Idris Ratlamwala

C31 - 2003145

**Experiment no. 1**

**Problem statement :** Exploring internal commands in Linux.

**Theory :**

The Linux command is a utility of the Linux operating system. All basic and advanced tasks can be done by executing commands. The commands are executed on the **Linux terminal**. The terminal is a command-line interface to interact with the system, which is similar to the command prompt in the Windows OS. Commands in Linux are ***case-sensitive***.

[Linux](https://www.javatpoint.com/linux-tutorial) provides a powerful command-line interface compared to other operating systems such as [Windows](https://www.javatpoint.com/windows) and MacOS. We can do basic work and advanced work through its terminal. We can do some basic tasks such as creating a file, deleting a file, moving a file, and more. In addition, we can also perform advanced tasks such as administrative tasks (including package installation, user management), networking tasks (ssh connection), security tasks, and many more.

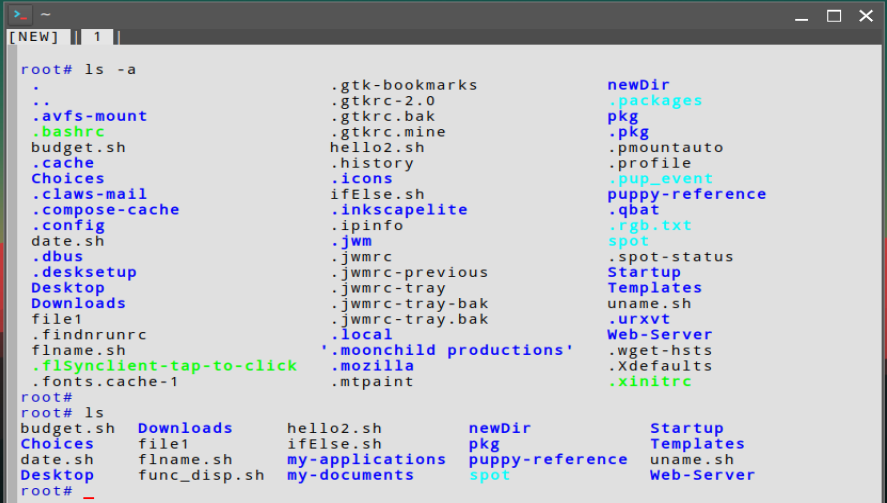
Letting the application to run Linux commands automatically for us has many obvious benefits. However, the disadvantage is that in many cases we are unable to customise the command execution to craft it towards the desired result. Furthermore, if something goes wrong, the user is most likely left in the dark without any hint on how to even begin to troubleshoot the issue.

**Linux Commands :**

1. **ls :**

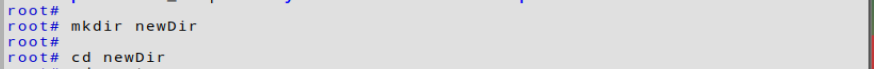
The ls command is used to list files or directories in Linux and other

Unix-based operating systems. Just like you navigate in your File Explorer or Finder with a GUI, the ls command allows you to list all files or directories in the current directory by default, and further interact with them via the command line.



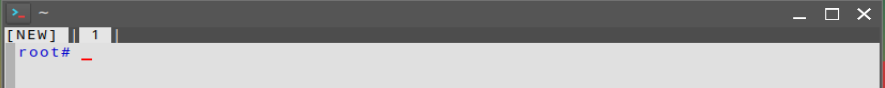
1. **mkdir :**

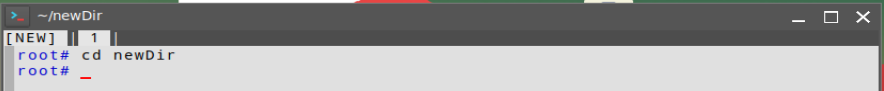
The mkdir command in Linux/Unix allows users to create or make new directories. mkdir stands for “make directory”. With mkdir, you can also set permissions, create multiple directories (folders) at once, and much more.

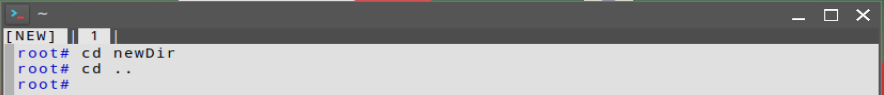


1. **cd :**

cd command in Linux known as change directory command. It is used to change the current working directory. It is one of the most basic and frequently used commands when working on the Linux terminal. The syntax for the cd command is as follows: cd [OPTIONS] directory.







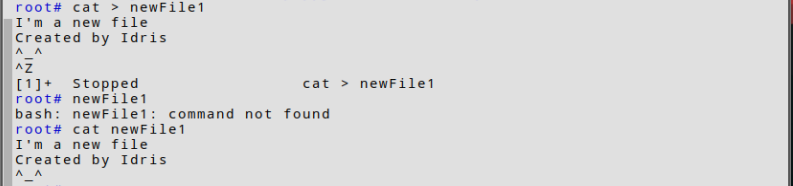
1. **rmdir :**

rmdir command is used to remove empty directories from the filesystem in Linux. The rmdir command removes each and every directory specified in the command line only if these directories are empty. So if the specified directory has some directories or files in it then this cannot be removed by the rmdir command.



1. **cat :**

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

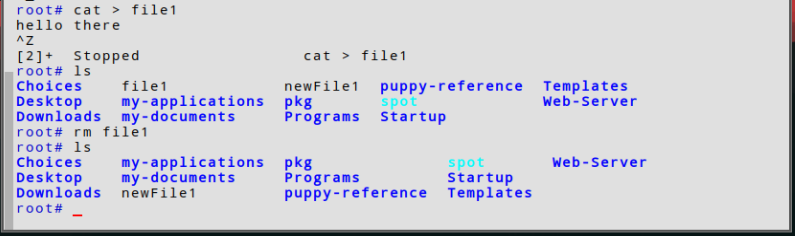


1. **rm :**

The 'rm' means to remove. This command is used to remove a file. The command line doesn't have a recycle bin or trash unlike other GUIs to recover the files. Hence, be very much careful while using this command. mv stands for the move. mv is used to move one or more files or directories from one place to another in a file system like UNIX. It has two distinct functions:

* 1. It renames a file or folder.
  2. It moves a group of files to a different directory.

No additional space is consumed on a disk during renaming. This command normally works silently means no prompt for confirmation.

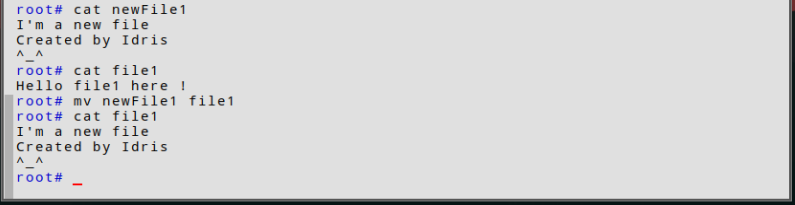


1. **mv :**

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

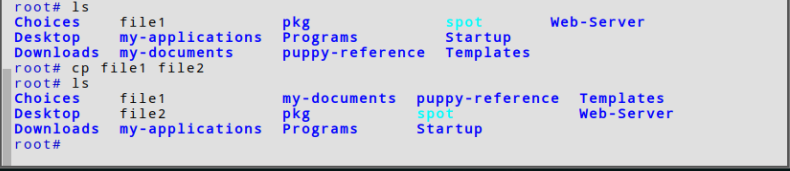
* 1. It renames a file or folder.
  2. It moves a group of files to a different directory.

No additional space is consumed on a disk during renaming. This command normally works silently means no prompt for confirmation.



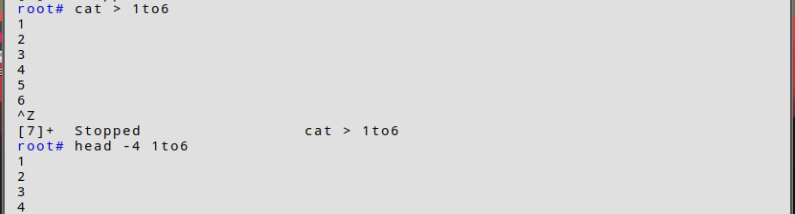
1. **cp :**

cp stands for copy. This command is used to copy files or groups of files or directories. It creates an exact image of a file on a disk with a different file name. cp command requires at least two filenames in its arguments.



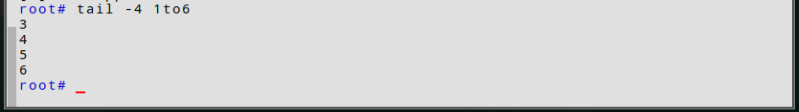
1. **head :**

The 'head' command displays the starting content of a file. By default, it displays starting 10 lines of any file. It writes results to standard output.



1. **tail :**

t is the complementary of head command. The tail command, as the name implies, prints the last N number of data of the given input. By default, it prints the last 10 lines of the specified files. If more than one file name is provided then data from each file is preceded by its file name.



1. **sort :**

SORT command is used to sort a file, arranging the records in a particular order. By default, the sort command sorts file assuming the contents are ASCII. Using options in the sort command can also be used to sort numerically.



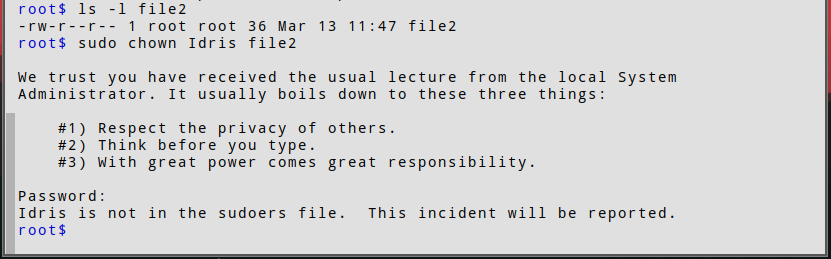
1. **wc :**

On Linux and Unix-like operating systems, the wc command allows you to count the number of lines, words, characters, and bytes of each given file or standard input and print the result.



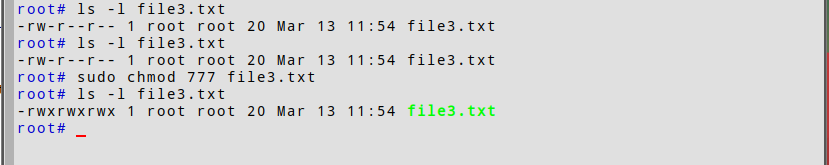
1. **chown :**

The chown command allows you to change the user and/or group ownership of a given file, directory, or symbolic link.



1. **chmod :**

In Unix-like operating systems, the chmod command is used to change the access mode of a file. The name is an abbreviation of change mode.

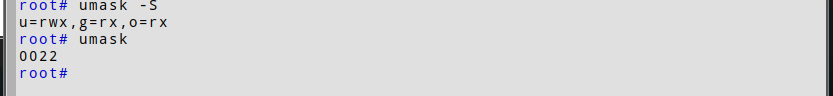


1. **chgrp :**

chgrp command in Linux is used to change the group ownership of a file or directory. All files in Linux belong to an owner and a group. You can set the owner by using “chown” command, and the group by the "chgrp" command.

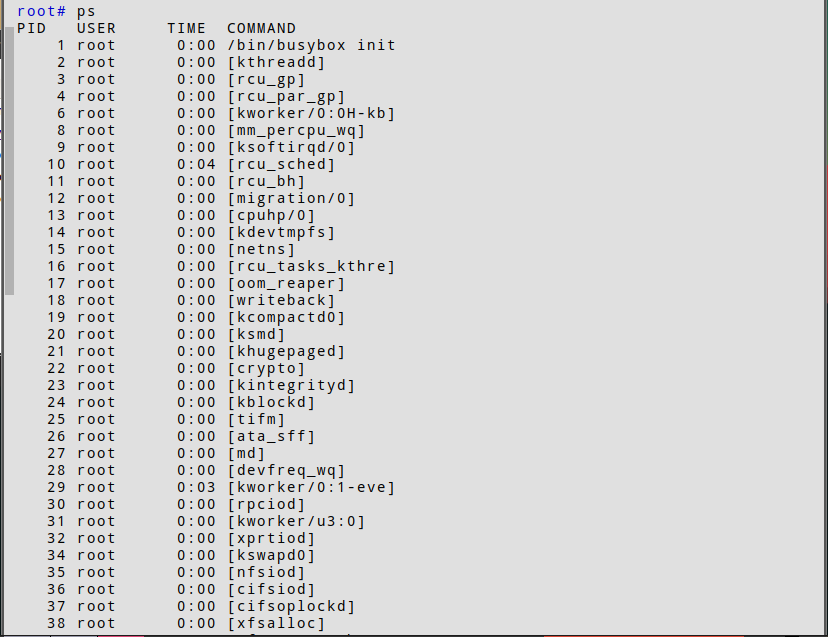
1. **unmask :**

Umask, or the user file-creation mode is a Linux command that is used to assign the default file permission sets for newly created folders and files. The term mask references the grouping of the permission bits, each of which defines how its corresponding permission is set for newly created files.



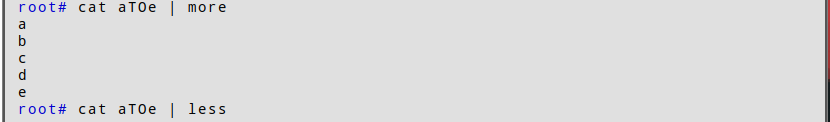
1. **ps :**

For viewing information related to the process on a system which stands for process status.



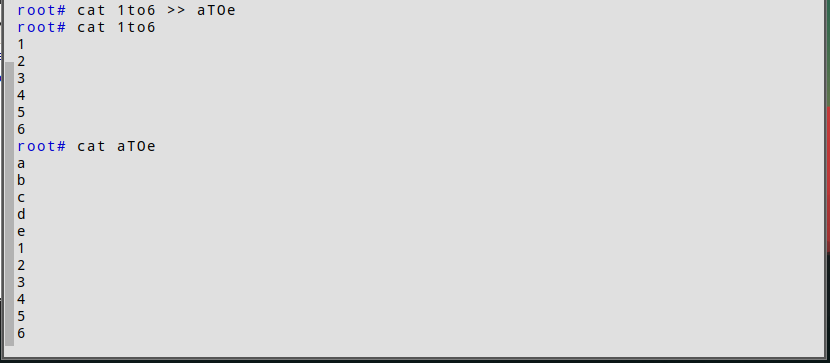
1. **pipe :**

n Linux, the pipe command lets you send the output of one command to another. Piping, as the term suggests, can redirect the standard output, input, or error of one process for further processing.



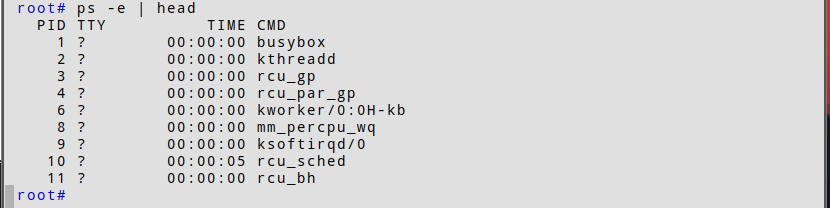
1. **Redirection operators :**

A redirection operator is a special character that can be used with a command, like a Command Prompt command or DOS command, to either redirect the input to the command or the output from the command.

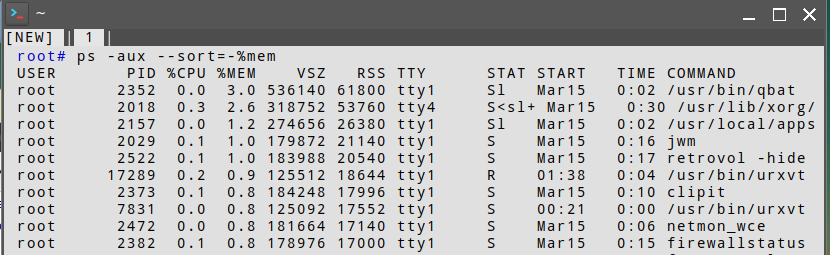


1. **Explore commands for the following :**

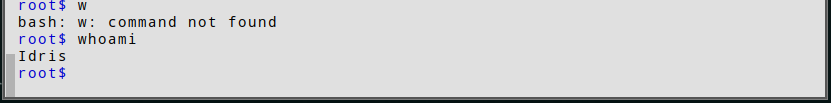
* **Display top 10 proccess in descending order**



* **Display the proccess with the highest memory usage**

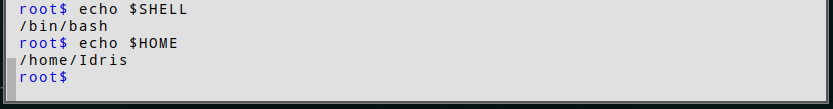


* **Display current user logged in and logname**

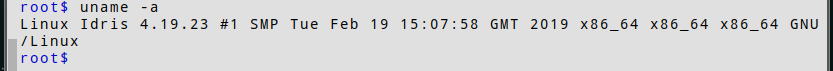


* **Display**

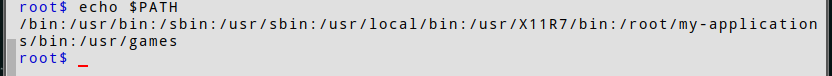
**Current shell & home dir**



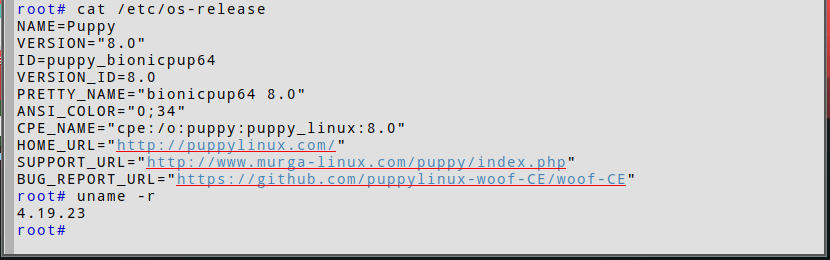
**OS type**



**Current path settings and working directory**



**OS version, release number, kernel version**



**Program :**

#include<stdio.h>

struct process

{

    int id,WT,AT,BT,TAT,PR;

};

struct process a[10];

void swap(int \*b,int \*c)

{

    int tem;

    tem=\*c;

    \*c=\*b;

    \*b=tem;

}

int main()

{

    int n,check\_ar=0;

    int Cmp\_time=0;

    float Total\_WT=0,Total\_TAT=0,Avg\_WT,Avg\_TAT;

    printf("Enter the number of process \n");

    scanf("%d",&n);

    printf("Enter the Arrival time , Burst time and priority of the process\n");

    printf("AT BT PR\n");

    for(int i=0;i<n;i++)

    {

        scanf("%d%d%d",&a[i].AT,&a[i].BT,&a[i].PR);

        a[i].id=i+1;

        if(i==0)

         check\_ar=a[i].AT;

        if(check\_ar!=a[i].AT )

         check\_ar=1;

    }

    if(check\_ar!=0)

    {

        for(int i=0;i<n;i++)

        {

            for(int j=0;j<n-i-1;j++)

            {

                if(a[j].AT>a[j+1].AT)

                {

                      swap(&a[j].id,&a[j+1].id);

                      swap(&a[j].AT,&a[j+1].AT);

                      swap(&a[j].BT,&a[j+1].BT);

                      swap(&a[j].PR,&a[j+1].PR);

                }

            }

        }

    }

    if(check\_ar!=0)

    {

        a[0].WT=a[0].AT;

        a[0].TAT=a[0].BT-a[0].AT;

        Cmp\_time=a[0].TAT;

        Total\_WT=Total\_WT+a[0].WT;

        Total\_TAT=Total\_TAT+a[0].TAT;

        for(int i=1;i<n;i++)

        {

            int min=a[i].PR;

            for(int j=i+1;j<n;j++)

            {

                if(min>a[j].PR && a[j].AT<=Cmp\_time)

                {

                      min=a[j].PR;

                      swap(&a[i].id,&a[j].id);

                      swap(&a[i].AT,&a[j].AT);

                      swap(&a[i].BT,&a[j].BT);

                      swap(&a[i].PR,&a[j].PR);

                }

            }

            a[i].WT=Cmp\_time-a[i].AT;

            Total\_WT=Total\_WT+a[i].WT;

            Cmp\_time=Cmp\_time+a[i].BT;

            a[i].TAT=Cmp\_time-a[i].AT;

            Total\_TAT=Total\_TAT+a[i].TAT;

        }

    }

    else

    {

        for(int i=0;i<n;i++)

        {

            int min=a[i].PR;

            for(int j=i+1;j<n;j++)

            {

                if(min>a[j].PR && a[j].AT<=Cmp\_time)

                {

                    min=a[j].PR;

                      swap(&a[i].id,&a[j].id);

                      swap(&a[i].AT,&a[j].AT);

                      swap(&a[i].BT,&a[j].BT);

                       swap(&a[i].PR,&a[j].PR);

                }

            }

            a[i].WT=Cmp\_time-a[i].AT;

            Cmp\_time=Cmp\_time+a[i].BT;

            a[i].TAT=Cmp\_time-a[i].AT;

            Total\_WT=Total\_WT+a[i].WT;

            Total\_TAT=Total\_TAT+a[i].TAT;

        }

    }

    Avg\_WT=Total\_WT/n;

    Avg\_TAT=Total\_TAT/n;

    printf("The process are\n");

    printf("ID\tWT\tTAT\n");

    for(int i=0;i<n;i++) printf("%d\t%d\t%d\n",a[i].id,a[i].WT,a[i].TAT);

    printf("Avg waiting time is: %f\n",Avg\_WT);

    printf("Avg turn around time is: %f\n",Avg\_TAT);

    return 0;

}

**Output :**

