **expr Command**

The **expr command** is used to evaluate a given expression.

**SYNOPSIS**

expr EXPRESSION

expr OPTION

**Basic arithmetic operations using expr command**

* **Addition**

****

* **Subtraction**

****

* **Multiplication**

****

* **Division**

****

* **Find the modulus value**

****

* **Find the length of a string**

****

* **Find the position of character in a string**

****

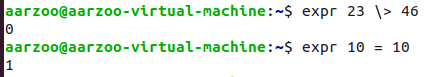
* **Find the substring from a string**

****

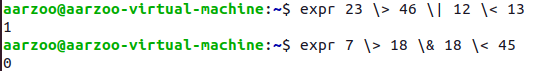
* **Matching the strings**

****

* **Comparing two expressions**

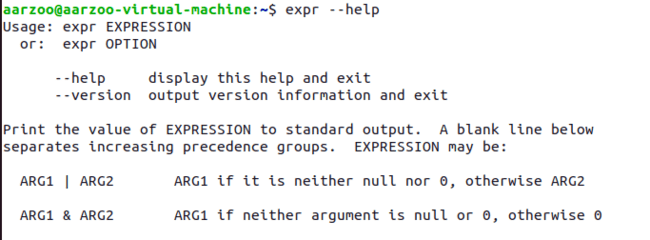
****

****

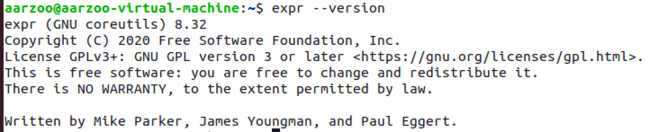
****

**COMMON OPTIONS IN expr COMMAND**

* **--help** : display help and exit.



* **--version** : output version info and exit.

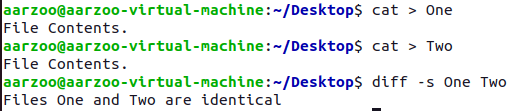


1. **diff Command**

diff stands for difference.

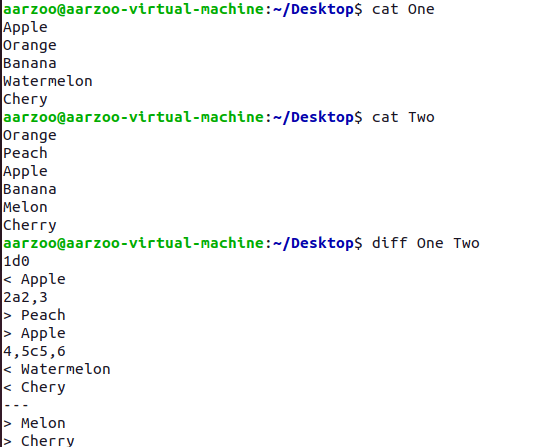
This command is used to display the differences in the files by comparing the files line by line. Unlike its fellow members, cmp and comm, it tells us which lines in one file have is to be changed to make the two files identical. It tells you the instructions on how to change the first file to make it match the second file.

We can use -s to get a prompt if the files are identical. Here first.txt and only.txt are identical.



Some special symbols are used by diff command.

These are:- a : add c : change d : delete

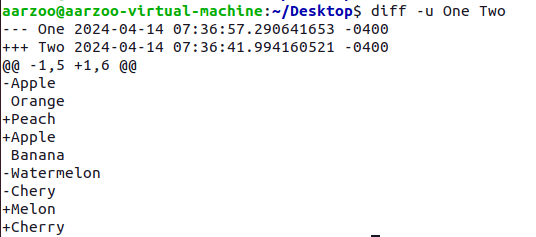


**COMMON OPTIONS IN diff COMMAND**

* **-s :** Used to get prompt if files are identical otherwise there comes no prompt.

****

* **-u :** Displays the changes need to be done in contextual format.

****

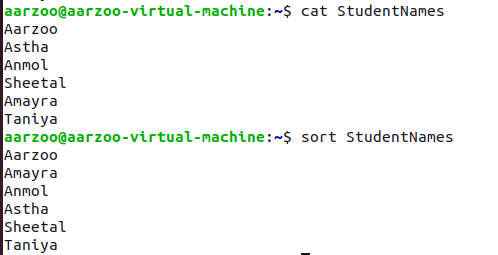
* **-q :** Tells if the files differ without any details

****

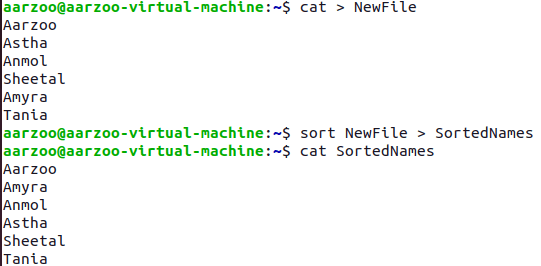
1. **sort Command**

The command sorts the lines of text files. It writes sorted concatenation of all FILE(s) to standard output.

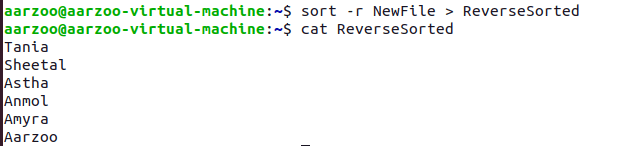
* The 'sort' command sorts the file content in an alphabetical order.



* Saving sorted Content to a new file

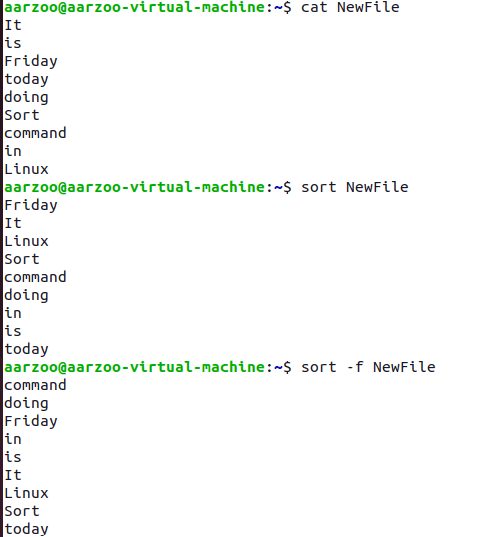


* Reverse sort

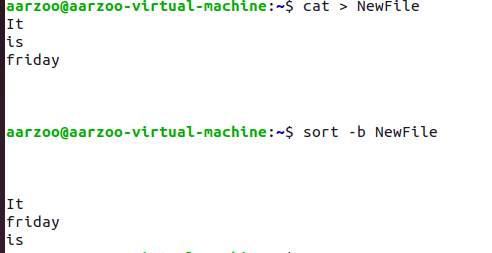


**COMMON OPTIONS IN sort COMMAND**

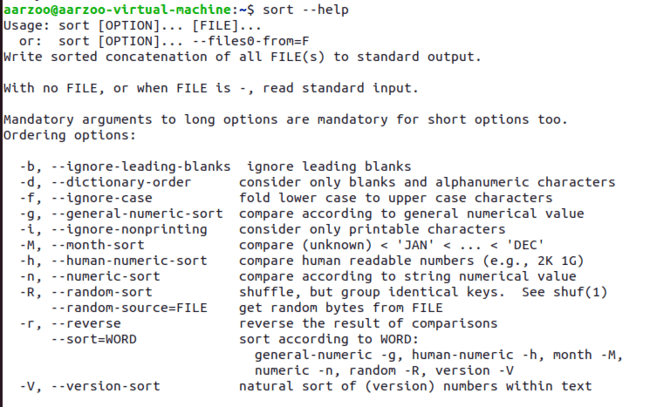
* **-f :** ignore-case



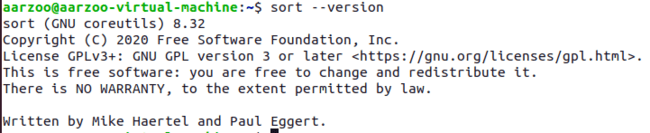
* **-b, --ignore-leading-blanks:** Ignore leading blanks.



* **--help :** Display help and exit.

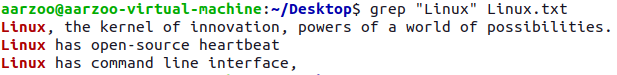


* **--version** : Output version info and exit.



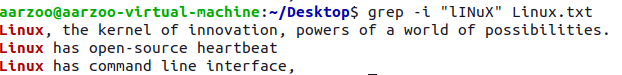
1. **grep Command**

Grep, short for “global regular expression print”, is a command used for searching and matching text patterns in files contained in the regular expressions.

****

**COMMON OPTIONS IN grep COMMAND**

* **-i :** The -i option enables to search for a string case insensitively in the given file. It matches the words like “lINuX”, “Linux”.



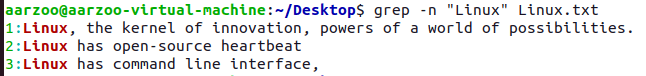
* **-c :** We can find the number of lines that matches the given string/pattern



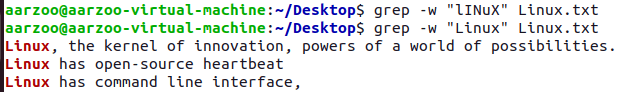
* **-l :** We can just display the files that contains the given string/pattern.



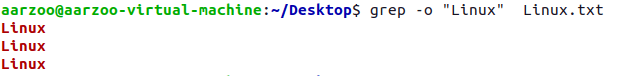
* **-n :** To number the lines where the string pattern is matched.



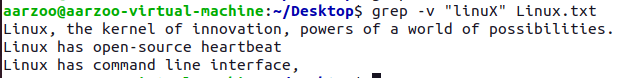
* **-w :** The -w option to grep makes it match only the whole words. Case sensitive matches are only made. For example : “lINux” is not found in the file.



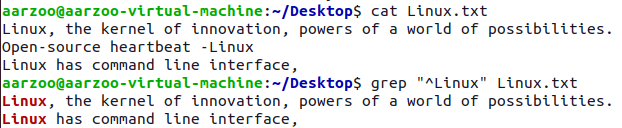
* **-o :** Displays only the matched string by using the -o option**.**



* **-v :** Display the lines that are not matched with the specified search string pattern using the -v option.



* **“^StartWord” :** The ^ regular expression pattern specifies the start of a line. This can be used in grep to match the lines which start with the given string or pattern.



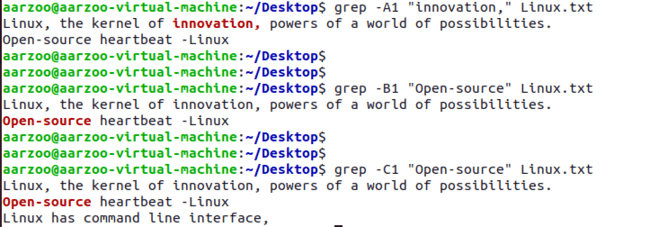
* **“EndWord$” :** The $ regular expression pattern specifies the end of a line. This can be used in grep to match the lines which end with the given string or pattern.



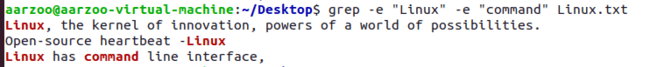
* **-R :** -R prints the searched pattern in the given directory recursively in all the files.



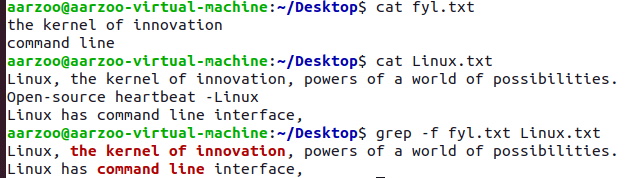
* **-A,-B,-C :** -A prints the searched line and n lines after the result, -B prints the searched line and n lines before the result, and -C prints the searched line and n lines after and before the result.



* **-e :** Can use multiple times to search for different word patterns.



* **-f :** File option Takes patterns from file, one per line.



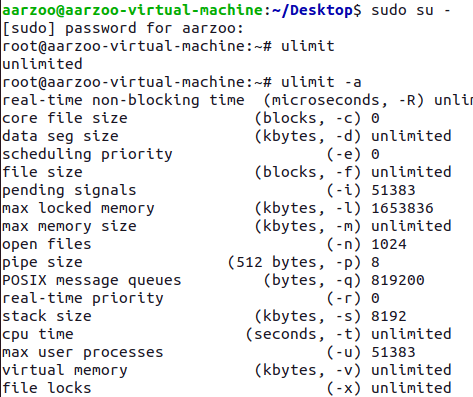
* **Searching a pattern using grep:-** Match a given pattern from the file for example:- Search a pattern from file that starts either with small case or large case alphabet contains $ in-between the file and ends with small case character.

**[{a-z},{A-Z}] \* [$] \* [a-z]**

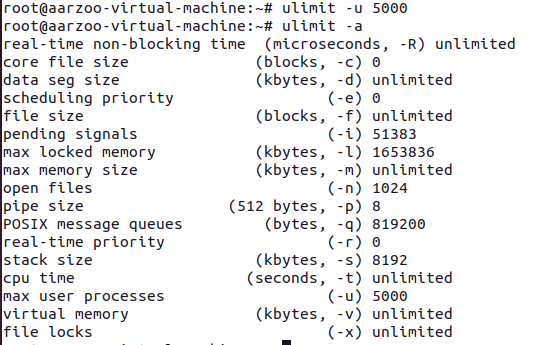
****

1. **ulimit Command**

get and set user limits



* We can change for example max user processes for a user at an instant, Here max user processes is changed from 51383 to 5000.



1. **du Command**

Estimate the file space usage.

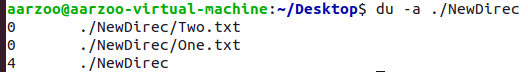
**SYNOPSIS**

du [OPTION]... [FILE]...



**COMMON OPTIONS IN du COMMAND**

* **-a :** write counts for all files, not just directories.



1. **free Command**

Display amount of free and used memory in the system

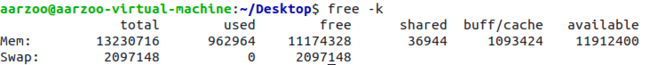


**COMMON OPTIONS IN free COMMAND**

* **-b :** Display the amount of memory in bytes.



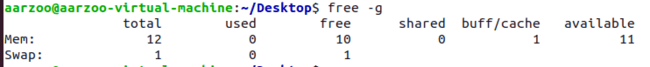
* **-k :** Display the amount of memory in kibibytes. This is the default



* **-m :** Display the amount of memory in mebibytes.



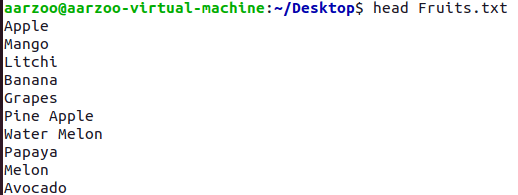
* **-g :** Display the amount of memory in gibibytes.

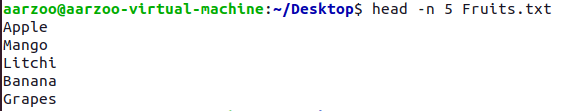


1. **head Command**

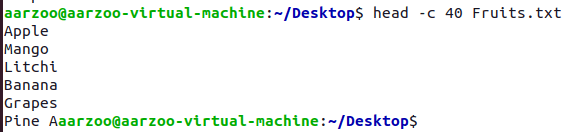
* output the first part of files
* Without any option, it displays only the first 10 lines of the file specified

**COMMON OPTIONS IN head COMMAND**

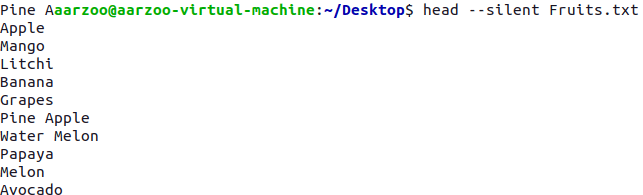
* **-n :** print the first NUM lines instead of the first 10; with the leading '-', print all but the last NUM lines of each file



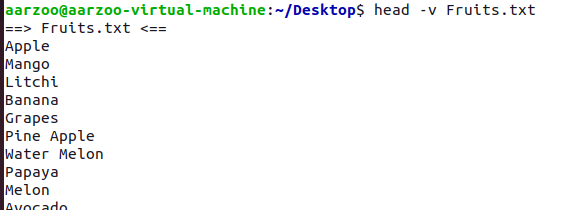
* **-c :** print the first NUM bytes of each file; with the leading '-', print all but the last NUM bytes of each file



* **-q, --quiet, --silent:** never print headers giving file names



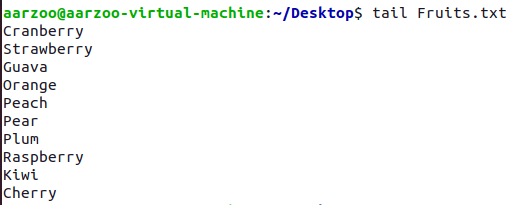
* **-v :** always print headers giving file names



1. **tail Command**

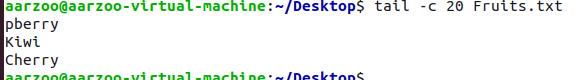
output the last part of files

Without any option, it displays only the last 10 lines of the file specified

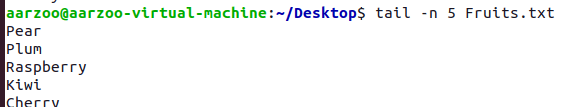


**COMMON OPTIONS IN tail COMMAND**

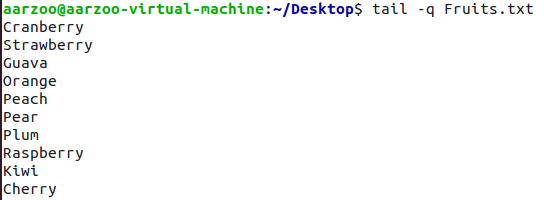
* **-c :** output the last NUM bytes; or use -c +NUM to output starting with byte NUM of each file



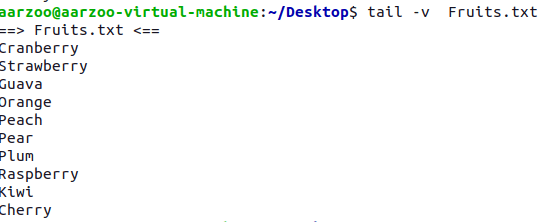
* **-n :** output the last NUM lines, instead of the last 10; or use -n +NUM to output starting with line NUM



* **-q :** never output headers giving file names



* **-v :** always output headers giving file names

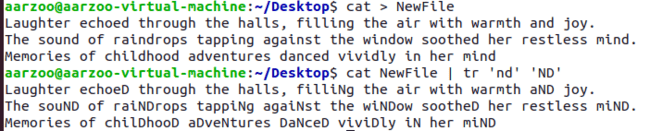


1. **tr Command**

translate or delete characters

tr [OPTION]... SET1 [SET2]

Change Case

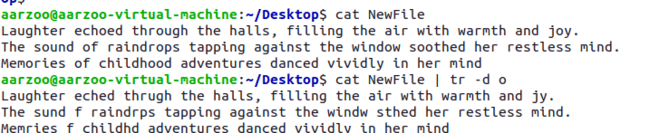


**COMMON OPTIONS IN tr COMMAND**

* **-c :** use the complement of SET1 The -c option complements the specified character set. Here, it replaces all non-vowel characters with an asterisk.



* **-d :** delete characters in SET1, do not translate



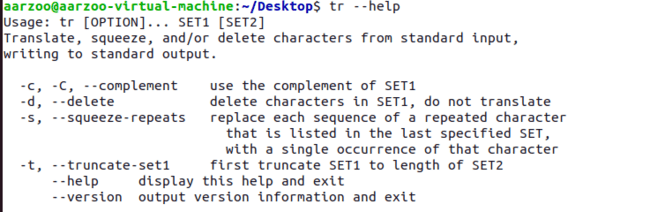
* **-s :** replace each sequence of a repeated character that is listed in the last specified SET, with a single occurrence of that character



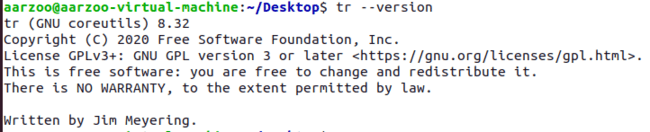
* **-t :** first truncate SET1 to length of SET2



* **--help :** display this help and exit



* **--version :** display this help and exit



1. **cut Command**

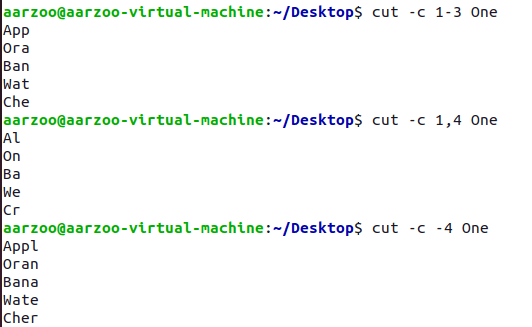
remove sections from each line of files

cut OPTION... [FILE]...

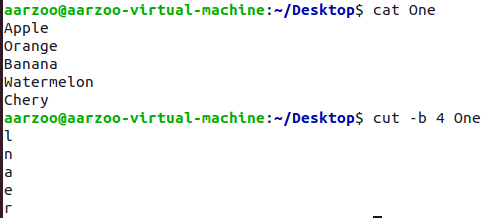
**COMMON OPTIONS IN cut COMMAND**

* **-c :** The '-c' option is used to cut a specific section by character. However, these character arguments can be a number or a range of numbers, a list of comma-separated numbers, or any other character.

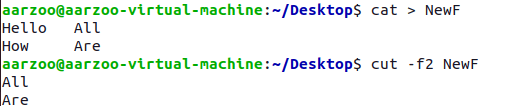
-4 means starting from the first character till the fourth one.



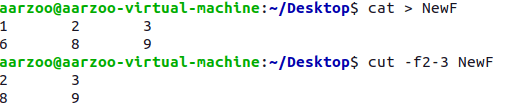
* **-b :** select only these bytes. 4th byte in this case



* **-f :** Cuts a field line (column).

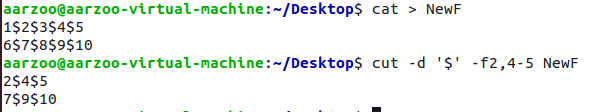




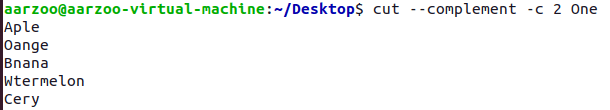


* **-d :** use DELIM instead of TAB for field delimiter

cut -d ' ' -f(columnNumber) <fileName>



* **--complement :** complement the set of selected bytes, characters or fields

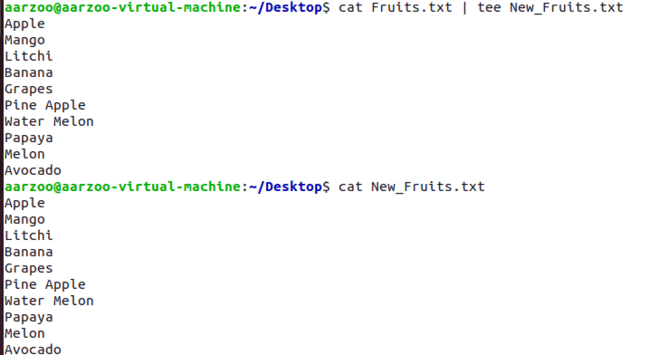


1. **tee Command**

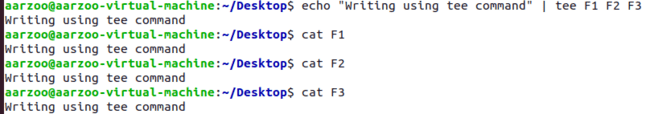
This command reads the standard input and writes it to both the standard output and one or more files. It basically breaks the output of a program so that it can be both displayed and saved in a file. It does both the tasks simultaneously, copies the result into the specified files or variables and also display the result.

tee [OPTION]... [FILE]...

The tee command is used to write a standard input to standard output and a file. It is used after a pipe. To write to standard output and a file, specify the tee command after a pipe and provide the file(s) name.

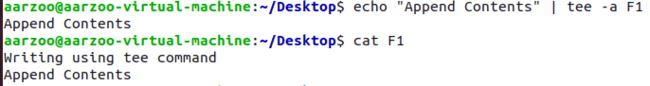


Writing to multiple files

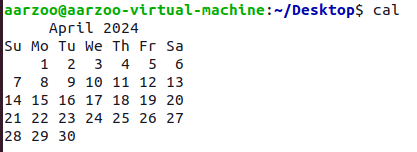


**COMMON OPTIONS IN tee COMMAND**

* **-a :** The '-a' option is used with the tee command to append the output and write it to a file



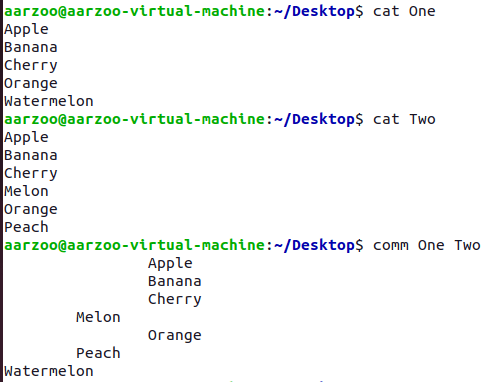
1. **cal Command**



1. **comm Command**

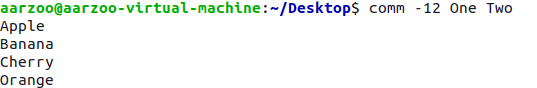
The 'comm' command compares two files or streams. By default, 'comm' will always display **three columns**.

First column indicates non-matching items of first file, second column indicates non-matching items of second file, and third column indicates matching items of both the files. Both the files has to be in sorted order for 'comm' command to be executed.

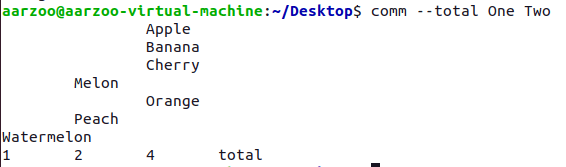


**COMMON OPTIONS IN comm COMMAND**

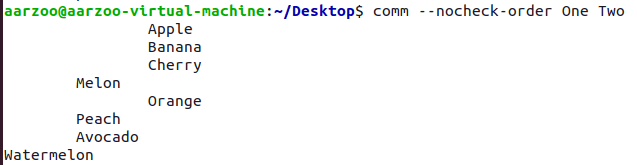
* -**1,-2,-3:** To hide the respective columns, -12 hides 1,2 column and displays only the common lines to the file.



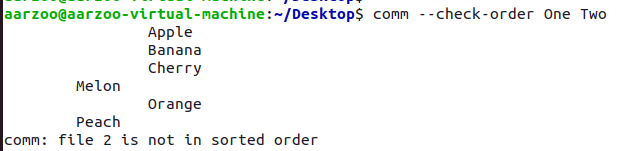
* --**total :** output a summary.



* **--nocheck-order :** do not check that the input is correctly sorted.



* **--check-order :** check that the input is correctly sorted, even if all input lines are pairable



**PRACTICAL:- 4**

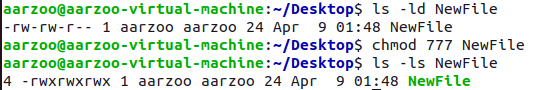
**Commands**

1. **Chmod command**

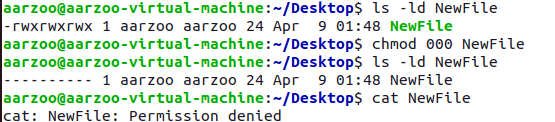
* You can see the permission for any file or directory by typing the ls -ld command.

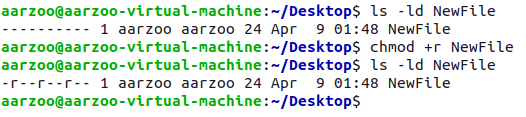


* Changing permissions with chmod (numbers)
* Each permission (read, write, and execute) is assigned a number—r=4, w=2, and x=1—and you use each set’s total number to establish the permission.

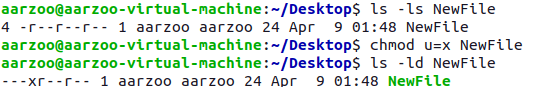


* Permission denied if we tried to access today.txt



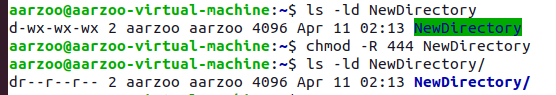
* Changing permissions with chmod (letters)
* You can also turn file permissions on and off using plus (+) and minus (–) signs, respectively, along with letters to indicate what changes and for whom. Using letters, for each fi le you can change permission for the user (u), group (g), other (o), and all users (a). What you would change includes the read (r), write (w), and execute (x) bits. 
* Giving write permission only to user

Here “=” deletes all the previous permissions and assign only the execute permission to user.

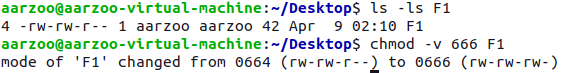


**COMMON OPTIONS IN chmod COMMAND**

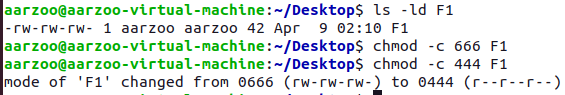
* **-R :** change files and directories recursively



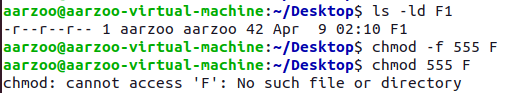
* **-v :** output a diagnostic for every file processed



* **-c :** like verbose but report only when a change is made



* **-f :** suppress most error messages : Error message of file not existing supressed.



1. **umask command**

The umask command is used to set this mask, or to show you its current value. The default creation permissions for files are 666 and for directories 777.

For example, to calculate how uname 022 will affect newly created files and directories, use:

Files: 666 - 022 = 644. The owner can read and modify the files. Group and others can only read the files.

Directories: 777 - 022 = 755.The owner can cd into the directory and list read, modify, create or delete the files in the directory. Group and others can cd into the directory and list and read the files. 

* To set a specific umask



**COMMON OPTIONS IN umask COMMAND**

* **-p :** if MODE is omitted, output in a form that may be reused as input



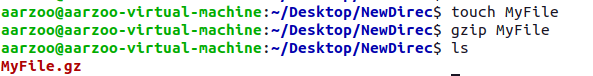
* **-S :** makes the output symbolic; otherwise an octal number is output



1. **gzip command**

The gzip program is used to compress one or more files. When executed, it replaces the original file with a compressed version of the original. The corresponding gunzip pro gram is used to restore compressed files to their original, uncompressed form.

The file has been zipped with .gz extension.



**COMMON OPTIONS IN gzip COMMAND**

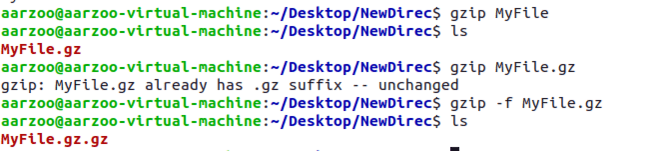
* **-c :** Write output on standard output; keep original files unchanged.



* **-d :** Decompress. This causes gzip to act like gunzip. May also be specified with --decompress or --uncompress.



* **-f :** Force compression even if a compressed version of the original file already exists. May also be specified with --force.



* **-l :** For each compressed file, list the following fields: compressed size: size of the compressed file uncompressed size: size of the uncompressed file ratio: compression ratio (0.0% if unknown) uncompressed\_name: name of the uncompressed file



* **-v :** Display the name and percentage reduction for each file compressed or decompressed.



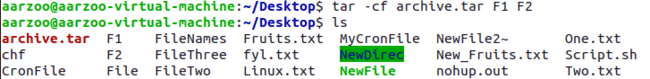
1. **tar command**

The tar  command is used to create an archive, grouping multiple files in a single file.

Its name comes from the past and means tape archive. Back when archives were stored on tapes.

This command creates an archive named archive.tar  with the content of file1  and file2 :

* The c  option stands for create. The f  option is used to write to file the archive.

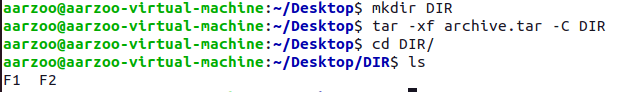




* List the files contained in an archive

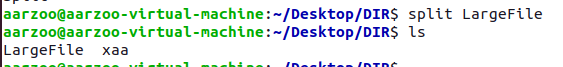


* To extract them to a specific directory, use:



1. **split command**

Split command in Linux is used to split large files into smaller files



**COMMON OPTIONS IN split COMMAND**

* **-a:** generate suffixes of length N (default 2)

****

* **-n :** Generates a specific number of output files.

****

* **-additional-suffix=SUFFIX:** append an additional SUFFIX to file names

****

* **-b:** put SIZE bytes per output file

This will split the file into parts, each containing 15 bytes.

****

* **-C:** put at most SIZE bytes of records per output file

This will split the file into parts, each containing up to 15 bytes of records.

****

* **-d:** use numeric suffixes starting at 0, not alphabetic

****

* **--numeric-suffixes[=FROM]:** same as -d, but allow setting the start value

****

* **--verbose:** print a diagnostic just before each output file is opened

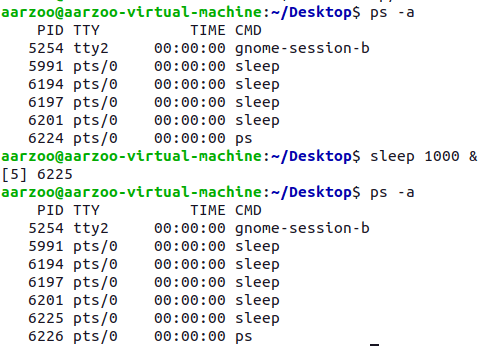
****

1. **sleep command**

delay for a specified amount of time



Sleep as a background process



1. **shutdown command**

Halt, power-off or reboot the machine



**COMMON OPTIONS IN shutdown COMMAND**

* + **-r:** Reboot the machine.

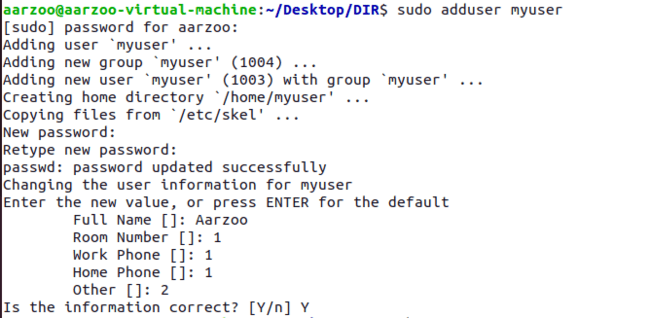


* + **-P:** Output version info and exit

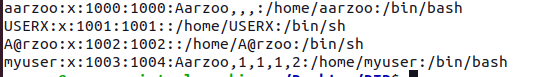


1. **adduser command(\*)**

* add a user to the system

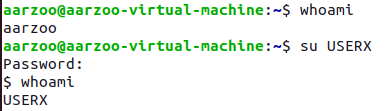
****

* Displaying user using cat /etc/passwd

****

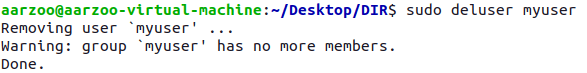
1. **su command**

change user ID or become superuser

****

1. **deluser**

remove a user from the system

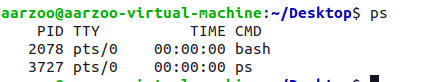
****

**PRACTICAL:- 5**

**Commands**

1. **ps command**

* The ps command is used for checking running process.

****

PID - Process ID gives the unique identification number of the process

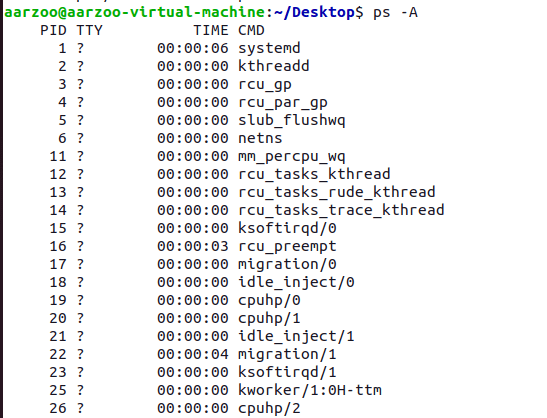
TTY – terminal type

Time – Cpu utilization time by the process

CMD – command used by the process

**COMMON OPTIONS IN ps COMMAND**

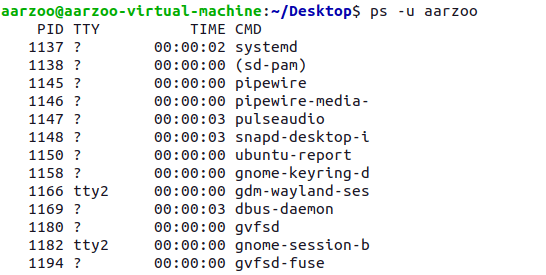
* + **-A:** to view all the running process

****

* **-C:** gives information about a particular process

****

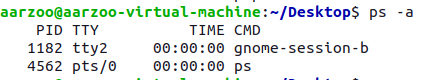
* **-u:** if we want to check running process for particular user

****

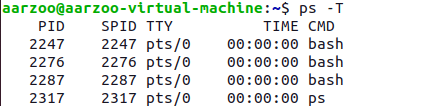
* **-C:** Search Process PID

****

* **-a:** View processes not associated with terminal

****

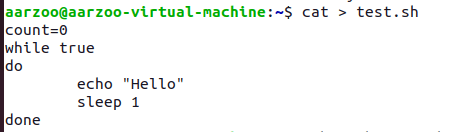
* **-T:** View processes associated with the terminal

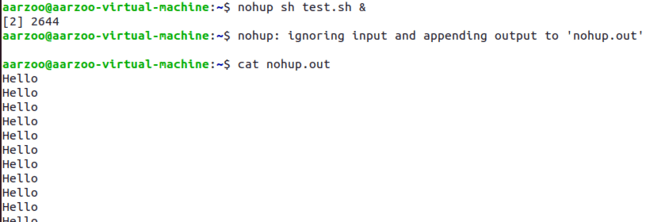
****

1. **nohup (no hangup)**

If you want your process keep running even after closing your terminal, you can use nohup.

Nohup, short for nohangup is a command in Linux systems that keep processes running even after exiting the shell or terminal.Nohup prevents the processesor jobs from receiving the SIGHUP(SignalHangUP) signal.This is a signal that is sent to a process upon closing or exiting the terminal.



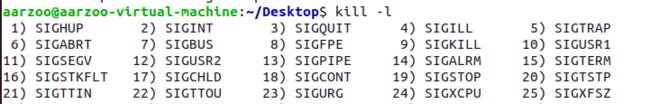


Closed the terminal – Process still running

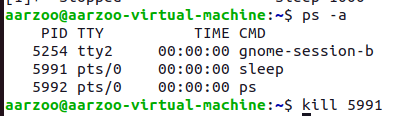
1. **Kill**

It send a signal to a process.

* + **-l :** List signal names. This option has optional argument, which will convert signal number to signal name, or other way round.

****

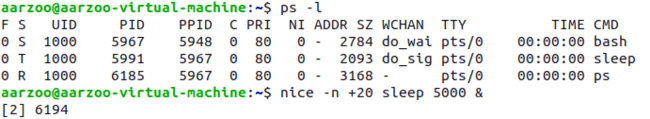
* + Killing by mentioning the process ID.

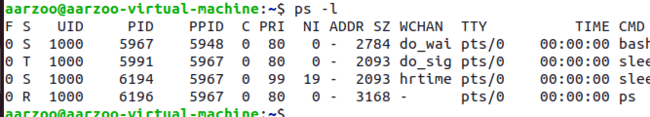


1. **Nice**

run a program with modified scheduling priority

Here scheduling priority of pid 5991 is being changed using nice command

****

****

1. **At**

To Schedule a task using the “at” command in Linux, you simply specify the time for the task followed by the task itself.

The “at” command is a powerful tool for scheduling tasks to run at a specific time in the Linux environment.

One time execution

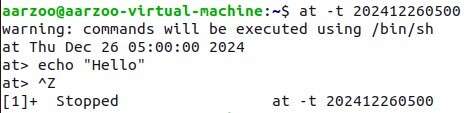
C:\Users\aarzo\AppData\Local\Packages\Microsoft.Windows.Photos_8wekyb3d8bbwe\TempState\ShareServiceTempFolder\Screenshot 2024-04-11 174044.jpeg

**COMMON OPTIONS IN AT COMMAND**

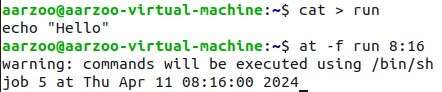
* **-l:** Lists the user pending tasks

C:\Users\aarzo\AppData\Local\Packages\Microsoft.Windows.Photos_8wekyb3d8bbwe\TempState\ShareServiceTempFolder\Screenshot 2024-04-11 174216.jpeg

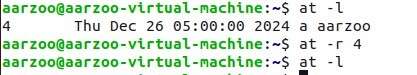
* **-t:** Run the job at time​, given in the format [[CC​]YY​]MMDDhhmm​[.ss​].



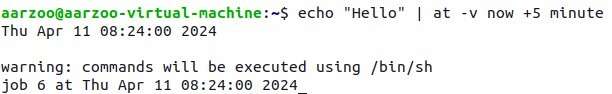
* **-f:** Reads the job from file​ rather than standard input



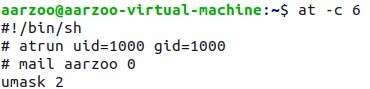
* **-r:** It removes a job from the at queue.



* **-v:** Shows thet timet he job will be executed before reading the job.Times displayed will be in the format "Thu Feb 20 14:50:00 1997".



* **-c:** cats the jobs listed on the command line to standard output.

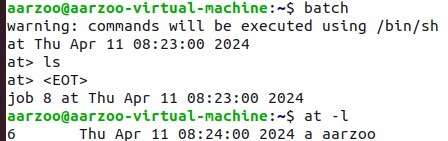


1. **batch**

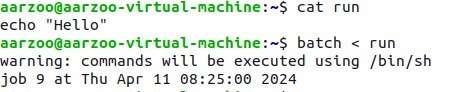
* sed to read commands from standard input or a specified file and execute them when system load levels permit i.e. when the load average drops below 1.5
* It is important to note that batch does not accepts any parameters.
* One time execution

Working with batch command

* + Type batch and press enter you will see a prompt where you can enter commands and end it with <ctrl-d>



* + Providing file containing commands to batch command for execution



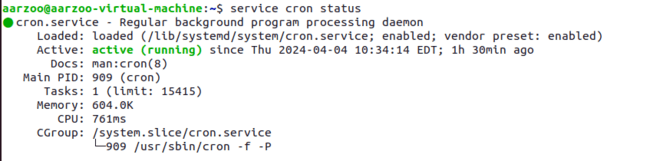
1. **crontab**

The crontab is a list of commands that we want to run on a regular schedule, and also the name of the command used to manage that list. Crontab stands for “cron table, ” because it uses the job scheduler cron to execute tasks.

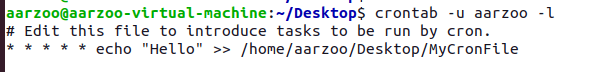
**SYNOPSIS**

crontab [ -u user ] file

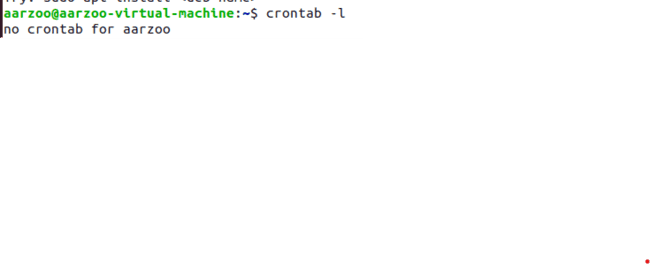
crontab [ -u user ] [ -i ] { -e | -l | -r }



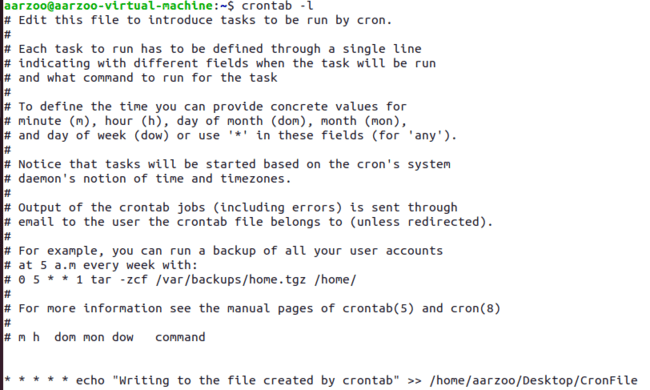




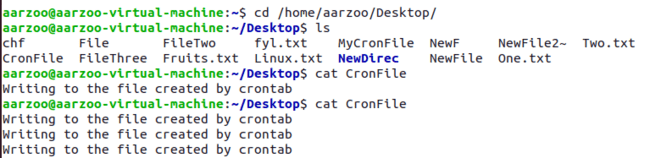
* To view the list of existing cron jobs in your crontab, use the command: crontab -l



* To add cronjob we can use crontab -e command



* Working of crontab



* + Removing the cronjob using -r .



* Generating prompt before the removal of cronjob.



1. **wall**

write a message to all users

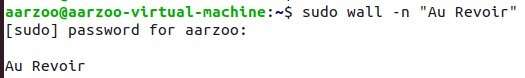
**SYNOPSIS**

wall [-n] [-t timeout] [-g group] [message | file]

C:\Users\aarzo\AppData\Local\Packages\Microsoft.Windows.Photos_8wekyb3d8bbwe\TempState\ShareServiceTempFolder\Screenshot 2024-04-11 180133.jpeg

**COMMON OPTIONS IN wall COMMAND**

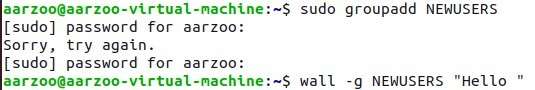
* + -**n --nobanner:** Suppress the banner.



* + **-t, --timeout:** Abandon the write attempt to the terminals after timeout seconds. This timeout must be a positive integer. The default value is 300 seconds,

C:\Users\aarzo\AppData\Local\Packages\Microsoft.Windows.Photos_8wekyb3d8bbwe\TempState\ShareServiceTempFolder\Screenshot 2024-04-11 212453.jpeg

* + **-g, --group**: Limit printing message to members of group defined as a group argument.



1. **Write**

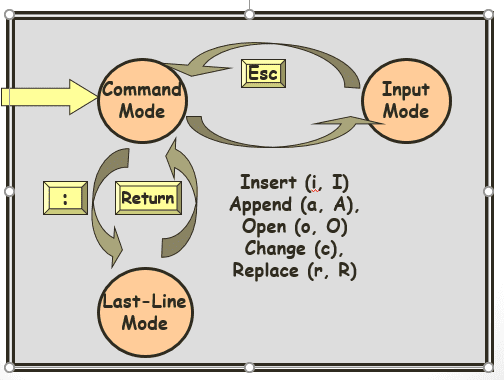
The write command sends a message to another user.The write utility allows you to communicate with other users by copying lines from your terminal to theirs.When you run the write command, the user you are writing to gets a message of the format:

C:\Users\aarzo\AppData\Local\Packages\Microsoft.Windows.Photos_8wekyb3d8bbwe\TempState\ShareServiceTempFolder\Screenshot 2024-04-11 212035.jpeg

**PRACTICAL:-6**

## What is VI Editor:

* The VI editor is the most popular and classic text editor in the Linux family.
* It is available in almost all Linux Distributions
* It works the same across different platforms and Distributions
* vi has multiple modes of operation: **input mode**, **command mode**, **last-line mode**
* Command mode takes the user commands, and the Insert mode is for editing text.



**Modes:**

* **Command mode:** by default, the vi editor opens in command mode. You can use commands to move cursor, delete. cut, copy, paste and save changes. Commands are case sensitive.
* **Insert mode:** Insert mode is what we work in most of the time and text is inserted into the file in this mode only. You can enter the insert mode by pressing “i” key.
* **Command line mode:** You can enter this mode by typing ":" which displays the command line entry at the bottom of the screen.

## Starting vi

* To start Vi, open a terminal or console and simply type vi filename (for existing file) or vi newfile (newfile will be name of your new file)
* The file will open in command mode.
* Type “i” to go to Insert mode and add your text in the file.
* Once you are done editing the file, Press “ESC” to go to command mode and save your changes to the file. Press “Shift+ZZ” to save the file.
* Sh Filename to execute the file.

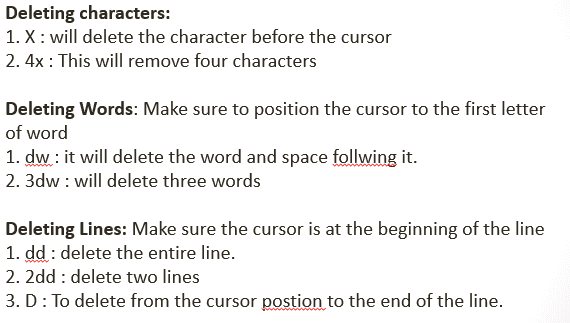
**Creating File:-** We can create a file using vi command and then writing the name of the file.It will take us to the Vi Editor where we can change the mode to insert mode by typing i, then we can write the contents to that file and shift back to command mode by pressing Esc. and then press “Shift + ZZ” to save file and go back to terminal.

‘sh’ Filename then executes this file.

****

****

## Deleting characters, words, lines:



Character deletion : x – deletes character before the cursor.





nx – deletes n characters.

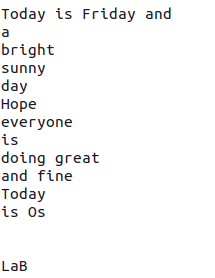




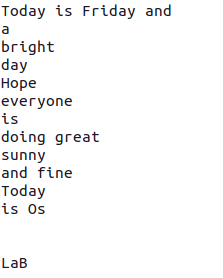
**Copy and replace commands:-**

* 5 mo 10 :- Moves content of line 5 to line 10.
* 5,10 mo 20 : Moves content of line to 10 to the line 20.
* 1,$ mo 20 : Moves content of line 1 to where the cursor is placed right now to line 20.
* .,$ mo 15 : Moves the content from start till where the cursor is to line 15.
* 1, 10 mo $ : Moves the content of line 1 to 10 to where the cursor points.
* 5 co 10 : Copies the content of line 5 to line 10 .
* 5,10 co 20 : Copies the content of line 5 to 10 to line 20.
* .,$ co 15 : Copies the content from start till where the cursor is to line 15 .
* 1, 10 co $ : Copies the content from line 1 to 10 t0 where the cursor points.
* 1, 10 s/abc/xyz : Replaces the pattern abc to xyz in the first occurrence of the pattern.
* 1, 10 s/abc/xyz/g : Replaces all the ‘abc’ to ‘xyz’ from line 1 to line 10.

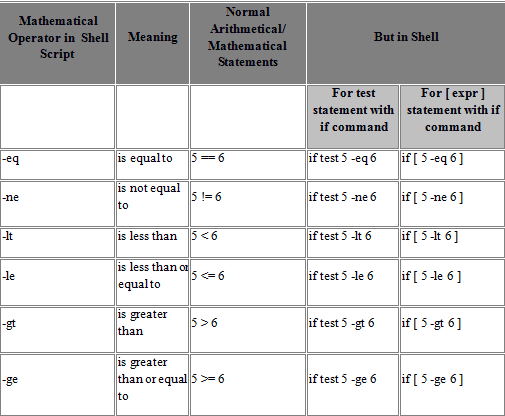
Original Contents of the file

****

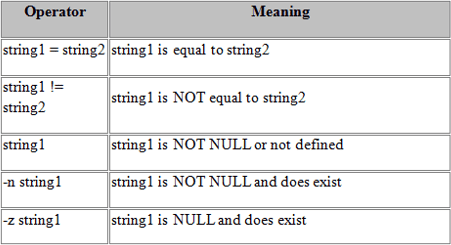
5 mo 10 :- Moves content of line 5 to line 10.

****

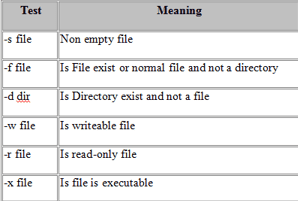
## Mathematical operations :

****

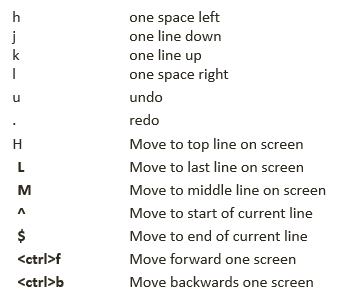
**String operations:**

****

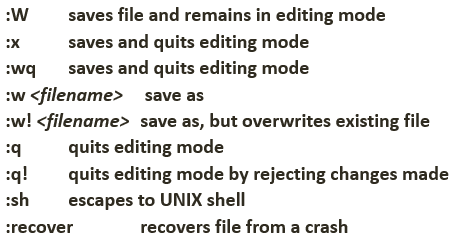
## Files and directories:



**Moving the cursor:**



**Execute mode:**

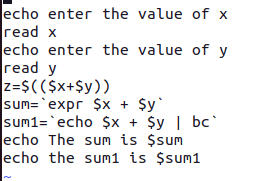


# **PRACTICAL - 7**

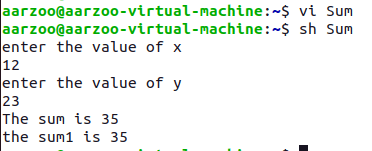
## Aim-: To execute bash scripts for the given progams.

* **Addition of two numbers**

**IMPLEMENTATION –**

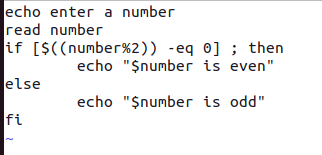
****

**OUTPUT –**

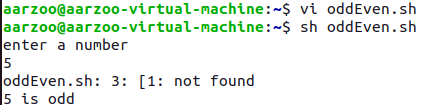
****

* **Check whether the number entered is odd/even**

**IMPLEMENTATION –**

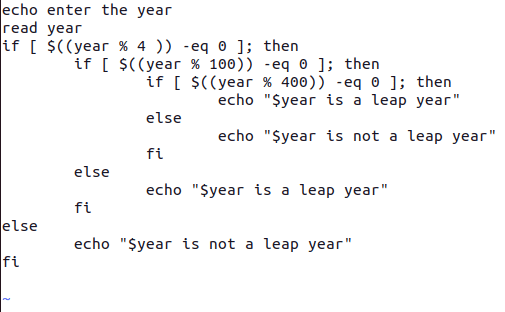
****

**OUTPUT –**

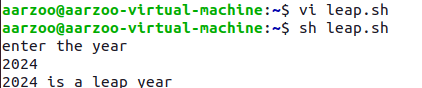
****

* **Check whether the year is leap or not**

**IMPLEMENTATION –**

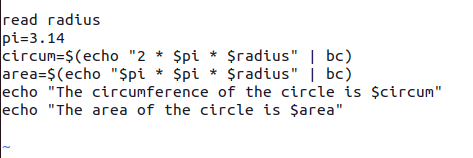
****

**OUTPUT –**

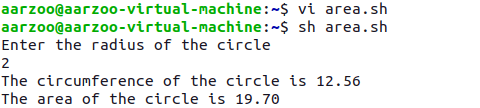
****

* **Calculate the average salaries of employees**

**IMPLEMENTATION –**

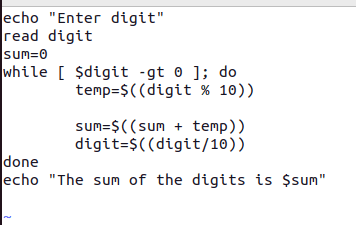
****

**OUTPUT –**

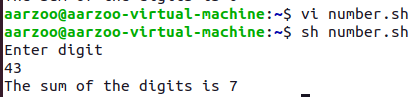
****

* **To find the sum of digits of a number**

**IMPLEMENTATION –**

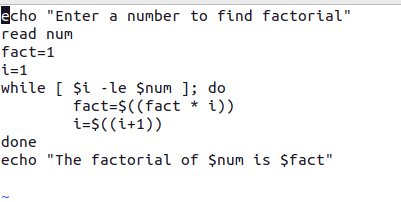
****

**OUTPUT –**

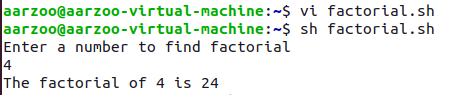
****

* **To find factorial of a number**

**IMPLEMENTATION –**

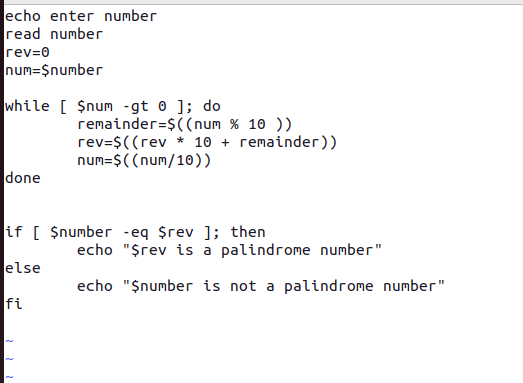
****

**OUTPUT –**

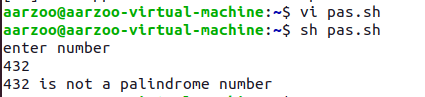
****

* **Check whether the number is palindrome or not**

**IMPLEMENTATION –**

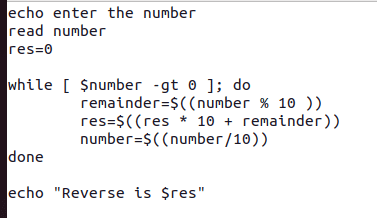
****

**OUTPUT –**

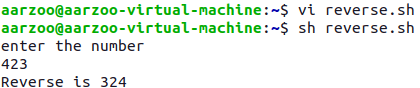
****

* **Reverse of a given number**

**IMPLEMENTATION –**

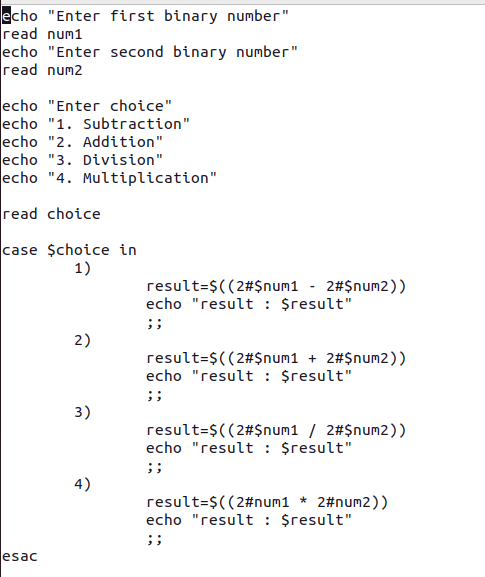
****

**OUTPUT –**

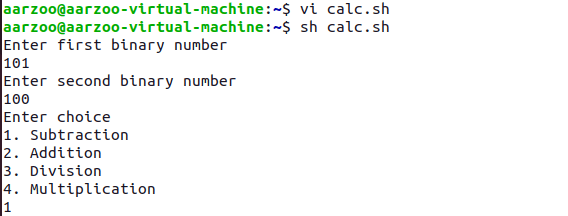
****

* **To create a binary calculator using switch case**

**IMPLEMENTATION –**

****

**OUTPUT –**

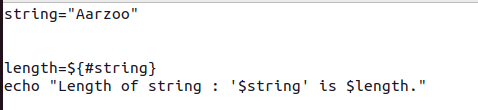
****

# **PRACTICAL - 8**

## Aim-:To implement following string operations in vim editor.

* Find the length of string

**IMPLEMENTATION –**

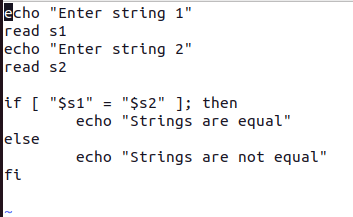
****

**OUTPUT-**

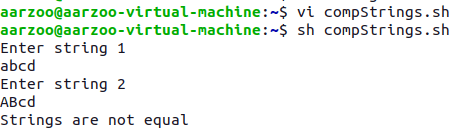
****

* Compare two strings

**IMPLEMENTATION –**

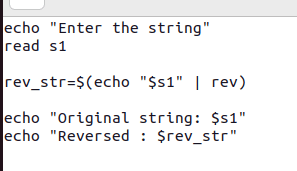
****

**OUTPUT-**

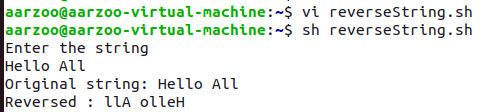


* Reverse of a string

**IMPLEMENTATION –**

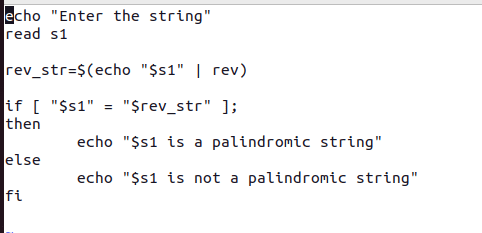
****

**OUTPUT-**

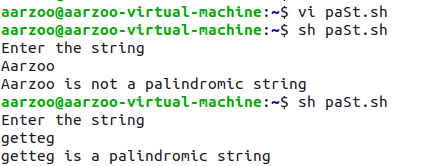
****

* String is palindrome or not

**IMPLEMENTATION –**

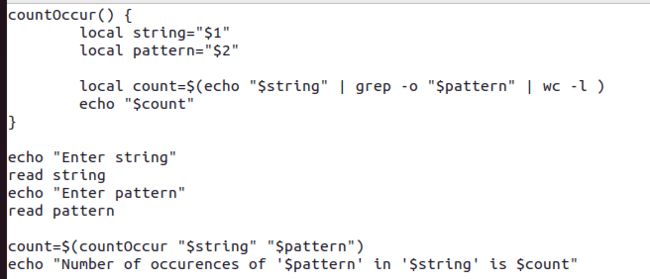
****

**OUTPUT-**

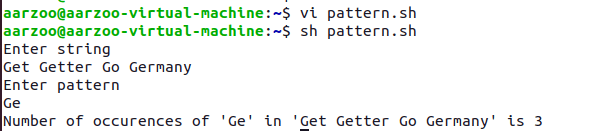
****

* Occurrence of a pattern

**IMPLEMENTATION –**

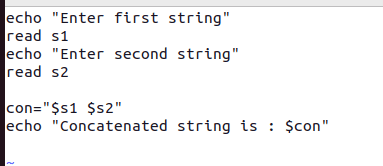
****

**OUTPUT-**

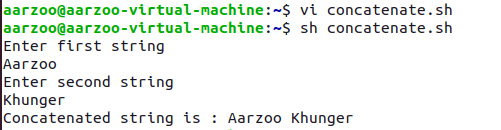


* Concatenate strings

**IMPLEMENTATION –**

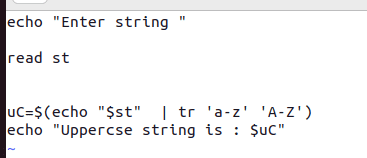
****

**OUTPUT-**

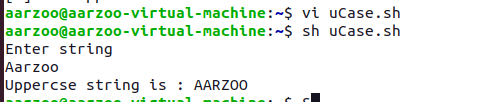
****

* Convert lowercase letters into uppercase.

**IMPLEMENTATION –**

****

**OUTPUT-**

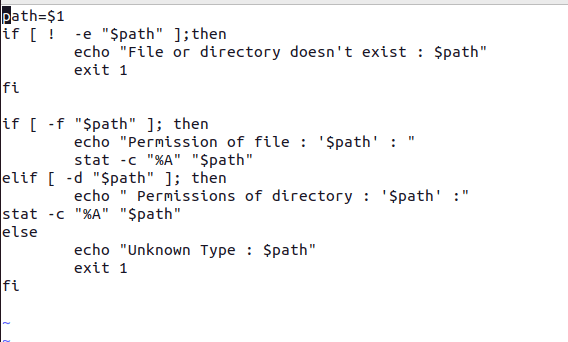


# **PRACTICAL-9**

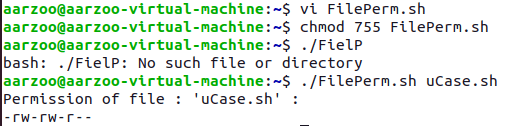
## To perform File Operations

### To see permissions of a file after checking whether it is file or directory

**IMPLEMENTATION –**

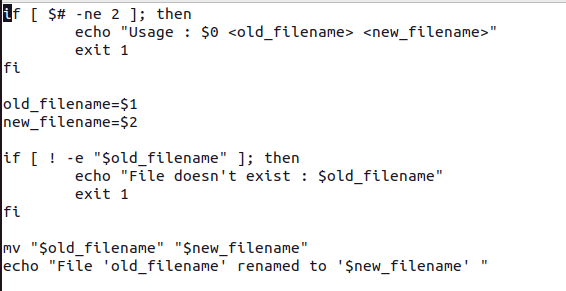
****

**OUTPUT –**

****

### Rename file

**IMPLEMENTATION –**

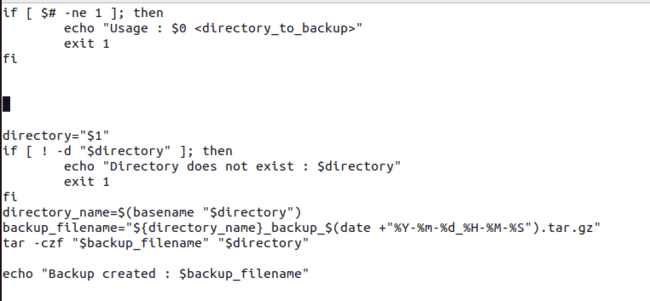
****

**OUTPUT –**

****

### Take backup of a directory

**IMPLEMENTATION –**

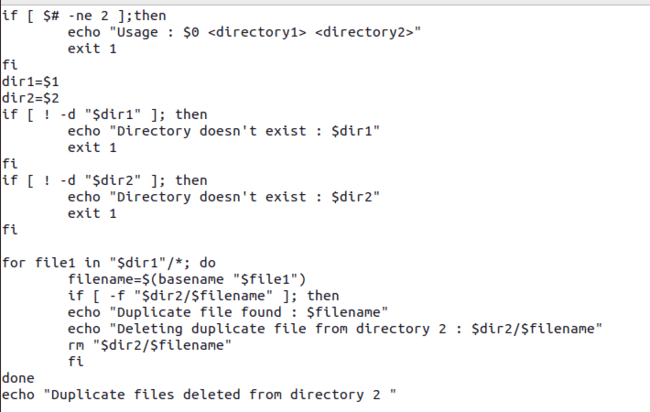
****

**OUTPUT –**



### Compare directories and delete duplicate files

**IMPLEMENTATION –**

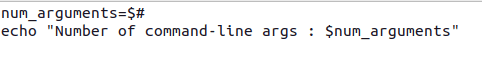
**s**

**OUTPUT –**



### Number of arguments.

**IMPLEMENTATION –**

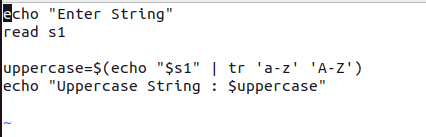
****

**OUTPUT-**

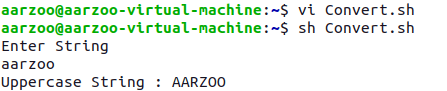


### Convert lowercase letters into uppercase.

**IMPLEMENTATION –**

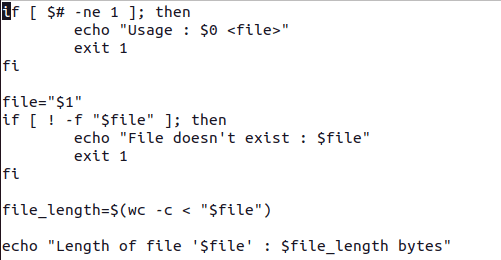
****

**OUTPUT-**



### To calculate length of a file

**IMPLEMENTATION –**

****

**OUTPUT-**



**PRACTICAL - 10**

**WRITE C++ PROGRAMS TO IMPLEMENT CPU SCHEDULING ALGORITHMS**

* **FCFS**

**Code:**

pair<int, vector<PCB>> FCFS(vector<PCB> &process\_list)

{

    sort\_arrival\_time\_then\_id(process\_list);

    int time = 0;

    cout << "0 | ";

    for (auto &current\_process : process\_list)

    {

        time += current\_process.burst\_time;

        cout << "P" << current\_process.process\_id << " | " << time << " | ";

        current\_process.completion\_time = time;

        current\_process.turnaround\_time = current\_process.completion\_time - current\_process.arrival\_time;

        current\_process.waiting\_time = current\_process.turnaround\_time - current\_process.burst\_time;

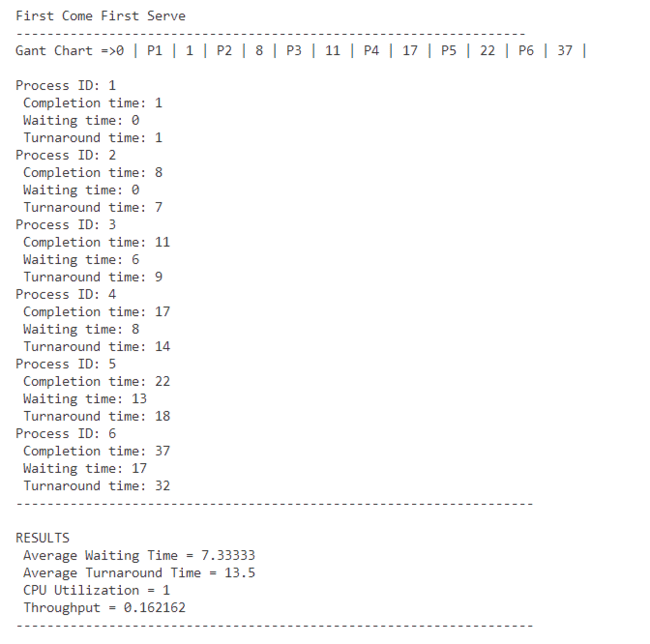
    }

    cout << endl;

    return {time, process\_list};

}

**Output:**

****

* **SJF NON-PREMPTIVE**

**Code:**

pair<int, vector<PCB>> SJF\_Non\_premptive(vector<PCB> &process\_list)

{

    int time = 0;

    cout << "0 | ";

    deque<PCB> remain\_processes(process\_list.begin(), process\_list.end());

    while (!remain\_processes.empty())

    {

        sort\_on\_sjf(remain\_processes, time);

        PCB current\_process = remain\_processes.front();

        remain\_processes.pop\_front();

        time += current\_process.burst\_time;

        cout << "P" << current\_process.process\_id << " | " << time << " | ";

        int id = current\_process.process\_id;

        int index = -1;

        for (int i = 0; i < process\_list.size(); i++)

        {

            if (id == process\_list[i].process\_id)

            {

                index = i;

                break;

            }

        process\_list[index].completion\_time = time;

        process\_list[index].turnaround\_time = process\_list[index].completion\_time - process\_list[index].arrival\_time;

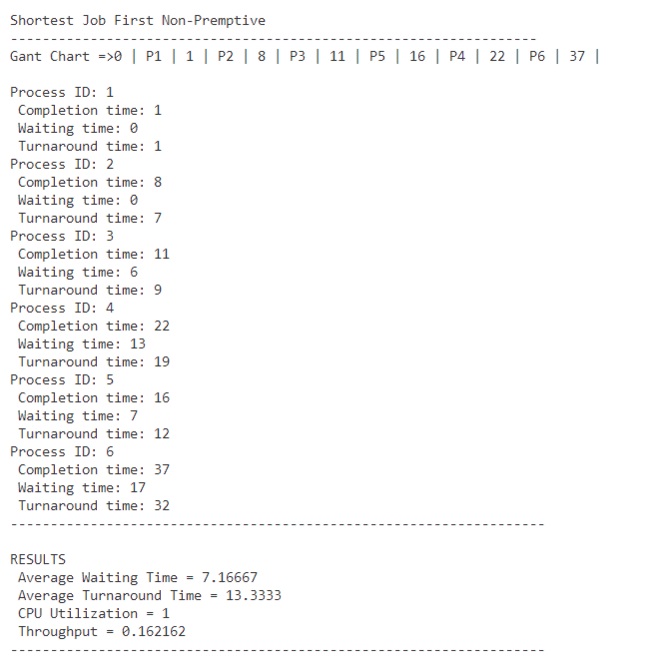
        process\_list[index].waiting\_time = process\_list[index].turnaround\_time - process\_list[index].burst\_time;

    }

    return {time, process\_list};

}

**Output:**

****

* **SJF PREMPTIVE**

**Code:**

pair<int, vector<PCB>> SJF\_premptive(vector<PCB> &process\_list)

{

    deque<PCB> remain\_processes(process\_list.begin(), process\_list.end());

    while (!remain\_processes.empty())

    {

        sort\_on\_sjf(remain\_processes, time);

        PCB current\_process = remain\_processes.front();

        current\_process.remaining\_burst -= 1;

        remain\_processes.pop\_front();

        time += 1;

        cout << "P" << current\_process.process\_id << " | " << time << " | ";

        int id = current\_process.process\_id;

        int index = -1;

        for (int i = 0; i < process\_list.size(); i++)

        {

            if (id == process\_list[i].process\_id)

            {

                index = i;

                break;

            }

        }

        if (current\_process.remaining\_burst > 0)

                    remain\_processes.push\_back(current\_process);

        else

        {

            process\_list[index].completion\_time = time;

            process\_list[index].turnaround\_time = process\_list[index].completion\_time - process\_list[index].arrival\_time;

            process\_list[index].waiting\_time = process\_list[index].turnaround\_time - process\_list[index].burst\_time;

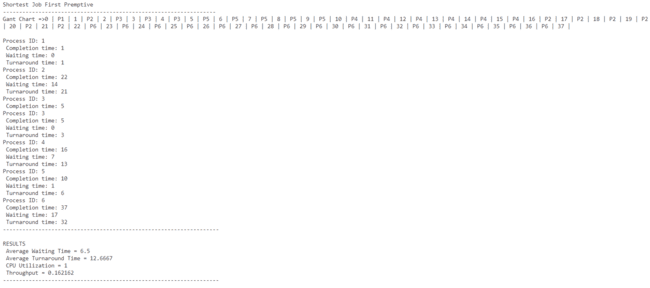
        }

    }

    return {time, process\_list};

}

**Output:**

****

* **PRIORITY NON-PREMPTIVE**

**Code:**

pair<int, vector<PCB>> priority\_Non\_Premptive(vector<PCB> &process\_list)

{

    deque<PCB> remaining\_process(process\_list.begin(), process\_list.end());

    int time = 0;

    cout << "0 |";

    while (!remaining\_process.empty())

    {

        sort\_priority(remaining\_process, time);

        PCB curr\_process = remaining\_process.front();

        remaining\_process.pop\_front();

        time += curr\_process.burst\_time;

        cout << "P" << curr\_process.process\_id << " | " << time << " | ";

        int id = curr\_process.process\_id;

        int index = -1;

        for (int i = 0; i < process\_list.size(); i++)

        {

            if (id == process\_list[i].process\_id)

            {

                index = i;

                break;

            }

        }

        process\_list[index].completion\_time = time;

        process\_list[index].turnaround\_time = process\_list[index].completion\_time - process\_list[index].arrival\_time;

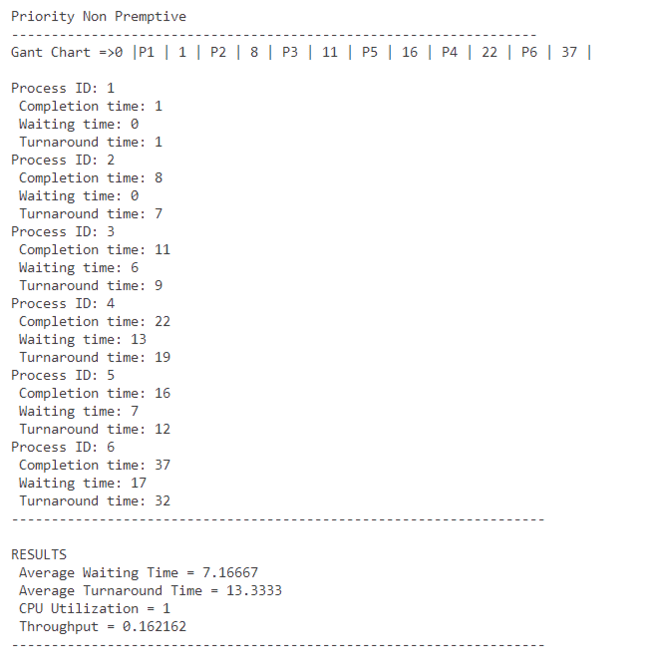
        process\_list[index].waiting\_time = process\_list[index].turnaround\_time - process\_list[index].burst\_time;

    }

    return {time, process\_list};

}

**Output:**

****

* **PRIORITY PREMPTIVE**

**Code:**

pair<int, vector<PCB>> priority\_Premptive(vector<PCB> &process\_list)

{

    int time = 0;

    cout << "0 | ";

    deque<PCB> remain\_processes(process\_list.begin(), process\_list.end());

    while (!remain\_processes.empty())

    {

        sort\_priority(remain\_processes, time);

        PCB current\_process = remain\_processes.front();

        current\_process.remaining\_burst -= 1;

        remain\_processes.pop\_front();

        time += 1;

        cout << "P" << current\_process.process\_id << " | " << time << " | ";

        if (current\_process.remaining\_burst > 0)

        {

            remain\_processes.push\_back(current\_process);

        }

        else

        {

            int id = current\_process.process\_id;

            int index = -1;

            for (int i = 0; i < process\_list.size(); i++)

            {

                if (id == process\_list[i].process\_id)

                {

                    index = i;

                    break;

                }

            }

            process\_list[index].completion\_time = time;

            process\_list[index].turnaround\_time = process\_list[index].completion\_time - process\_list[index].arrival\_time;

            process\_list[index].waiting\_time = process\_list[index].turnaround\_time - process\_list[index].burst\_time;

        }

    }

    return {time, process\_list};

}

# **PRACTICAL - 11**

## AIM – To implement banker’s algorithm

## 1.Safety Algorithm

### IMPLEMENTATION –

#include <iostream>

#include <vector>

using namespace std;

// Function to check if the system is in a safe state or not

bool isSafe(vector<vector<int>>& max, vector<vector<int>>& allocation, vector<int>& available, vector<int>& work, vector<bool>& finish) {

    int n = max.size(); // Number of processes

    int m = max[0].size(); // Number of resources

    vector<bool> safeSeq(n, false); // Initialize the safety sequence

    // Initialize the work vector with available resources

    for (int i = 0; i < m; ++i)

        work[i] = available[i];

    // Initialize the finish vector with false

    for (int i = 0; i < n; ++i)

        finish[i] = false;

    // Check for each process if it can finish

    for (int i = 0; i < n; ++i) {

        if (!finish[i]) {

            bool canAllocate = true;

            for (int j = 0; j < m; ++j) {

                if (max[i][j] - allocation[i][j] > work[j]) {

                    canAllocate = false;

                    break;

                }

            }

            if (canAllocate) {

                for (int j = 0; j < m; ++j)

                    work[j] += allocation[i][j];

                safeSeq[i] = true;

                finish[i] = true;

                i = -1; // Start over to check for processes again

            }

        }

    }

    // If all processes are in the safe sequence, return true, otherwise return false

    for (int i = 0; i < n; ++i) {

        if (!safeSeq[i])

            return false; }

    return true;

}

int main() {

    int n, m; // Number of processes and resources respectively

    cout << "Enter the number of processes: ";

    cin >> n;

    cout << "Enter the number of resources: ";

    cin >> m;

    // Declare and initialize the max and allocation matrices

    vector<vector<int>> max(n, vector<int>(m));

    vector<vector<int>> allocation(n, vector<int>(m));

    // Input for maximum resource requirement of each process

    cout << "Enter the maximum resource requirement for each process:\n";

    for (int i = 0; i < n; ++i) {

        cout << "For process " << i + 1 << " : ";

        for (int j = 0; j < m; ++j) {

            cin >> max[i][j];

        }

    }

    // Input for allocated resources for each process

    cout << "Enter the allocated resources for each process:\n";

    for (int i = 0; i < n; ++i) {

        cout << "For process " << i + 1 << " : ";

        for (int j = 0; j < m; ++j) {

            cin >> allocation[i][j];

        }

    }

    vector<int> available(m);

    cout << "Enter the available resources: ";

    for (int i = 0; i < m; ++i) {

        cin >> available[i];

    }

    // Check for safety and display the result

    vector<int> work(available);

    vector<bool> finish(n, false);

    if (isSafe(max, allocation, available, work, finish))

        cout << "System is in a safe state.\n";

    else

        cout << "System is not in a safe state.\n";

    return 0;

}

## 2. REQUEST AND RELEASE ALGORITHM

### **IMPLEMENTATION –**

#include <iostream>

#include <vector>

using namespace std;

// Function to check if the system is in a safe state or not

bool isSafe(vector<vector<int>>& max, vector<vector<int>>& allocation, vector<int>& available, vector<int>& work, vector<bool>& finish) {

    int n = max.size(); // Number of processes

    int m = max[0].size(); // Number of resources

    vector<bool> safeSeq(n, false); // Initialize the safety sequence

    for (int i = 0; i < m; ++i)

        work[i] = available[i];

    for (int i = 0; i < n; ++i)

        finish[i] = false;

    for (int i = 0; i < n; ++i) {

        if (!finish[i]) {

            bool canAllocate = true;

            for (int j = 0; j < m; ++j) {

                if (max[i][j] - allocation[i][j] > work[j]) {

                    canAllocate = false;

                    break;

                }

            }

            if (canAllocate) {

                for (int j = 0; j < m; ++j)

                    work[j] += allocation[i][j];

                safeSeq[i] = true;

                finish[i] = true;

                i = -1; // Start over to check for processes again

            }

        }

    }

    // If all processes are in the safe sequence, return true, otherwise return false

    for (int i = 0; i < n; ++i) {

        if (!safeSeq[i])

            return false;

    }

    return true;

}

int main() {

    int n, m; // Number of processes and resources respectively

    cout << "Enter the number of processes: ";

    cin >> n;

    cout << "Enter the number of resources: ";

    cin >> m;

    // Declare and initialize the max and allocation matrices

    vector<vector<int>> max(n, vector<int>(m));

    vector<vector<int>> allocation(n, vector<int>(m));

    // Input for maximum resource requirement of each process

    cout << "Enter the maximum resource requirement for each process:\n";

    for (int i = 0; i < n; ++i) {

        cout << "For process " << i + 1 << " : ";

        for (int j = 0; j < m; ++j) {

            cin >> max[i][j];

        }

    }

    // Input for allocated resources for each process

    cout << "Enter the allocated resources for each process:\n";

    for (int i = 0; i < n; ++i) {

        cout << "For process " << i + 1 << " : ";

        for (int j = 0; j < m; ++j) {

            cin >> allocation[i][j];

        }

    }

    // Input for available resources

    vector<int> available(m);

    cout << "Enter the available resources: ";

    for (int i = 0; i < m; ++i) {

        cin >> available[i];

    }

    // Check for safety and display the result

    vector<int> work(available);

    vector<bool> finish(n, false);

    if (isSafe(max, allocation, available, work, finish))

        cout << "System is in a safe state.\n";

    else

        cout << "System is not in a safe state.\n";

    return 0;

}

## 

## OUTPUT-

Enter the number of processes: 3

Enter the number of resources: 3

Enter the maximum resource requirement for each process:

For process 1 : 4

5

4

3

For process 3 : 3

4

5

Enter the allocated resources for each process:

For process 1 : 7

7

7

For process 2 :

PS E:\osAlgos\ProcessScheduling> g++ safetyAlgo.cpp

PS E:\osAlgos\ProcessScheduling> ./a.exe

Enter the number of processes: 3

Enter the number of resources: 3

Enter the maximum resource requirement for each process:

For process 1 : 5

5

5

For process 2 : 4

4

4

For process 3 : 3

3

3

Enter the allocated resources for each process:

For process 1 : 1

3

1

For process 2 : 2

2

2

For process 3 : 1

3

1

Enter the available resources: 6

6

6

System is in a safe state.

# **PRACTICAL - 12**

## AIM – C++ programming to implement Page Faults

### 1**.FIFO**

#### IMPLEMENTATION –

#include <iostream>

#include <vector>

#include <queue>

#include <unordered\_set>

using namespace std;

void fifoPageReplacement(vector<int>& pages, int capacity) {

    queue<int> pageFrame;

    unordered\_set<int> pageSet;

    int pageFaults = 0;

    for (int page : pages) {

        if (pageSet.find(page) == pageSet.end()) {

            pageFaults++;

            cout<<page<<" is not present in frame , page fault occurs."<<endl;

            if (pageFrame.size() == capacity) {

                int oldestPage = pageFrame.front();

                pageFrame.pop();

                pageSet.erase(oldestPage);

            }

            pageFrame.push(page);

            pageSet.insert(page);

        }

        else{

            cout<<page<<" is already present in page frame"<<endl;

        }

    }

    cout << "Total page faults using FIFO: " << pageFaults << endl;

}

int main() {

    int n;

    cout << "Enter the number of pages: ";

    cin >> n;

    vector<int> pages(n);

    cout << "Enter the page numbers to be requested: ";

    for (int i = 0; i < n; ++i)

        cin >> pages[i];

    int capacity;

    cout << "Enter the capacity of the page frame: ";

    cin >> capacity;

    fifoPageReplacement(pages, capacity);

}

#### OUTPUT -

Enter the number of pages: 7

Enter the page numbers to be requested: 34

2

6

9

1

53

6

Enter the capacity of the page frame: 3

34 is not present in frame , page fault occurs.

2 is not present in frame , page fault occurs.

6 is not present in frame , page fault occurs.

9 is not present in frame , page fault occurs.

1 is not present in frame , page fault occurs.

53 is not present in frame , page fault occurs.

6 is not present in frame , page fault occurs.

Total page faults using FIFO: 7

### **2.OPTIMAL PAGE REPLACEMENT**

#### IMPLEMENTATION –

#include <iostream>

#include <vector>

#include <unordered\_set>

#include <unordered\_map>

#include <limits>

using namespace std;

void optimalPageReplacement(const vector<int>& pages, int frameSize) {

    int pageFaults = 0;

    unordered\_set<int> frames;

    unordered\_map<int, int> nextUse;

    for (int i = 0; i < pages.size(); ++i) {

        int page = pages[i];

        cout<<"current page requested : "<<page<<endl;

        // Check if the page is already in the frame

        if (frames.find(page) == frames.end()) {

            cout<<page<<" is not available in frame"<<endl;

            ++pageFaults;

            // If the frame is not full, simply insert the page

            if (frames.size() < frameSize) {

                cout<<"frame space is available for page : "<<page<<endl;

                frames.insert(page);

            } else {

                // Find the page in the frame that won't be used for the longest period

                int pageToRemove = -1;

                int farthestUse = numeric\_limits<int>::min();

                for (int framePage : frames) {

                    if (nextUse[framePage] < i) {

                        pageToRemove = framePage;

                        break;

                    }

                    if (nextUse[framePage] > farthestUse) {

                        farthestUse = nextUse[framePage];

                        pageToRemove = framePage;

                    }

                }

                if(nextUse[pageToRemove]<i){

                    for (int removePage : frames) {

                    if (nextUse[removePage] < nextUse[pageToRemove]) {

                        pageToRemove = removePage;

                       }

                }

                }

                cout<<"page to remove is "<<pageToRemove<<endl;

                frames.erase(pageToRemove);

                frames.insert(page);

            }

            // Update next use of each page in the frame

        }

        for (int j=i+1; j<pages.size();j++) {

                if (pages[j] == page){

                     nextUse[page] = j;

                     break;

                }

                else{

                    nextUse[page]=i;

                }

                }

                for (int framePage : frames) {

                    cout<<framePage<<"\t";

                    }

                    cout<<endl;

    }

     cout << "Number of Page Faults using Optimal Page Replacement: " << pageFaults << endl;

}

int main() {

    int n;

    cout << "Enter the number of pages: ";

    cin >> n;

    vector<int> pages(n);

    cout << "Enter the page numbers to be requested: ";

    for (int i = 0; i < n; ++i)

        cin >> pages[i];

    int framesize;

    cout << "Enter the capacity of the page frame: ";

    cin >> framesize;

    optimalPageReplacement(pages,framesize);

    return 0;

}

#### **OUTPUT -**

Enter the number of pages: 10

Enter the page numbers to be requested: 2

4

1

3

5

2

4

7

2

4

Enter the capacity of the page frame: 4

current page requested : 2

2 is not available in frame

frame space is available for page : 2

2

current page requested : 4

4 is not available in frame

frame space is available for page : 4

4 2

current page requested : 1

1 is not available in frame

frame space is available for page : 1

1 4 2

current page requested : 3

3 is not available in frame

frame space is available for page : 3

3 1 4 2

current page requested : 5

5 is not available in frame

page to remove is 1

5 3 4 2

current page requested : 2

5 3 4 2

current page requested : 4

5 3 4 2

current page requested : 7

7 is not available in frame

page to remove is 3

5 4 7 2

current page requested : 2

5 4 7 2

current page requested : 4

5 4 7 2

Number of Page Faults using Optimal Page Replacement: 6

### **3.LEAST RECENTLY USED**

#### IMPLEMENTATION –

#include <bits/stdc++.h>

#include <limits>

using namespace std;

int leastRecentlyUsed(const vector<int>& pages, int frameSize) {

    int pageFaults = 0;

    unordered\_set<int> frames;

    unordered\_map<int, int> prevUse;

    for (int i = 0; i < pages.size(); ++i) {

        int page = pages[i];

        cout<<"page to insert is "<<page<<endl;

        if (frames.find(page) == frames.end()) {

            cout<<page<<" is not present in frame "<<endl;

            ++pageFaults;

            if (frames.size() < frameSize) {

                cout<<"frame space is availabale for "<<page<<" to insert"<<endl;

                frames.insert(page);

            } else {

                int pageToRemove = -1;

                int farthestUse = INT16\_MAX;

                for (int framePage : frames) {

                    if (prevUse[framePage] < farthestUse) {

                        farthestUse = prevUse[framePage];

                        pageToRemove = framePage;

                    }

                }

                cout<<"removed page is : "<<pageToRemove<<endl;

                frames.erase(pageToRemove);

                prevUse[pageToRemove]=INT16\_MAX;

                frames.insert(page);

            }

        }

        else{

            cout<<page<<" is already present in frame "<<endl;

        }

         prevUse[page] = i;}

    return pageFaults;

}

int main() {

    int n;

    cout << "Enter the number of pages: ";

    cin >> n;

    vector<int> pages(n);

    cout << "Enter the page numbers to be requested: ";

    for (int i = 0; i < n; ++i)

        cin >> pages[i];

    int frameSize;

    cout << "Enter the capacity of the page frame: ";

    cin >> frameSize;

    int pageFaults = leastRecentlyUsed(pages, frameSize);

    cout << "Number of Page Faults using LRU Page Replacement: " << pageFaults << endl;

}

#### **OUTPUT –**

Enter the number of pages: 10

Enter the page numbers to be requested: 7

1

2

3

1

4

2

5

3

6

Enter the capacity of the page frame: 4

page to insert is 7

7 is not present in frame

frame space is availabale for 7 to insert

page to insert is 1

1 is not present in frame

frame space is availabale for 1 to insert

page to insert is 2

2 is not present in frame

frame space is availabale for 2 to insert

page to insert is 3

3 is not present in frame

frame space is availabale for 3 to insert

page to insert is 1

1 is already present in frame

page to insert is 4

4 is not present in frame

removed page is : 7

page to insert is 2

2 is already present in frame

page to insert is 5

5 is not present in frame

removed page is : 3

page to insert is 3

3 is not present in frame

removed page is : 1

page to insert is 6

6 is not present in frame

removed page is : 4

Number of Page Faults using LRU Page Replacement: 8

# **PRACTICAL -15**

## AIM - C++ programs to implement Memory management algorithms (MVT)

### 1**.FIRST FIT**

#### IMPLEMENTATION –

#include <iostream>

#include <vector>

using namespace std;

class MemoryBlock {

    public:

    int id; // Process ID

    int size; // Size of the memory block

    bool allocated; // Indicates whether the block is allocated or free

};

// Function to allocate memory using First Fit algorithm

void firstFit(vector<MemoryBlock>& memory, int processId, int processSize) {

    for (int i = 0; i < memory.size(); ++i) {

        if (!memory[i].allocated && memory[i].size >= processSize) {

            memory[i].id = processId;

            memory[i].allocated = true;

            cout << "Process " << processId << " allocated memory starting at address " << i << " with size " << processSize << endl;

            return;

        }

    }

    cout << "Insufficient memory to allocate process " << processId;

}

// Function to deallocate memory

void deallocate(vector<MemoryBlock>& memory, int processId) {

    for (int i = 0; i < memory.size(); ++i) {

        if (memory[i].id == processId) {

            memory[i].id = -1;

            memory[i].allocated = false;

            cout << "Memory allocated to process " << processId << " deallocated successfully." << endl;

            return;

        }

    }

    cout << "Process " << processId << " not found." << endl;

}

// Function to display memory status

void displayMemory(const vector<MemoryBlock>& memory) {

    cout << "Memory Status:" << endl;

    for (int i = 0; i < memory.size(); ++i) {

        if (memory[i].allocated) {

            cout << "Address " << i << " - " << i + memory[i].size - 1 << ": Process " << memory[i].id << endl;

        } else {

            cout << "Address " << i << " - " << i + memory[i].size - 1 << ": Free" << endl;

        }

    }

}

int main() {

    vector<MemoryBlock> memory = {

        { -1, 5, false },  // Block 0: Free block of size 5

        { -1, 3, false },  // Block 1: Free block of size 3

        { -1, 6, false }   // Block 2: Free block of size 6

    };

    // Allocate memory for processes

    firstFit(memory, 1, 4); // Process 1: Allocate 4 units

    firstFit(memory, 2, 5); // Process 2: Allocate 5 units

    firstFit(memory, 3, 2); // Process 3: Allocate 2 units

    // Display memory status

    displayMemory(memory);

    // Deallocate memory

    deallocate(memory, 1); // Deallocate memory allocated to process 1

    deallocate(memory, 2); // Deallocate memory allocated to process 2

    // Display memory status after deallocation

    displayMemory(memory);

}

#### **OUTPUT –**

Process 1 allocated memory starting at address 0 with size 4

Process 2 allocated memory starting at address 2 with size 5

Process 3 allocated memory starting at address 1 with size 2

Memory Status:

Address 0 - 4: Process 1

Address 1 - 3: Process 3

Address 2 - 7: Process 2

Memory allocated to process 1 deallocated successfully.

Memory allocated to process 2 deallocated successfully.

Memory Status:

Address 0 - 4: Free

Address 1 - 3: Process 3

Address 2 - 7: Free

### **2. WORST FIT**

#### IMPLEMENTATION –

#include <iostream>

#include <vector>

using namespace std;

class MemoryBlock {

    public:

    int id; // Process ID

    int size; // Size of the memory block

    bool allocated; // Indicates whether the block is allocated or free

};

int sum(vector<MemoryBlock> memory , int i){

    int total = 0;

    for(int j=0;j<=i;j++){

        total += memory[j].size;

    }

    return total;

}

// Function to allocate memory using First Fit algorithm

void worstFit(vector<MemoryBlock>& memory, int processId, int processSize) {

    int allocatedSize = -1;

    int allocatedIndex = -1;

    for (int i = 0; i < memory.size(); ++i) {

        if (!memory[i].allocated && memory[i].size >= processSize && memory[i].size > allocatedSize) {

           allocatedIndex = i;

           allocatedSize = memory[i].size;

        }

    }

    if(allocatedIndex>=0){

         memory[allocatedIndex].id = processId;

            memory[allocatedIndex].allocated = true;

            cout << "Process " << processId << " allocated memory starting at address " << sum(memory,allocatedIndex-1) << " with size " << processSize << endl;

            return;

    }

    cout << "Insufficient memory to allocate process " << processId << endl;

}

// Function to deallocate memory

void deallocate(vector<MemoryBlock>& memory, int processId) {

    for (int i = 0; i < memory.size(); ++i) {

        if (memory[i].id == processId) {

            memory[i].id = -1;

            memory[i].allocated = false;

            cout << "Memory allocated to process " << processId << " deallocated successfully." << endl;

            return;

        }

    }

    cout << "Process " << processId << " not found." << endl;

}

// Function to display memory status

void displayMemory(const vector<MemoryBlock>& memory) {

    cout << "Memory Status:" << endl;

    for (int i = 0; i < memory.size(); ++i) {

        int startingAddress = sum(memory,i-1);

        int endingAddress = sum(memory,i);

        if (memory[i].allocated) {

            cout << "Address " << startingAddress << " - "<<endingAddress-1<< ": Process " << memory[i].id << endl;

        } else {

            cout << "Address " << startingAddress << " - " << endingAddress-1 << ": Free" << endl;

        }

    }

}

int main() {

    // Example memory blocks

    vector<MemoryBlock> memory = {

        { -1, 5, false },  // Block 0: Free block of size 5

        { -1, 3, false },  // Block 1: Free block of size 3

        { -1, 6, false }   // Block 2: Free block of size 6

    };

    // Allocate memory for processes

    worstFit(memory, 1, 4); // Process 1: Allocate 4 units

    worstFit(memory, 2, 5); // Process 2: Allocate 5 units

    worstFit(memory, 3, 2); // Process 3: Allocate 2 units

    // Display memory status

    displayMemory(memory);

    // Deallocate memory

    deallocate(memory, 1); // Deallocate memory allocated to process 1

    deallocate(memory, 2); // Deallocate memory allocated to process 2

    // Display memory status after deallocation

    displayMemory(memory);

    return 0;

}

#### **OUTPUT –**

Process 1 allocated memory starting at address 8 with size 4

Process 2 allocated memory starting at address 0 with size 5

Process 3 allocated memory starting at address 5 with size 2

Memory Status:

Address 0 - 4: Process 2

Address 5 - 7: Process 3

Address 8 - 13: Process 1

Memory allocated to process 1 deallocated successfully.

Memory allocated to process 2 deallocated successfully.

Memory Status:

Address 0 - 4: Free

Address 5 - 7: Process 3

Address 8 - 13: Free