FACULTY OF ENGINEERING & TECHNOLOGY

BACHELOR OF TECHNOLOGY

OPERATING SYSTEMS LAB

(303105252)

4th SEMESTER

Department of Artificial Intelligence

LABORATORY MANUAL 2024-25

CERTIFICATE

This is to certified that Mr. Meet Limbachiya with Enrolment No: 2303031240589 has successfully completed his/her laboratory experiments in the OPERATING SYSTEM (303105252) from the department of AI & AIDS during the academic year 2024-2025.

Date of Submission:

Staff in charge:

Head of Department:

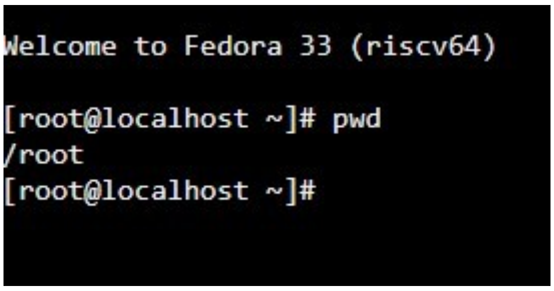
INDEX

PRACTICAL – 1

Aim: Study of Basic commands of Linux.

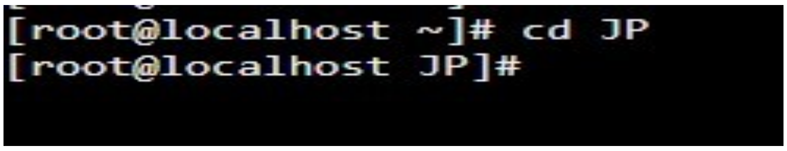
Procedure:

1. **PWD**
   - Description: The pwd Linux command prints the current working directory path, starting from the root (/). Use the pwd command to find your way in the Linux file system structure maze or to pass the working directory in a Bash script. In this tutorial, you will learn to use the pwd command.
   - Syntax: pwd



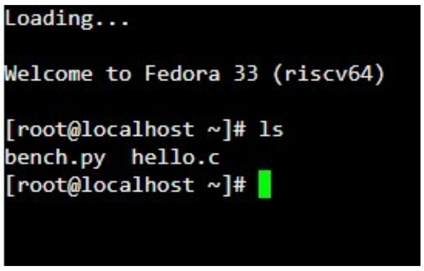
```
Welcome to Fedora 33 (riscv64)

[root@localhost ~]# pwd
/root
[root@localhost ~]#
```

2. **CD**
   - Description: The cd command is used to change the current directory in both Linux and other Unix-like systems.
   - Syntax: cd [directory]



```
[root@localhost ~]# cd JP
[root@localhost JP]#
```

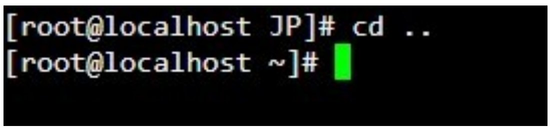
3. **LS**
   - Description: we use ls command to list files and directories. This command will print all the file and directories in the current directory.
   - Syntax: ls [directory]



```
Loading...

Welcome to Fedora 33 (riscv64)

[root@localhost ~]# ls
bench.py  hello.c
[root@localhost ~]# 
```

4. CD ..

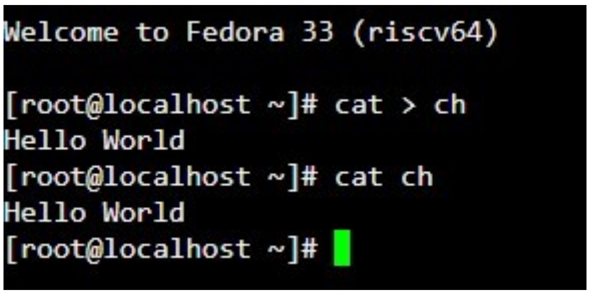
- Description: This command is used to move to the parent directory of current directory, or the directory one level up from the current directory. ".." represents parent directory.
- Syntax: cd ..



```
[root@localhost JP]# cd ..
[root@localhost ~]# 
```

5. CAT

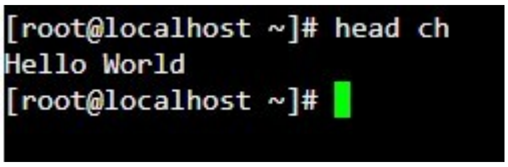
- Description: The cat command is a utility command in Linux. One of its most common usages is to print the content of a file onto the standard output stream. Other than that, the cat command also allows us to write some texts into a file.
- Syntax: cat [file-name]



```
Welcome to Fedora 33 (riscv64)

[root@localhost ~]# cat > ch
Hello World
[root@localhost ~]# cat ch
Hello World
[root@localhost ~]# 
```

6. HEAD

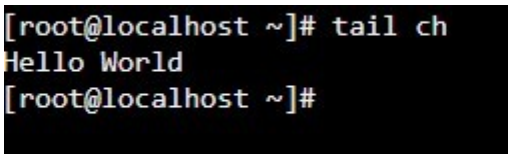
- Description: The head command, as the name implies, print the top N number of data of the given input. By default, it prints the first 10 lines of the specified files. If more than one file name is provided then data from each file is preceded by its file name.
- Syntax: head [option] [file]



```
[root@localhost ~]# head ch
Hello World
[root@localhost ~]#
```

7. TAIL

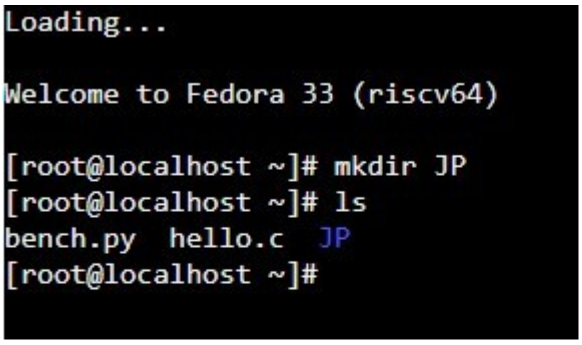
- Description: Tail is a command which prints the last few numbers of lines (10 lines by default) of a certain file, then terminates. By default, "tail" prints the last 10 lines of a file, then exits. as you can see, this prints the last 10 lines of /var/log/messages.
- Syntax: tail [option] [file]



```
[root@localhost ~]# tail ch
Hello World
[root@localhost ~]#
```

8. MKDIR

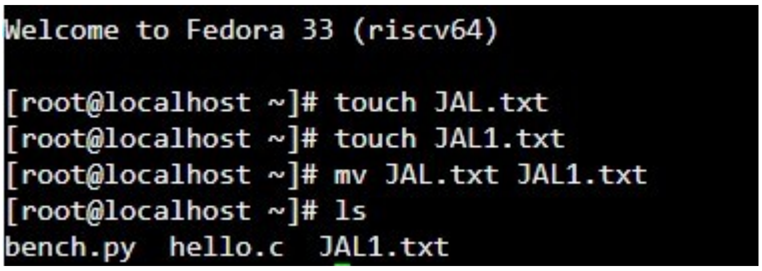
- Description: The mkdir command in Linux/Unix allows users to create or make new directories. mkdir stands for "make directory." With mkdir , you can also set permissions, create multiple directories (folders) at once, and much more.
- Syntax: mkdir [directory name]



```
Loading...

Welcome to Fedora 33 (riscv64)

[root@localhost ~]# mkdir JP
[root@localhost ~]# ls
bench.py   hello.c   JP
[root@localhost ~]#
```

9. MV

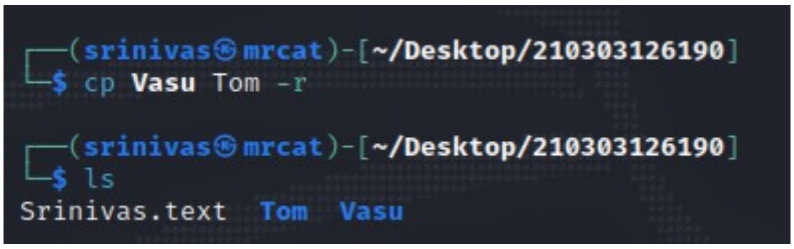
- Description: The mv command termed as "Move", which is a command-line utility to move files or directories from source to target. It supports the moving of a single file, multiple files, and directories.
- Syntax: mv [option] source destination



```
Welcome to Fedora 33 (riscv64)

[root@localhost ~]# touch JAL.txt
[root@localhost ~]# touch JAL1.txt
[root@localhost ~]# mv JAL.txt JAL1.txt
[root@localhost ~]# ls
bench.py   hello.c   JAL1.txt
```

10. CP

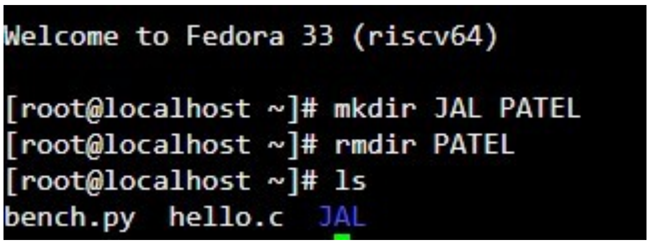
- Description: cp command copies files (or, optionally, directories). The copy is completely independent of the original. You can either copy one file to another, or copy arbitrarily many files to a destination directory. In the first format, when two file names are given, cp command copies SOURCE file to DEST file.
- Syntax: cp [option] source destination



```
┌──(srinivas㉿mrcat)-[~/Desktop/210303126190]
└─$ cp Vasu Tom -r

┌──(srinivas㉿mrcat)-[~/Desktop/210303126190]
└─$ ls
Srinivas.text  Tom  Vasu
```

11. RMDIR

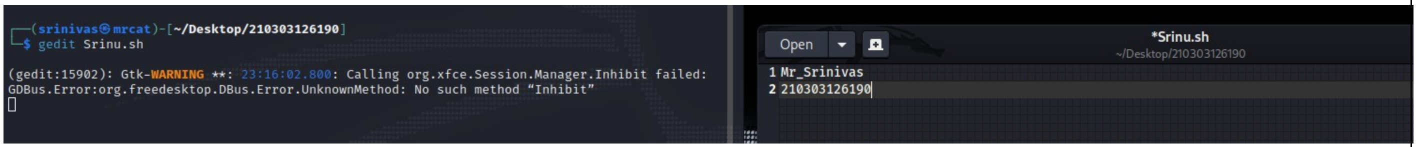
- Description: mdir command is used remove empty directories from the filesystem in Linux. The rmdir command removes each and every directory specified in the command line only if these directories are empty. So if the specified directory has some directories or files in it then this cannot be removed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



```
Welcome to Fedora 33 (riscv64)

[root@localhost ~]# mkdir JAL PATEL
[root@localhost ~]# rmdir PATEL
[root@localhost ~]# ls
bench.py   hello.c   JAL
```

12. GEDIT

- Description: The gedit command is used to create and open a file
- Syntax: gedit filename.txt

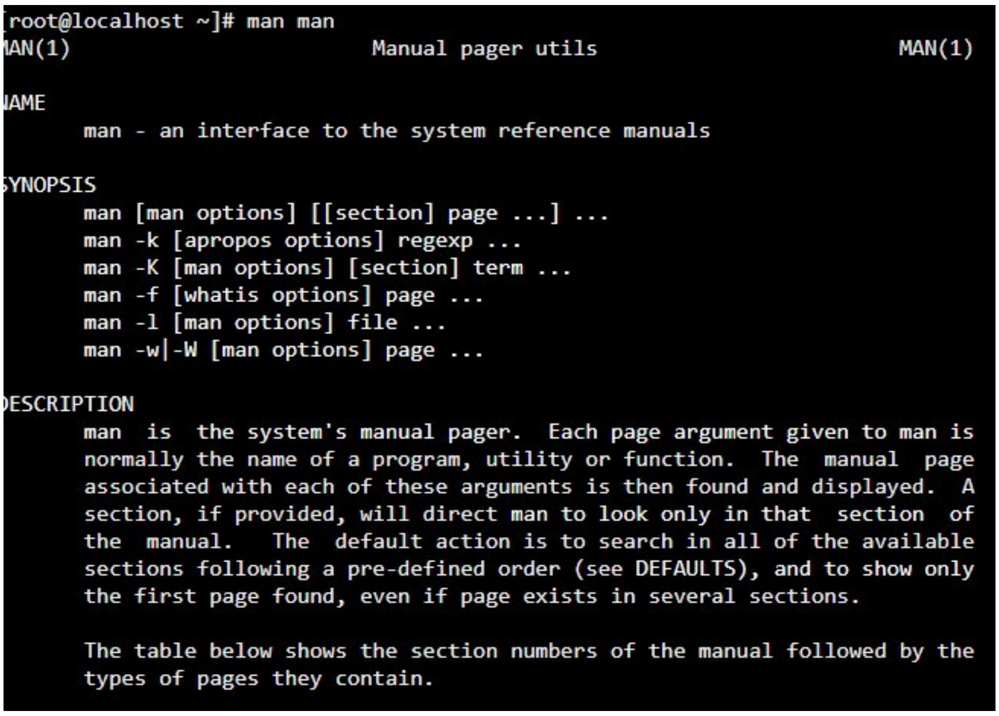


```
┌──(srinivas㉿mrcat)-[~/Desktop/210303126190]
└─$ gedit Srinu.sh

(gedit:15902): Gtk-WARNING **: 23:16:02.800: Calling org.xfce.Session.Manager.Inhibit failed:
GDBus.Error:org.freedesktop.DBus.Error.UnknownMethod: No such method “Inhibit”
```

```
*Srinu.sh
~/Desktop/210303126190
Mr_Srinivas
210303126190
```

13. MAN

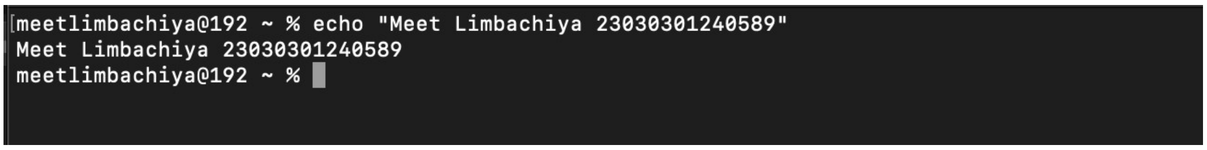
- Description: man command in Linux is used to display the user manual of any command that we can run on the terminal. It provides a detailed view of the command which includes NAME, SYNOPSIS, DESCRIPTION, OPTIONS, EXIT STATUS, RETURN VALUES, ERRORS, FILES, VERSIONS, EXAMPLES, AUTHORS
- Syntax: man command



```
[root@localhost ~]# man man
AN(1)                         Manual pager utils                         MAN(1)

AME
       man - an interface to the system reference manuals

YNOPSIS
       man [man options] [[section] page ...] ...
       man -k [apropos options] regexp ...
       man -K [man options] [section] term ...
       man -f [whatis options] page ...
       man -l [man options] file ...
       man -w|-W [man options] page ...

ESCRIPTION
       man  is  the system's manual pager.  Each page argument given to man is
       normally the name of a program, utility or function.  The  manual  page
       associated with each of these arguments is then found and displayed.  A
       section, if provided, will direct man to look only in that  section  of
       the  manual.   The  default action is to search in all of the available
       sections following a pre-defined order (see DEFAULTS), and to show only
       the first page found, even if page exists in several sections.

       The table below shows the section numbers of the manual followed by the
       types of pages they contain.
```

14. ECHO

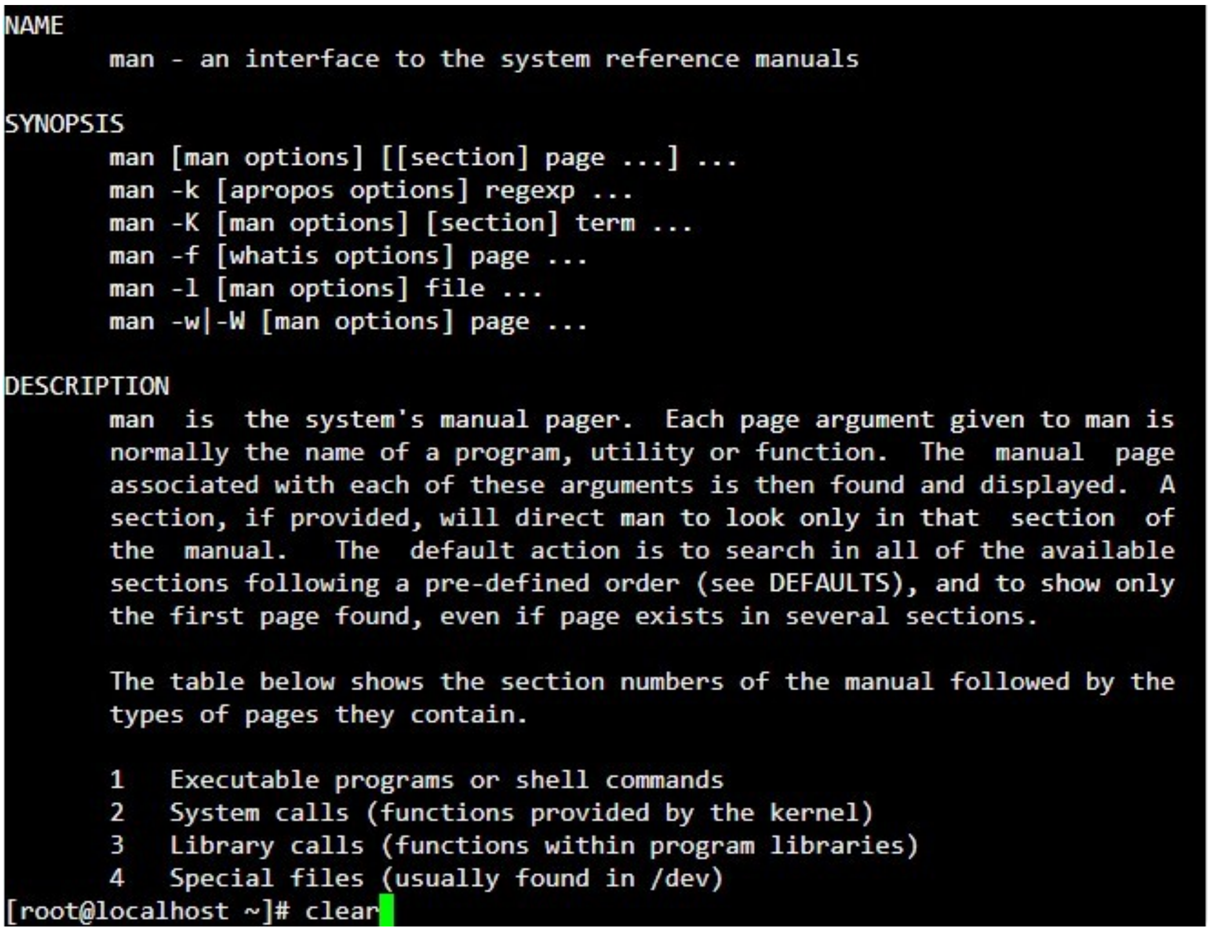
- Description: Display text on the screen
- Syntax: Display text on the screen



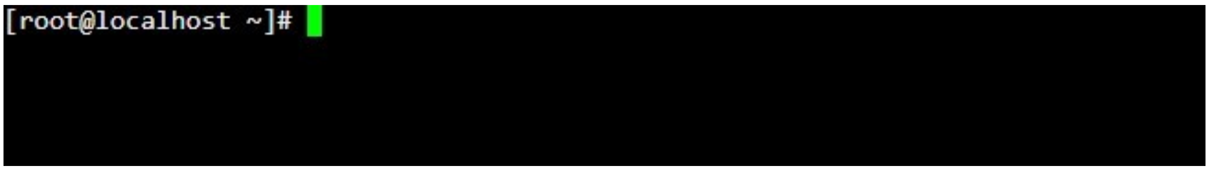
```
meetlimbachiya@192 ~ % echo "Meet Limbachiya 23030301240589"
Meet Limbachiya 23030301240589
meetlimbachiya@192 ~ %
```

15. CLEAR

- Description: Used to clear the screen
- Syntax: clear



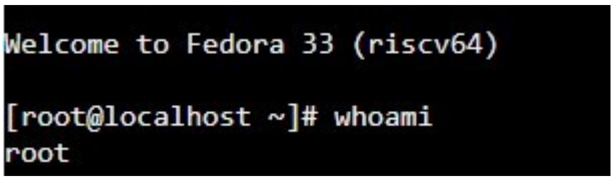
```
NAME
       man - an interface to the system reference manuals

SYNOPSIS
       man [man options] [[section] page ...] ...
       man -k [apropos options] regexp ...
       man -K [man options] [section] term ...
       man -f [whatis options] page ...
       man -l [man options] file ...
       man -w|-W [man options] page ...

DESCRIPTION
       man  is  the system's manual pager.  Each page argument given to man is
       normally the name of a program, utility or function.  The  manual  page
       associated with each of these arguments is then found and displayed.  A
       section, if provided, will direct man to look only in that  section  of
       the  manual.   The  default action is to search in all of the available
       sections following a pre-defined order (see DEFAULTS), and to show only
       the first page found, even if page exists in several sections.

       The table below shows the section numbers of the manual followed by the
       types of pages they contain.

       1   Executable programs or shell commands
       2   System calls (functions provided by the kernel)
       3   Library calls (functions within program libraries)
       4   Special files (usually found in /dev)
[root@localhost ~]# clear
```



```
[root@localhost ~]#
```

16. WHOAMI

- Description: whoami prints the effective user ID. This command prints the username associated with the current effective user ID
- Syntax: whoami [option]



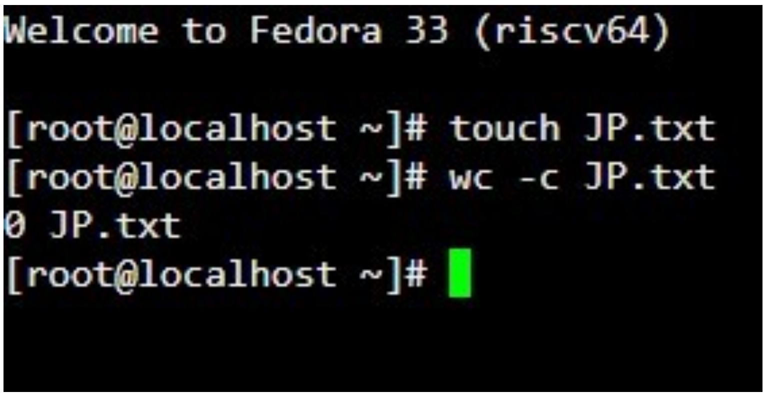
```
Welcome to Fedora 33 (riscv64)

[root@localhost ~]# whoami
root
```

17. WC

- Description: wc (word count) command, can return the number of lines, words, and characters in a file.
- Syntax: wc [option]… [file]…

Example:

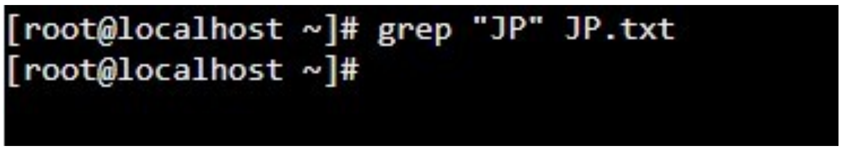
- ✓ Print the byte counts of file myfile.txt
  wc -c myfile.txt
- ✓ Print the line counts of file myfile.tx
  wc -l myfile.txt
- ✓ Print the word counts of file myfile.txt
  wc -w myfile.txt



```
Welcome to Fedora 33 (riscv64)

[root@localhost ~]# touch JP.txt
[root@localhost ~]# wc -c JP.txt
0 JP.txt
[root@localhost ~]#
```

18. GREP

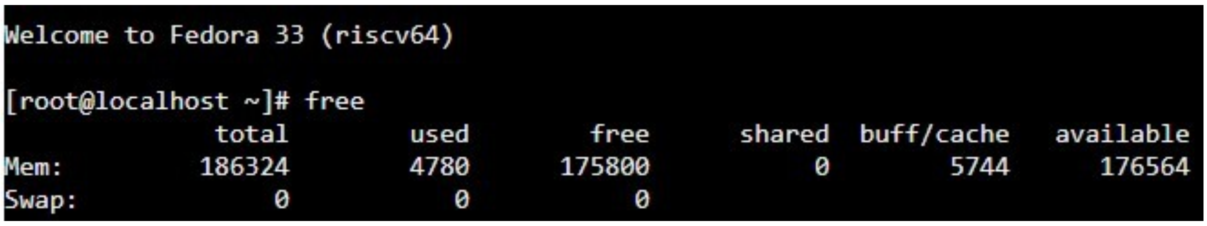
- Description: grep command uses a search term to look through a file
- Syntax: grep [option]… Pattern [file]



```
[root@localhost ~]# grep "JP" JP.txt
[root@localhost ~]#
```

19. FREE

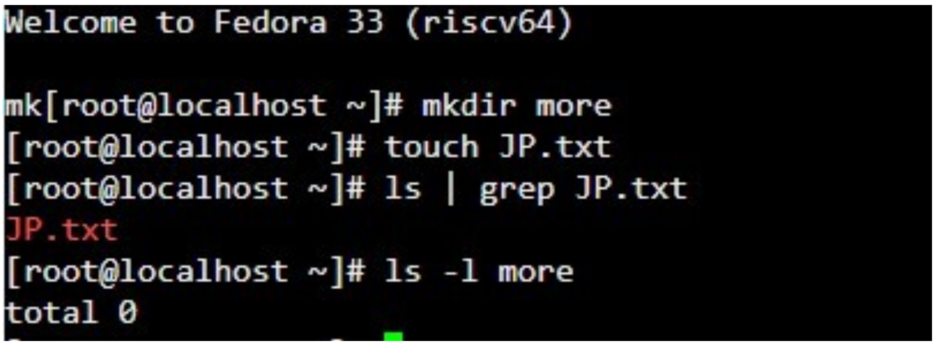
- Description: To display the RAM details in Linux machine need to write following command.
- Syntax: free



```
Welcome to Fedora 33 (riscv64)

[root@localhost ~]# free
              total        used        free      shared  buff/cache   available
Mem:         186324        4780      175800           0        5744      176564
Swap:             0           0           0
```

20. PIPE (|)

- Description: Pipe command is used to send output of one program as a input to another. Pipes "|" help combine 2 or more commands
- Syntax: Command 1 | command 2



```
Welcome to Fedora 33 (riscv64)

mk[root@localhost ~]# mkdir more
[root@localhost ~]# touch JP.txt
[root@localhost ~]# ls | grep JP.txt
JP.txt
[root@localhost ~]# ls -l more
total 0
```

PRACTICAL – 2

Aim: Study the basics of Shell programming.

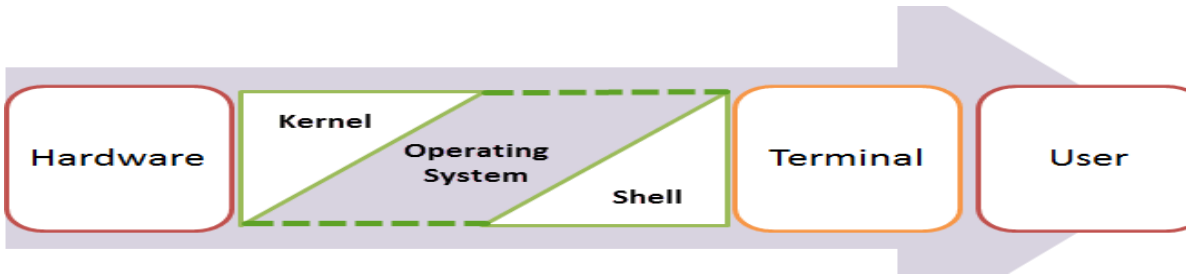
What is a Shell?

It is a list of commands in a computer program that is run by the Unix shell which is a command line interpreter. A shell script usually has comments that describe the steps.

An Operating is made of many components,

But its two prime components are –

- ✓ Kernel
- ✓ Shell



A Kernel is at the nucleus of a computer. It makes the communication between the hardware and software possible. While the Kernel is the innermost part of an operating system, a shell is the outermost one. A shell in a Linux operating system takes input from you in the form of commands, processes it, and then gives an output. It is the interface through which a user works on the programs, commands, and scripts. A shell is accessed by a terminal which runs it. When you run the terminal, the Shell issues a command prompt (usually $), where you can type your input, which is then executed when you hit the Enter key. The output or the result is thereafter displayed on the terminal. The Shell wraps around the delicate interior of an Operating system protecting it from accidental damage. Hence the name Shell.

Types Of Shells:

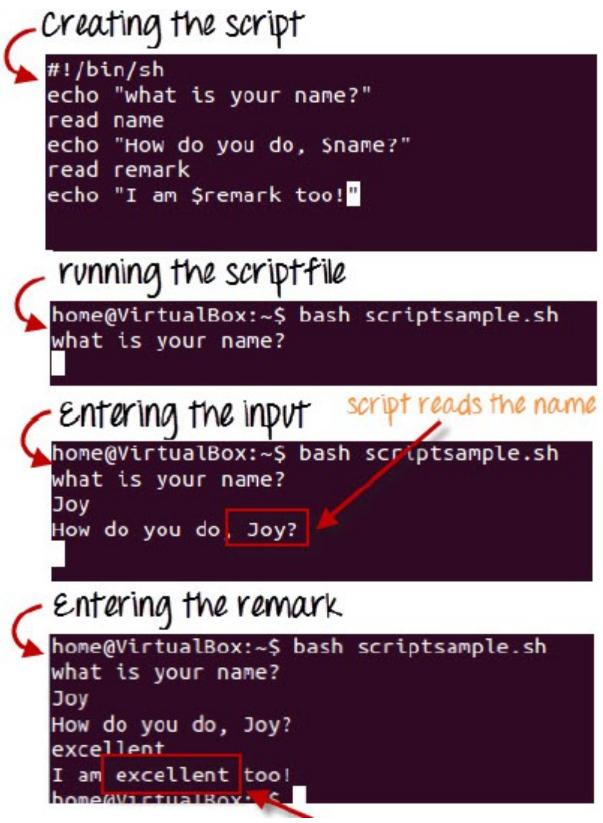
1. **Bournee shell:** This is default shell for version 7 unix. The character $ is the default prompt for the bourne shell.
2. **C shell:** This is a unix shell and a command processor that is run in a text window . The character % is the default prompt for the C shell. File commands can also be read easily by the C shell , which is known as a script.

How to create file in linux:

In Linux there are two commands which are used to create the files in Linux:

1. Gedit
2. nano

What is Shell Scripting?



```
#!/bin/sh
echo "what is your name?"
read name
echo "How do you do, $name?"
read remark
echo "I am $remark too!"
```

```
home@VirtualBox:~$ bash scriptsample.sh
what is your name?
```

```
home@VirtualBox:~$ bash scriptsample.sh
what is your name?
Joy
How do you do, Joy?
```

```
home@VirtualBox:~$ bash scriptsample.sh
what is your name?
Joy
How do you do, Joy?
excellent
I am excellent too!
home@VirtualBox:~$
```

Shell scripting is writing a series of command for the shell to execute. It can combine lengthy and repetitive sequences of commands into a single and simple script, which can be stored and executed anytime. This reduces the effort required by the end user. Let us understand the steps in creating a Shell Script

1. Create a file using a vi editor(or any other editor). Name script file with extension .sh

2. Start the script with #! /bin/sh

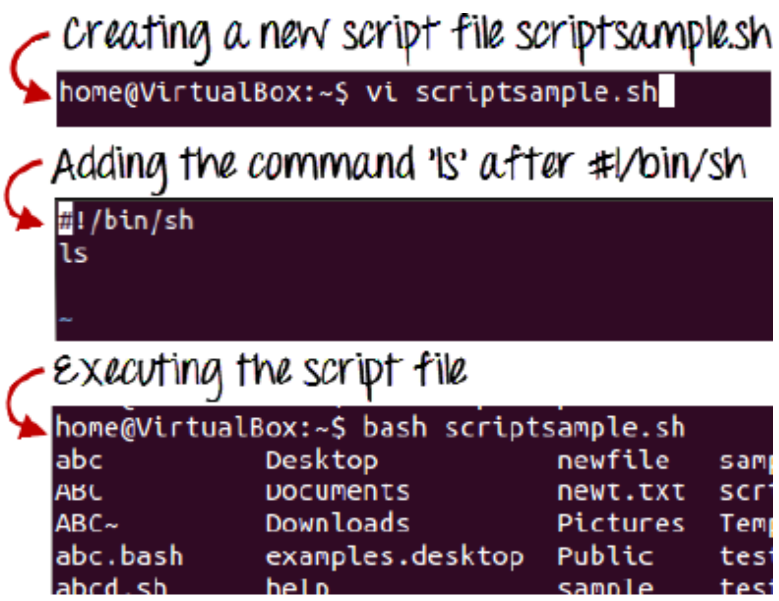
3. Write some code.

4. Save the script file as filename.sh

5. For executing the script type bash filename.sh

"#!" is an operator called shebang which directs the script to the interpreter location. So, if we use"#! /bin/sh" the script gets directed to the bourne-shell. Let's create a small script -

Let's create a small script –


Creating a new script file scriptsample.sh
Adding the command 'ls' after #!/bin/sh
Executing the script file

```
home@VirtualBox:~$ vi scriptsample.sh
```

```
#!/bin/sh
ls

~
```

```
home@VirtualBox:~$ bash scriptsample.sh
abc          Desktop            newfile     samp
ABC          Documents          newt.txt    scr
ABC~         Downloads          Pictures    Temp
abc.bash     examples.desktop   Public      test
abcd.sh      help               sample      test
```

#!/bin/sh
ls

Let's see the steps to create it –

Command 'ls' is executed when we execute the scrip sample.sh file.

Adding shell comments

Commenting is important in any program. In Shell programming, the syntax to add a comment is

#comment
Let understand this with an example.

What are Shell Variables?

As discussed earlier, Variables store data in the form of characters and numbers. Similarly, Shell variables are used to store information and they can by the shell only.

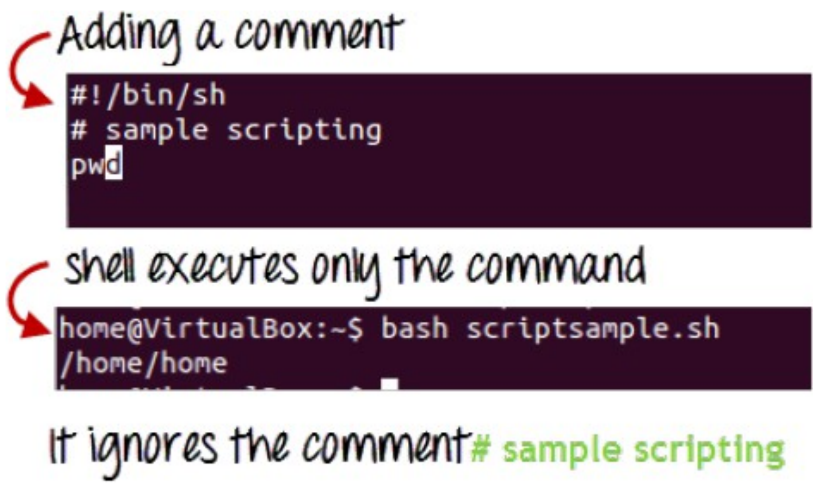
For example, the following creates a shell variable and then prints it:

variable ="Hello"
echo $variable

Below is a small script which will use a variable.

```
#!/bin/sh
echo "what is your name?"
read name
echo "How do you do, $name?"
read remark
echo "I am $remark too!"
```

Let's understand, the steps to create and execute the script


Adding a comment
shell executes only the command
It ignores the comment# sample scripting

```
#!/bin/sh
# sample scripting
pwd
```

```
home@VirtualBox:~$ bash scriptsample.sh
/home/home
```

As you see, the program picked the value of the variable 'name' as Joy and 'remark' as excellent.

This is a simple script. You can develop advanced scripts which contain conditional statements, loops, and functions. Shell scripting will make your life easy and Linux administration a breeze.

Summary:

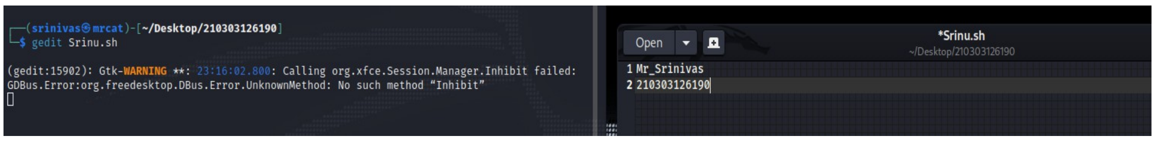
- Kernel is the nucleus of the operating systems, and it communicates between hardware and software
- Shell is a program which interprets user commands through CLI like Terminal
- The Bourne shell and the C shell are the most used shells in Linux
- Shell scripting is writing a series of command for the shell to execute
- Shell variables store the value of a string or a number for the shell to read
- Shell scripting can help you create complex programs containing conditional statements, loops, and functions .

These two commands are useful to create the files.

1. Gedit:

Syntax : gedit prac1.txt

Description: Gedit, the deafault GUI editor if you use Gnome ,also runs under KDE and other desktops . Most gNewsense and linux installations use gnome by default. To start Gedit open a terminal and type.



```
┌──(srinivas㉿mrcat)-[~/Desktop/210303126190]
└─$ gedit Srinu.sh

(gedit:15902): Gtk-WARNING **: 23:16:02.800: Calling org.xfce.Session.Manager.Inhibit failed: GDBus.Error:org.freedesktop.DBus.Error.UnknownMethod: No such method “Inhibit”
▯
```

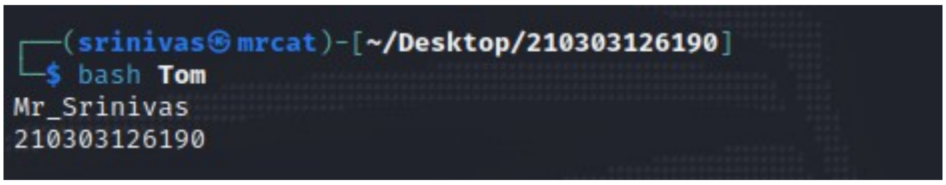
```
Mr_Srinivas
210303126190
```

After Gedit command function a new window will open in that we have to give input. After giving input we have save the file and after that use command bash.

2. Bash:

Syntax : bash prac11.sh

Description : it is used to read the data in existing file in the linux .



```
┌──(srinivas㉿mrcat)-[~/Desktop/210303126190]
└─$ bash Tom
Mr_Srinivas
210303126190
```

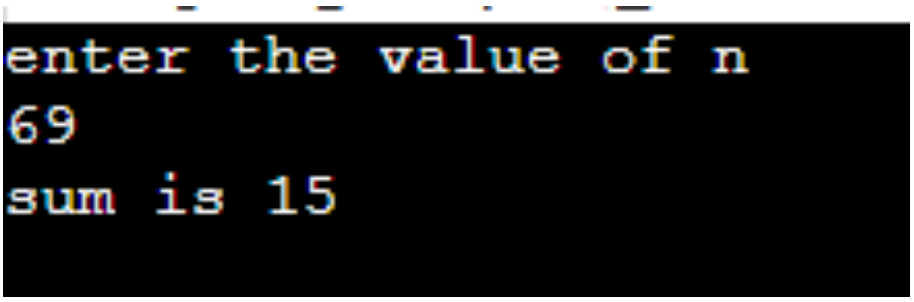
PRACTICAL –3

Aim: Write a Shell script to print given numbers sum of all digits.

Sample Code:

```
echo "enter the value of n";
read n;
sum=0;
while [ $n -gt 0 ]
do
a=`expr $n % 10`
sum=`expr $sum + $a`
n=`expr $n / 10`
done
echo "sum is $sum";
```

Output:



```
enter the value of n
69
sum is 15
```

PRACTICAL – 4

Aim: Write a shell script to validate the entered date.

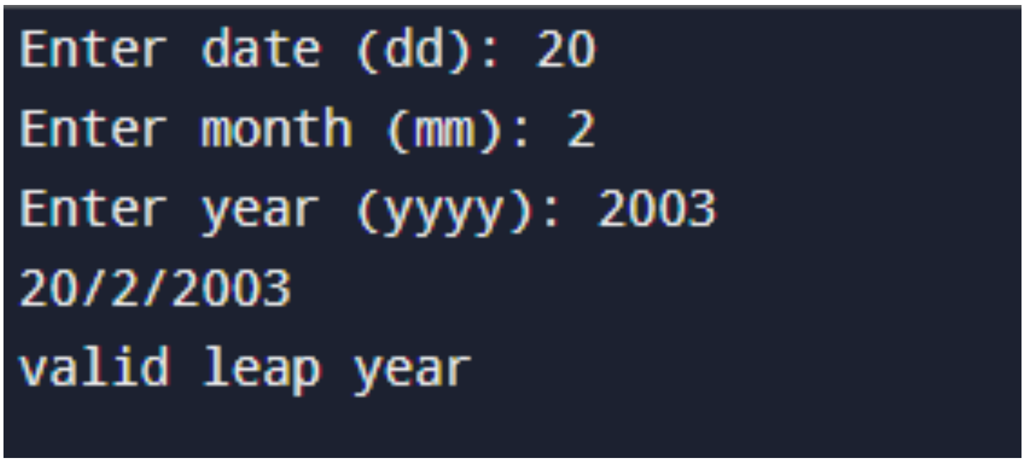
(eg. Date format is: dd-mm-yyyy).

Sample Code:

```
echo "date"
read d
echo "month"
read m
echo "year"
read y
n=`expr $y % 4`
echo $d"/"$m"/"$y
if  [ $m -eq 4 ] || [ $m -eq 6 ] || [ $m -eq 9 ] || [ $m -eq 11 ]
then
if [ $d -gt 0 ] && [ $d -lt 31 ]
then
echo "valid"
else
echo "not valid"
fi
elif  [ $m -eq 2 ]
then
if [ $n -eq 0 ] && [ $d -gt 0 ] && [ $d -lt 30 ]
then
echo "valid"
elif [ $n -gt 0 ] && [ $d -gt 0 ] && [ $d -lt 29 ]
then
echo "not valid"
else
echo "not valid"
fi
else
if [ $d -gt 0 ] && [ $d -lt 32 ]
then
echo "valid"
else
```

```
echo "not valid"
fi
fi
```

Output:



```
Enter date (dd): 20
Enter month (mm): 2
Enter year (yyyy): 2003
20/2/2003
valid leap year
```

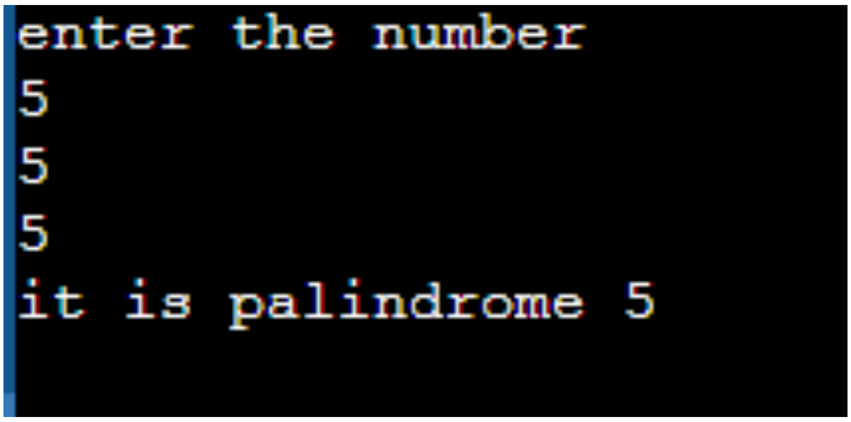
PRACTICAL – 5

Aim: Write a shell script to print whether the number is palindrome or not?

Sample Code:

```
echo " enter the number";
read a;
c=$a;
sum=0;
while [ $a -ne 0 ]
do
        b=`expr $a % 10`;
        echo "$a";
        sum=`expr $sum \* 10`;
        sum=`expr $sum + $b`;
        a=`expr $a \/ 10`;
done
echo "$sum";
if [ $c -eq $sum ]
then {
echo "it is palindrome $sum";
} fi;
```

Output:



```
enter the number
5
5
5
it is palindrome 5
```

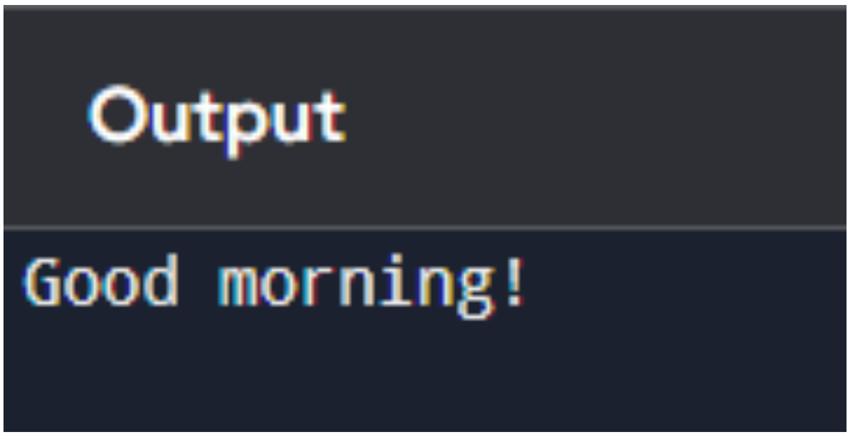
PRACTICAL – 6

Aim: Write a Shell script to say Good morning/Afternoon/Evening as you log in to system.

Sample Code:

```
current_time=$(date +%H)
if [ $current_time -lt 12 ]; then
echo "Good morning!"
elif [ $current_time -lt 17 ]; then
echo "Good afternoon!"
else
echo "Good evening";
fi
```

Output:



Output

```
Good morning!
```

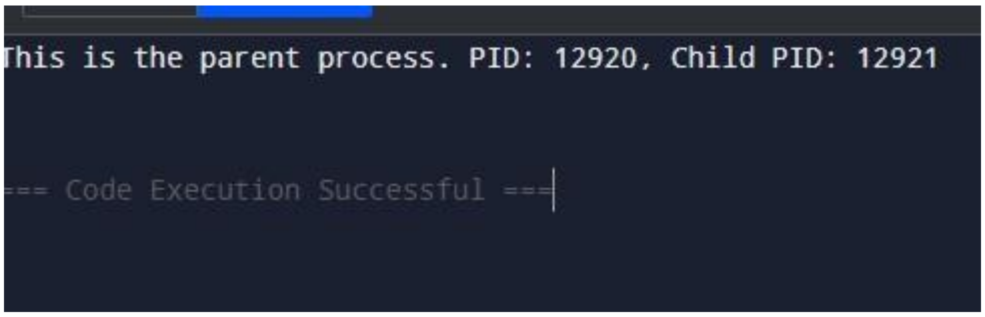
Practical - 7

Aim: Write a C program to create a child process.

SAMPLE CODE:-

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main() {
pid_t pid;
pid = fork();

if (pid < 0) {
perror("Fork failed");
exit(EXIT_FAILURE);
} else if (pid == 0) {
printf("Child Process: PID = %d, Parent PID = %d\n", getpid(), getppid());
} else {
printf("Parent Process: PID = %d, Child PID = %d\n", getpid(), pid);
}

printf("Process %d is exiting.\n", getpid());

return 0;
}
```

Output:



```
This is the parent process. PID: 12920, Child PID: 12921


=== Code Execution Successful ===
```

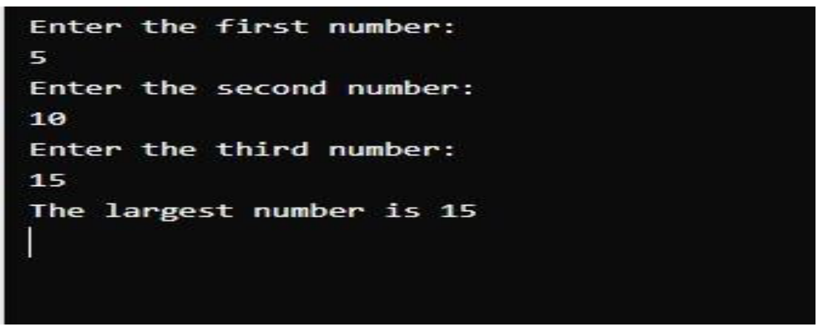
Practical -8

Aim: Finding out biggest number from given three numbers supplied as command line arguments.

SAMPLE CODE:-

```
#!/bin/bash

echo "Enter the first number:"

read num1

echo "Enter the second number:"

read num2

echo "Enter the third number:"

read num3

if [[ $num1 -ge $num2 && $num1 -ge $num3 ]]; then

   echo "The largest number is $num1"

elif [[ $num2 -ge $num1 && $num2 -ge $num3 ]]; then

   echo "The largest number is $num2"

else

   echo "The largest number is $num3"

fi
```

Output:



```
Enter the first number:
5
Enter the second number:
10
Enter the third number:
15
The largest number is 15
```

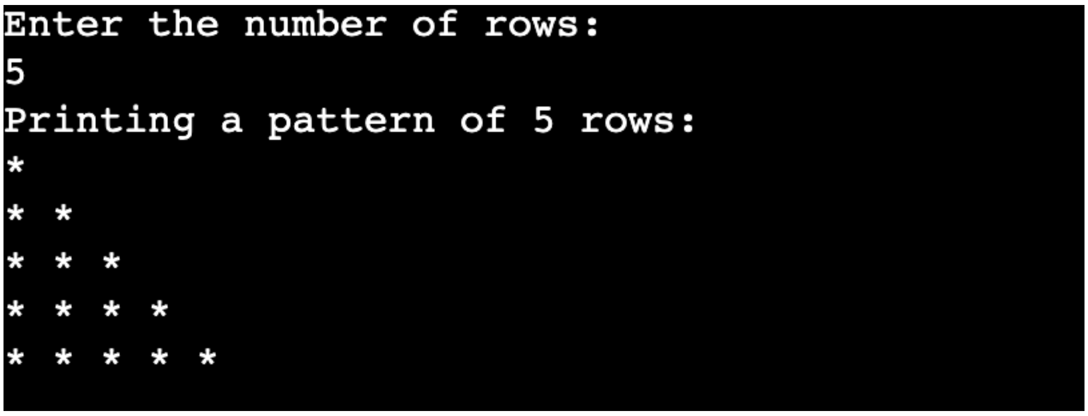
Practical - 9

Aim: Printing the patterns using for loop

SAMPLE CODE:-

```
echo "Printing a pattern of $largest rows:"

for ((i = 1; i <= largest; i++)); do
    for ((j = 1; j <= i; j++)); do
        echo -n "* "
    done
    echo
done
```

Output:



```
Enter the number of rows:
5
Printing a pattern of 5 rows:
*
* *
* * *
* * * *
* * * * *
```

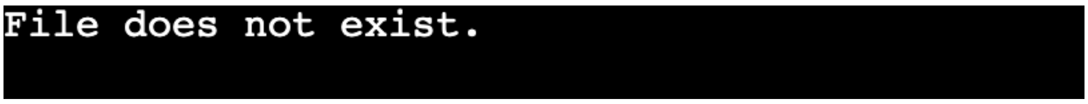
Practical – 10

Aim: Shell script to determine whether given file exist or not.

Sample Code:

```
fileExists() {
   if [ -e "$1" ]; then
      echo "File exists."
   else
      echo "File does not exist."
   fip
}
fileExists "yourfile.txt"
```

Output:



```
File does not exist.
```

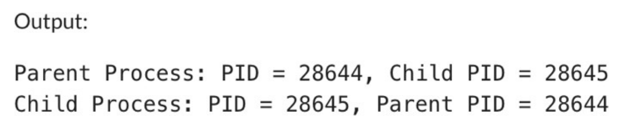
Practical – 11

Aim: Write a program for process creation using C. (use of gcc compiler)

Sample Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int main() {
    pid_t pid;
    pid = fork();

    if (pid < 0) {
        perror("Fork failed");
        return 1;
    } else if (pid == 0) {
        printf("Child Process: PID = %d, Parent PID = %d\n", getpid(), getppid());
    } else {
        printf("Parent Process: PID = %d, Child PID = %d\n", getpid(), pid);
    }

    return 0;
}
```

Output:



Output:

```
Parent Process: PID = 28644, Child PID = 28645
Child Process: PID = 28645, Parent PID = 28644
```

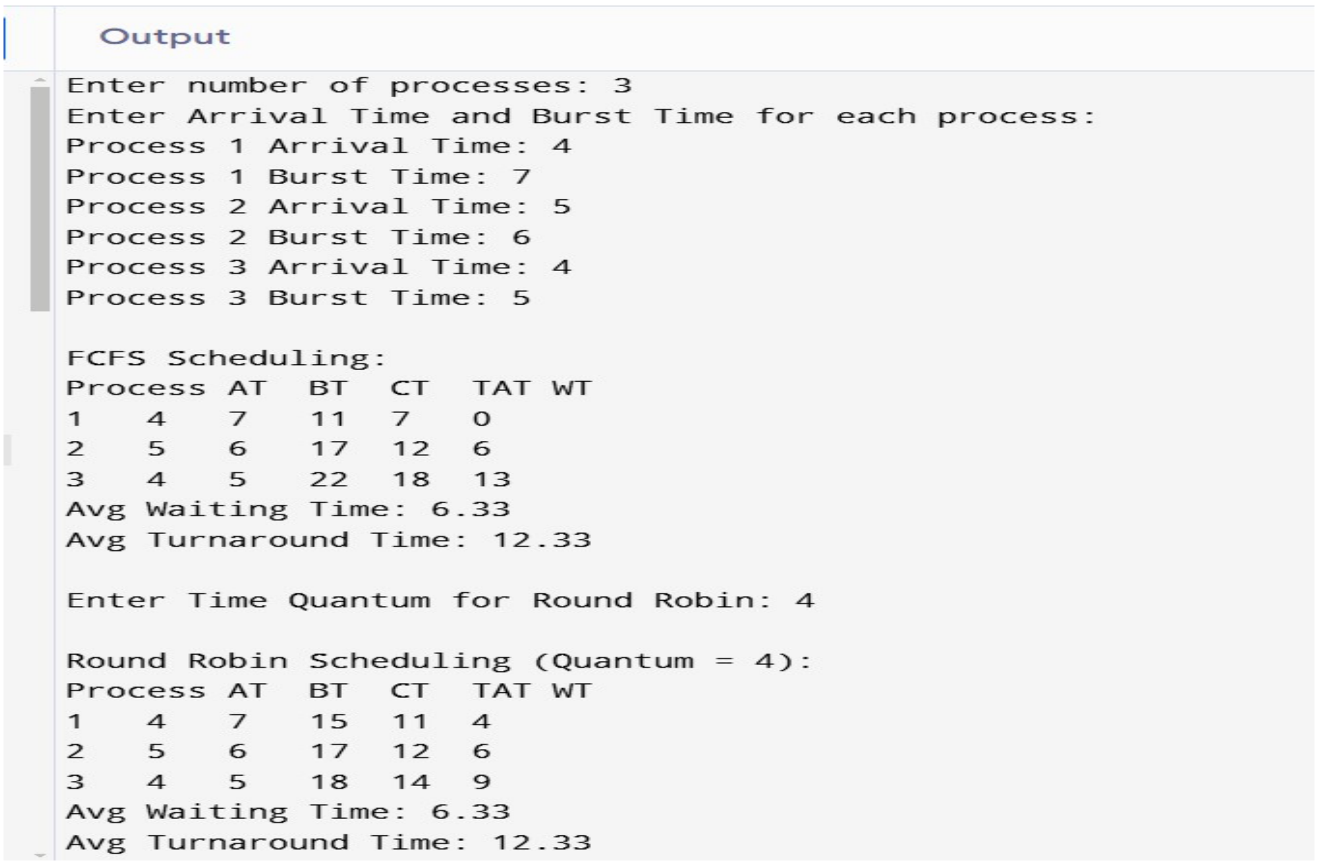
Practical – 12

Aim: Implementation of FCFS &Round Robin Algorithm.

Sample Code:

```
#include <stdio.h>

void fcfs(int n, int at[], int bt[]) {
    int ct[n], tat[n], wt[n], total_wt = 0, total_tat = 0;
    ct[0] = at[0] + bt[0];
    for (int i = 1; i < n; i++)
        ct[i] = (ct[i - 1] > at[i]) ? ct[i - 1] + bt[i] : at[i] + bt[i];
    for (int i = 0; i < n; i++) {
        tat[i] = ct[i] - at[i];
        wt[i] = tat[i] - bt[i];
        total_wt += wt[i];
        total_tat += tat[i];
    }
    printf("\nFCFS Scheduling:\n");
    printf("Process\tAT\tBT\tCT\tTAT\tWT\n");
    for (int i = 0; i < n; i++)
        printf("%d\t%d\t%d\t%d\t%d\t%d\n", i + 1, at[i], bt[i], ct[i], tat[i], wt[i]);
    printf("Avg Waiting Time: %.2f\n", (float)total_wt / n);
    printf("Avg Turnaround Time: %.2f\n", (float)total_tat / n);


void roundRobin(int n, int at[], int bt[], int quantum) {
    int remaining_bt[n], ct[n], tat[n], wt[n], total_wt = 0, total_tat = 0, time = 0, done = 0;
    for (int i = 0; i < n; i++) remaining_bt[i] = bt[i];
    while (done < n) {
        for (int i = 0; i < n; i++) {
            if (remaining_bt[i] > 0) {
                int exec_time = (remaining_bt[i] > quantum) ? quantum : remaining_bt[i];
                time += exec_time;
                remaining_bt[i] -= exec_time;
                if (remaining_bt[i] == 0) {
                    ct[i] = time;
                    tat[i] = ct[i] - at[i];
                    wt[i] = tat[i] - bt[i];
                    total_wt += wt[i];
                    total_tat += tat[i];
                    done++;
                }
            }
        }
    }
    printf("\nRound Robin Scheduling (Quantum = %d):\n", quantum);
    printf("Process\tAT\tBT\tCT\tTAT\tWT\n");
```

```
        for (int i = 0; i < n; i++)
            printf("%d\t%d\t%d\t%d\t%d\t%d\n", i + 1, at[i], bt[i], ct[i], tat[i], wt[i]);
        printf("Avg Waiting Time: %.2f\n", (float)total_wt / n);
        printf("Avg Turnaround Time: %.2f\n", (float)total_tat / n);
    }

    int main() {
        int n, quantum;
        printf("Enter number of processes: ");
        scanf("%d", &n);
        int at[n], bt[n];
        printf("Enter Arrival Time and Burst Time for each process:\n");
        for (int i = 0; i < n; i++) {
            printf("Process %d Arrival Time: ", i + 1);
            scanf("%d", &at[i]);
            printf("Process %d Burst Time: ", i + 1);
            scanf("%d", &bt[i]);
        }
        fcfs(n, at, bt);
        printf("\nEnter Time Quantum for Round Robin: ");
        scanf("%d", &quantum);
        roundRobin(n, at, bt, quantum);
        return 0;
```


Output

```
Enter number of processes: 3
Enter Arrival Time and Burst Time for each process:
Process 1 Arrival Time: 4
Process 1 Burst Time: 7
Process 2 Arrival Time: 5
Process 2 Burst Time: 6
Process 3 Arrival Time: 4
Process 3 Burst Time: 5

FCFS Scheduling:
Process AT  BT  CT  TAT WT
1   4   7   11  7   0
2   5   6   17  12  6
3   4   5   22  18  13
Avg Waiting Time: 6.33
Avg Turnaround Time: 12.33

Enter Time Quantum for Round Robin: 4

Round Robin Scheduling (Quantum = 4):
Process AT  BT  CT  TAT WT
1   4   7   15  11  4
2   5   6   17  12  6
3   4   5   18  14  9
Avg Waiting Time: 6.33
Avg Turnaround Time: 12.33
```

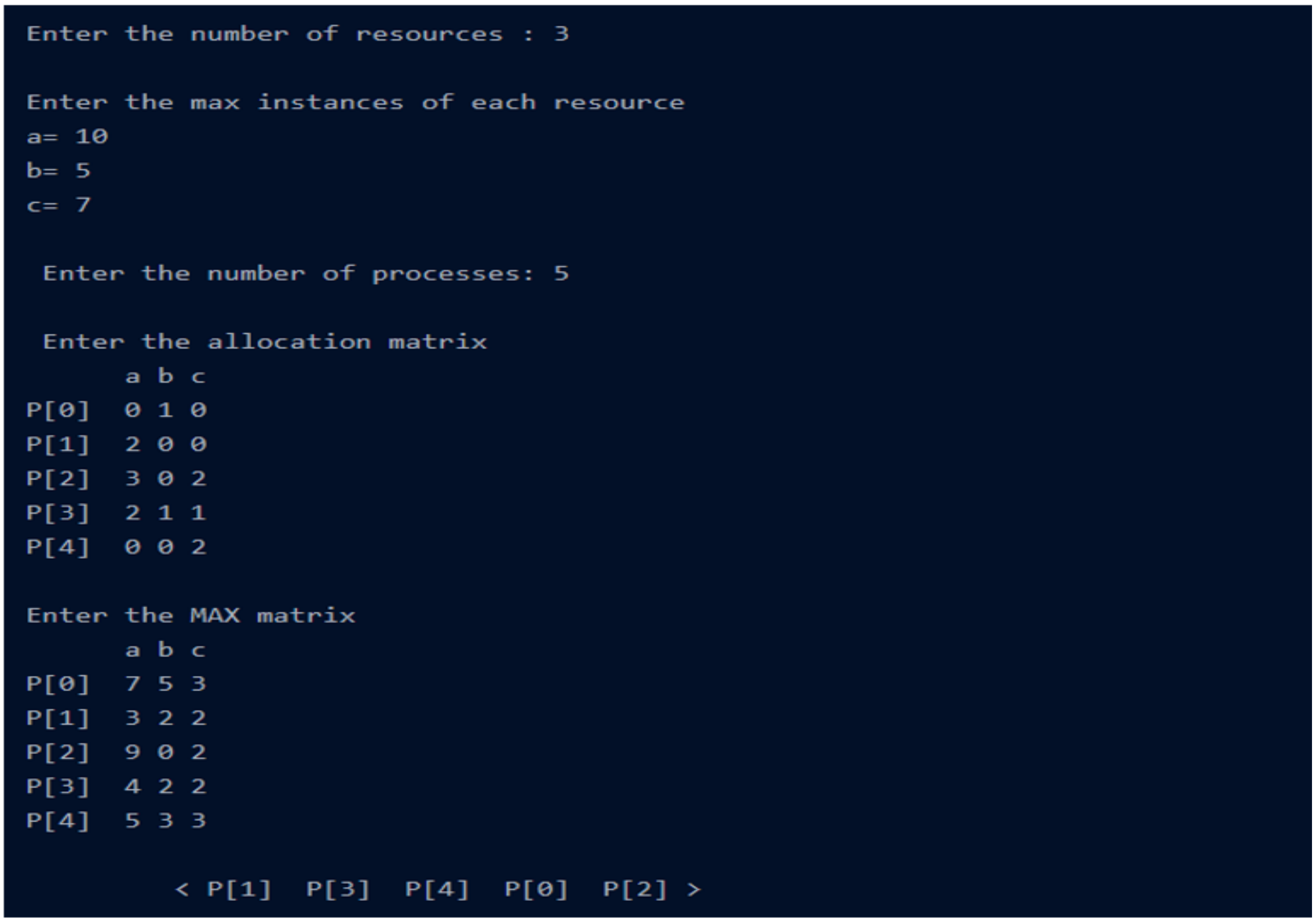
Practical – 13

Aim: Implementation of Banker's Algorithm.

Sample Code:

```
#include <stdio.h>

int isSafe(int n, int m, int alloc[][m], int max[][m], int avail[]) {
    int need[n][m], safeSeq[n], finish[n];
    int work[m];

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            need[i][j] = max[i][j] - alloc[i][j];
        }
    }

    for (int i = 0; i < m; i++) {
        work[i] = avail[i];
    }

    for (int i = 0; i < n; i++) {
        finish[i] = 0;
    }

    int count = 0;
    while (count < n) {
        int found = 0;
        for (int i = 0; i < n; i++) {
            if (!finish[i]) {
                int flag = 1;
                for (int j = 0; j < m; j++) {
                    if (need[i][j] > work[j]) {
                        flag = 0;
                        break;
                    }
                }
                if (flag) {
                    for (int j = 0; j < m; j++) {
                        work[j] += alloc[i][j];
                    }
                    safeSeq[count++] = i;
                    finish[i] = 1;
                    found = 1;
                }
            }
        }
        if (!found) {
            printf("System is in an unsafe state!\n");
            return 0;
        }
    }

    printf("System is in a safe state.\nSafe sequence is: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", safeSeq[i]);
    }
    printf("\n");

    return 1;
}

int main() {
    int n = 5;
    int m = 3;

    int alloc[5][3] = {
        {0, 1, 0},
        {2, 0, 0},
        {3, 0, 2},
        {2, 1, 1},
        {0, 0, 2}
    };

    int max[5][3] = {
        {7, 5, 3},
        {3, 2, 2},
        {9, 0, 2},
        {2, 2, 2},
        {4, 3, 3}
    };

    int avail[3] = {3, 3, 2};

    isSafe(n, m, alloc, max, avail);

    return 0;
}
```

OUTPUT:



```
Enter the number of resources : 3

Enter the max instances of each resource
a= 10
b= 5
c= 7

 Enter the number of processes: 5

 Enter the allocation matrix
      a b c
P[0]  0 1 0
P[1]  2 0 0
P[2]  3 0 2
P[3]  2 1 1
P[4]  0 0 2

Enter the MAX matrix
      a b c
P[0]  7 5 3
P[1]  3 2 2
P[2]  9 0 2
P[3]  4 2 2
P[4]  5 3 3

         < P[1]  P[3]  P[4]  P[0]  P[2] >
```