**CERTIFICATE**

*This is to certify that Mr./Ms.* ***….... Hemil…Chovatiya.............*** *with enrolment no.* ***..........200303108003.................*** *has successfully completed* ***his****/her laboratory experiments in the* ***…..Operating System Laboratory (******203105203)……*** *from the department of* ***........Information Technology(4ITA1)….….........*** *during the academic year* ***........2021-2022.........***



Date of Submission: ......................... Staff In charge: ...........................

Head of Department: ...........................................

# **INDEX**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Sr. No | Experiment Title | Page No | | Date of  Performance | Date of  Assessment | Marks  (out of  10) | Sign |
| From | To |
| 1 | Study of Basic commands of Linux. |  |  |  |  |  |  |
| 2 | Study the basics of shell programming. |  |  |  |  |  |  |
| 3 | Write a Shell script to print given numbers sum of all digits |  |  |  |  |  |  |
| 4 | Write a shell script to validate the entered date. (e.g. Date format is: dd-mm-yyyy) |  |  |  |  |  |  |
| 5 | **A.** Write a shell script to check entered string is palindrome or not. |  |  |  |  |  |  |
|  | **B.** Write a Shell script to say Good morning/Afternoon/ Evening as you log in to system. |  |  |  |  |  |  |
| 6 | Write a C program to create a child process.(use of gcc compiler). |  |  |  |  |  |  |
| 7 | 1. Finding out biggest number from given three numbers supplied as command line Arguments. |  |  |  |  |  |  |
| **B.** Printing the patterns using for loop. |
| 8 | Shell script to determine whether given file exist or not. |  |  |  |  |  |  |
| 9 | Implementation of FCFS & Round Robin Algorithm. |  |  |  |  |  |  |
| 10 | Implementation of Banker Algorithm. |  |  |  |  |  |  |

**PRACTICAL 1**

**AIM: Study basics command of Linux.**

**Linux Basic Commands:**

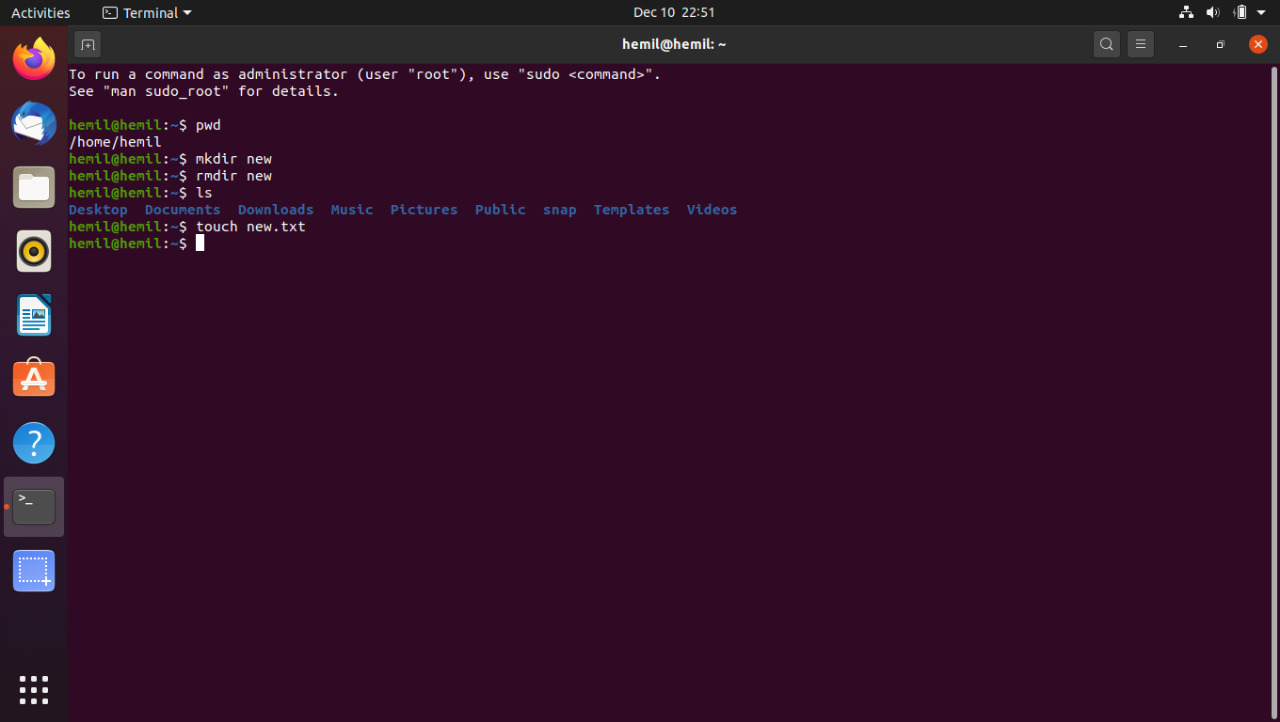
**1. pwd Command:** The pwd command is used to display the location of the current working directory.

**2. mkdir Command:** The mkdir command is used to create a new directory under any directory.

**3. rmdir Command:** The rmdir command is used to delete a directory.

**4. ls Command:** The ls command is used to display a list of content of a directory.

**5. touch Command:** The touch command is used to create empty files. We can create multiple empty files by executing it once.



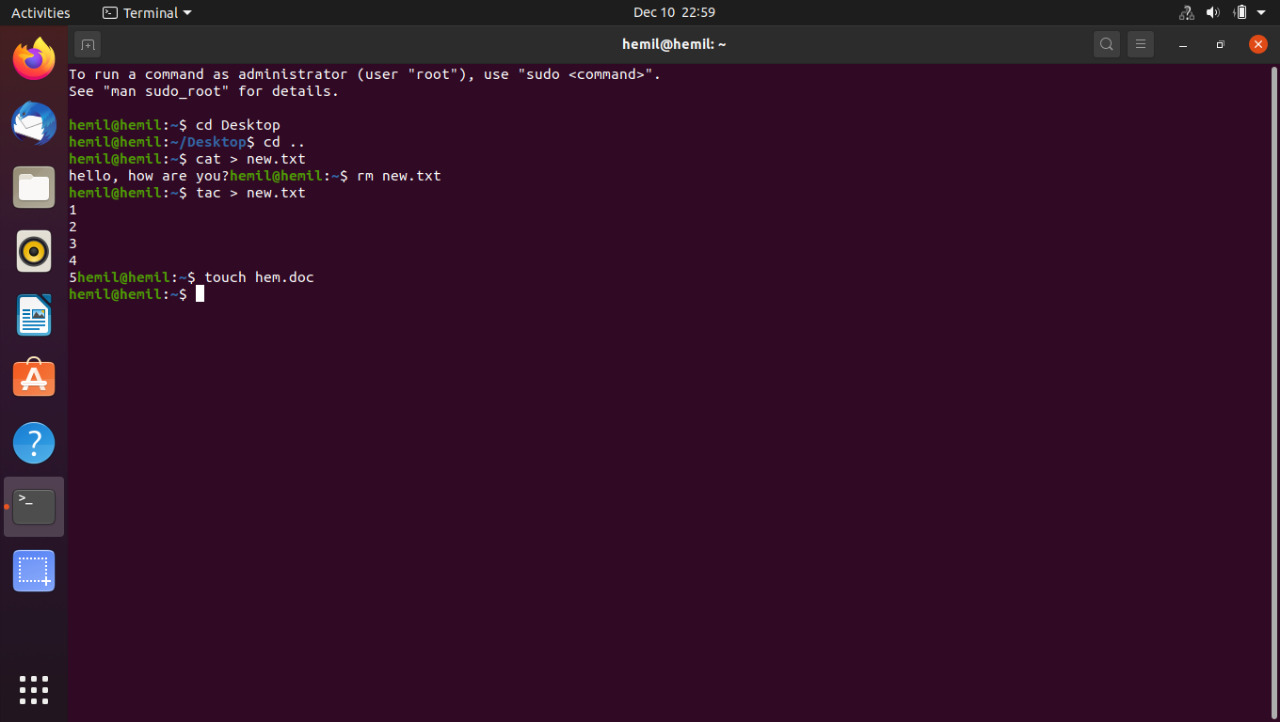
**6. cd Command:** The cd command is used to change the current directory.

**7. cat Command:** The cat command is a multi-purpose utility in the Linux system. It can be used to create a file, display content of the file, copy the content of one file to another file, and more.

**8. rm Command**: The rm command is used to remove a file.

**9. tac Command**: The tac command is the reverse of cat command, as its name specified. It displays the file content in reverse order (from the last line).

**10. cd .. command:** This command is used to go Back to previous main folder.



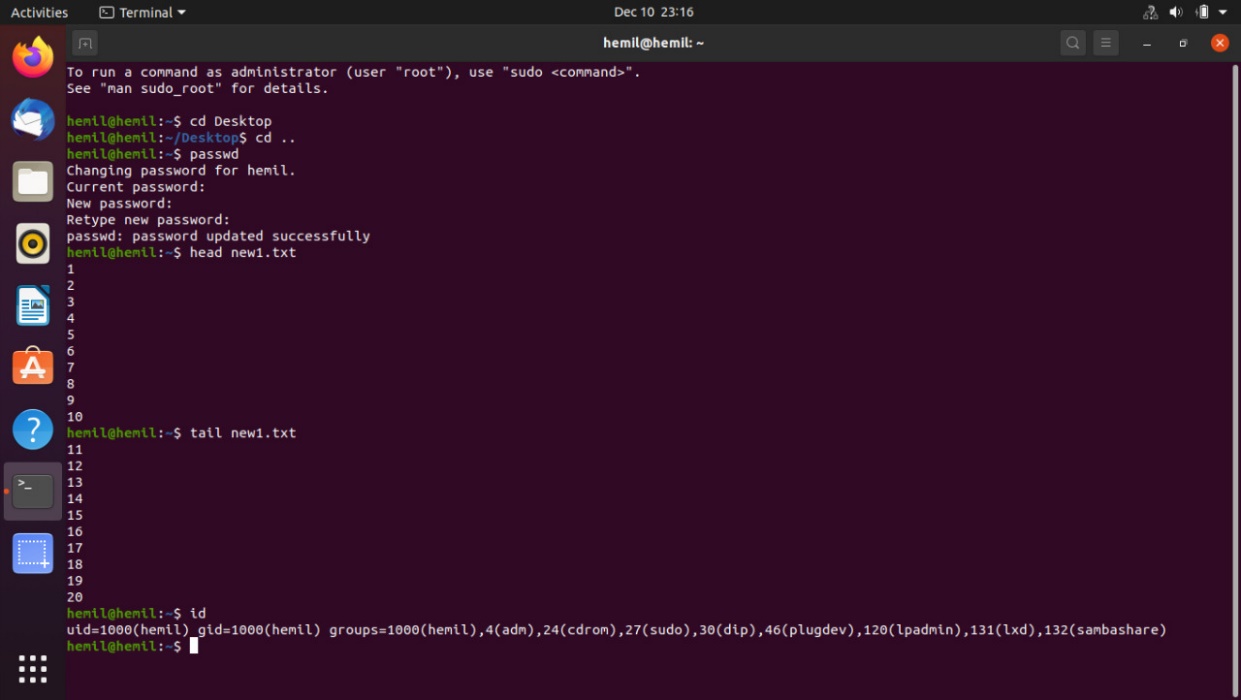
**11. head Command**: The head command is used to display the content of a file. It displays the first 10 lines of a file.

**12. tail Command**: The tail command is similar to the head command. The difference between both commands is that it displays the last ten lines of the file content. It is useful for reading the error message.

**13. passwd Command**: The passwd command is used to create and change the password for a user.

**14. id Command**: The id command is used to display the user ID (UID) and group ID (GID).

**15. su Command**: The su command provides administrative access to another user. In other words, it allows access of the Linux shell to another user.



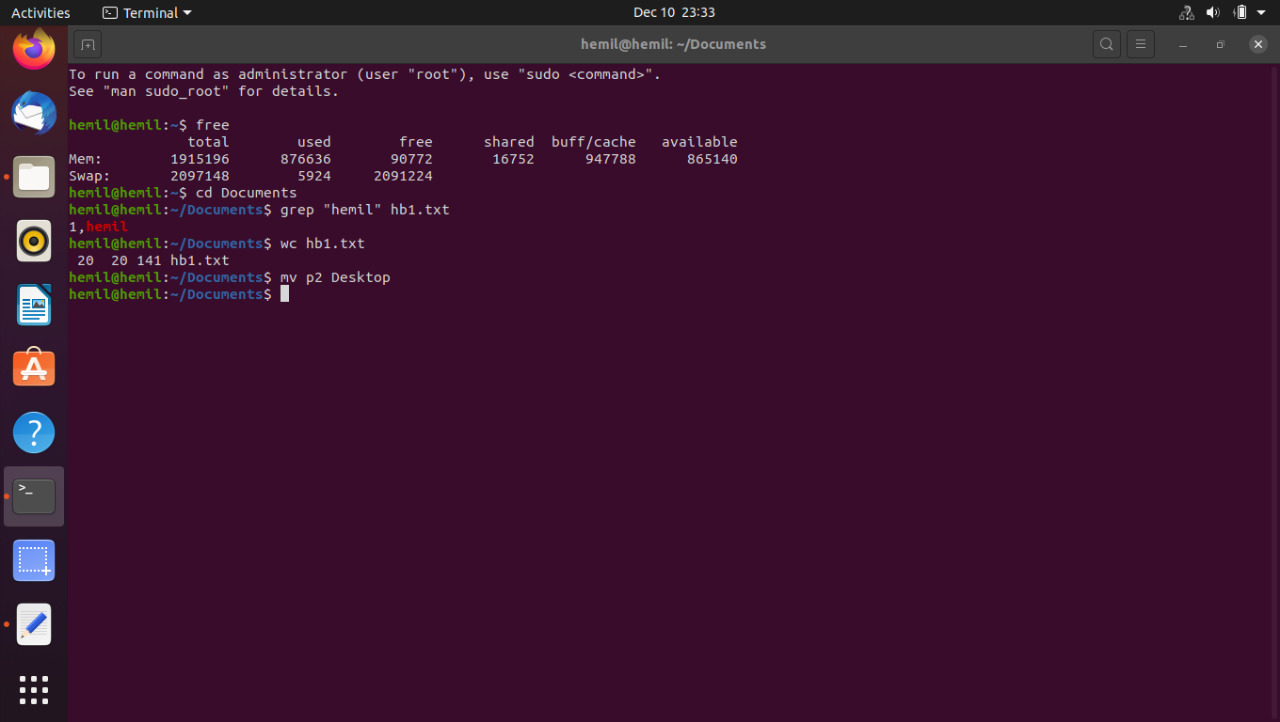
**16.** **Free command:** It gives information about used and unused memory usage and swap memory of a system

**17. grep command:** to perform text searches for a defined criteria of words or strings

**18.wc command:** It is used to find out number of lines, word count, byte and characters count in the files specified in the file arguments.

**19.mv command:** mv is used to move one or more files or directories from one place to another in a file system.

**20.** **PING command:**it is used to check the network connectivity between host and server/host



**PRACTICAL 2**

**AIM: Study the basics of shell programming.**

**THEORY:**

**1. What is the shell script?**

A Shell provides you with an interface to the Unix system. It gathers input from you and executes programs based on that input. When a program finish executing, it displays that program's output.

Shell is an environment in which we can run our commands, programs, and shell scripts. There are different flavours of a shell, just as there are different flavours of operating systems. Each flavour of shell has its own set of recognized commands and functions.

**2. Type of shell script.**

In Unix, there are two major types of shells −

• Bourne shell − If you are using a Bourne-type shell, the $ character is the default prompt.

• C shell − If you are using a C-type shell, the % character is the default prompt.

The Bourne Shell has the following subcategories −

• Bourne shell (sh)

• Korn shell (ksh)

• Bourne Again shell (bash)

• POSIX shell (sh)

The different C-type shells follow –

• C shell (csh)

• TENEX/TOPS C shell (tcsh)

**3. Creating shell files.**

1. Start the script with **#! /bin/sh**

2. Write some code.

3. Save the script file as **filename.sh**

4. For executing the script type bash filename.sh

**Command: #/bin/sh**

**4. touch command**: It is used to create a file without any content. The file created using touch command is empty. This command can be used when the user doesn’t have data to store at the time of file creation.

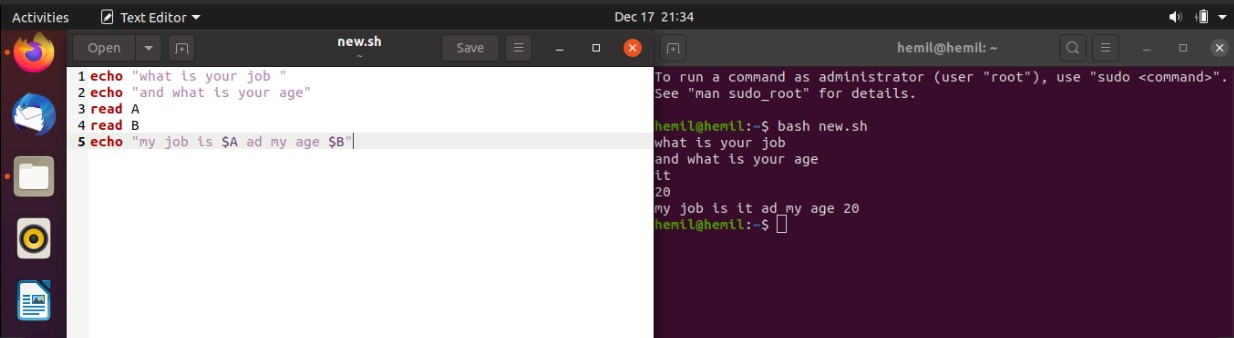
Syntax: touch\_ filename.txt

**5. cat command:** It is used to create the file with content.

Syntax: cat > filename.txt

**6. Text editor:** Bash will execute as a different process. This way, changes that occur while the file is being executed cannot affect your shell.

Ex: bash \_ filename.sh

****

**PRACTICAL 3**

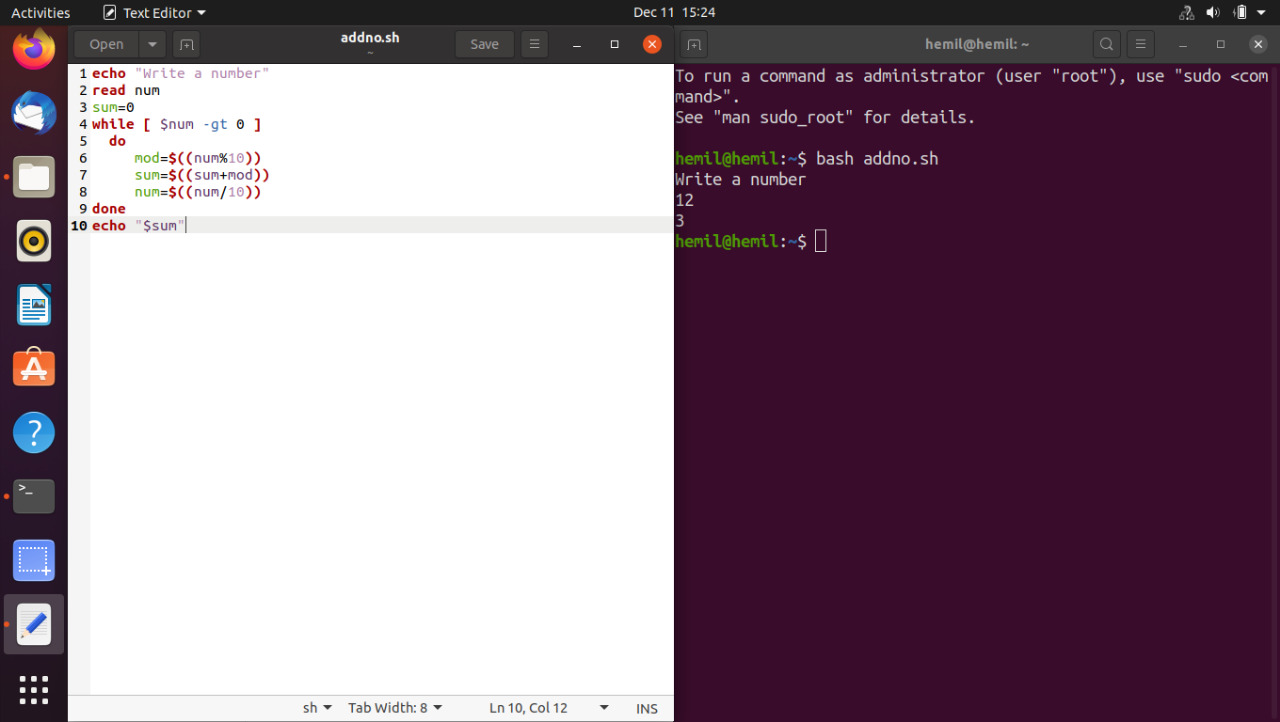
**AIM: Write a Shell script to print given numbers sum of all digits**

**Algorithm:**

**General Algorithm for sum of digits in a given number:**

1. Get the number as input
2. Declare a variable to store the sum and set it to 0
3. Repeat the next two steps till the number is not 0
4. Get the rightmost digit of the number with help of the remainder ‘%’ operator by dividing it by 10 and add it to sum.
5. Divide the number by 10 with help of ‘/’ operator to remove the rightmost digit.
6. Print or return the sum

**Code : Output:**

****

**Flowchart:**

Start

sum=0

print sum

num!=0

mod=num%10

read num

End

num=num/10

sum=sum+mod

False

True

**PRACTICAL 4**

**AIM: Write a shell script to validate the entered date. (e.g., Date format is: dd-mm-yyyy)**

**Algorithm:**

* Enter date in DD/MM/YYYY Format.
* Check year validation, if year is not valid print error.
* If year is valid, check month validation (i.e., month is between 1 to 12), if month is not valid print error.
* If month is valid, then finally check day validation with leap year condition, here we will day range from 1 to 30, 1 to 31, 1 to 28 and 1 to 29.
* If day is valid print date is correct otherwise print error.

**Code:**

dd=0

mm=0

yy=0

days=0

echo -n "Enter day(dd):"

read dd

echo -n "Enter month(mm):"

read mm

echo -n "Enter year(yyyy):"

read yy

if [ $mm -le 0 -o $mm -gt 12 ];

then

echo "$mm is invalid month."

exit 1

fi

case $mm in

1) days=31;;

2) days=28;;

3) days=31;;

4) days=30;;

5) days=31;;

6) days=30;;

7) days=31;;

8) days=31;;

9) days=30;;

10) days=31;;

11) days=30;;

12) days=31;;

\*) days=-1;;

esac

if [ $mm -eq 2 ];

then

if [ $((yy % 4)) -ne 0 ] ; then

:

elif [ $(yy % 400) -eq 0 ] ; then

days=29

elif [ $((yy % 100)) -eq 0 ] ; then

:

else

days=29

fi

fi

if [ $dd -le 0 -o $dd -gt $days ];

then

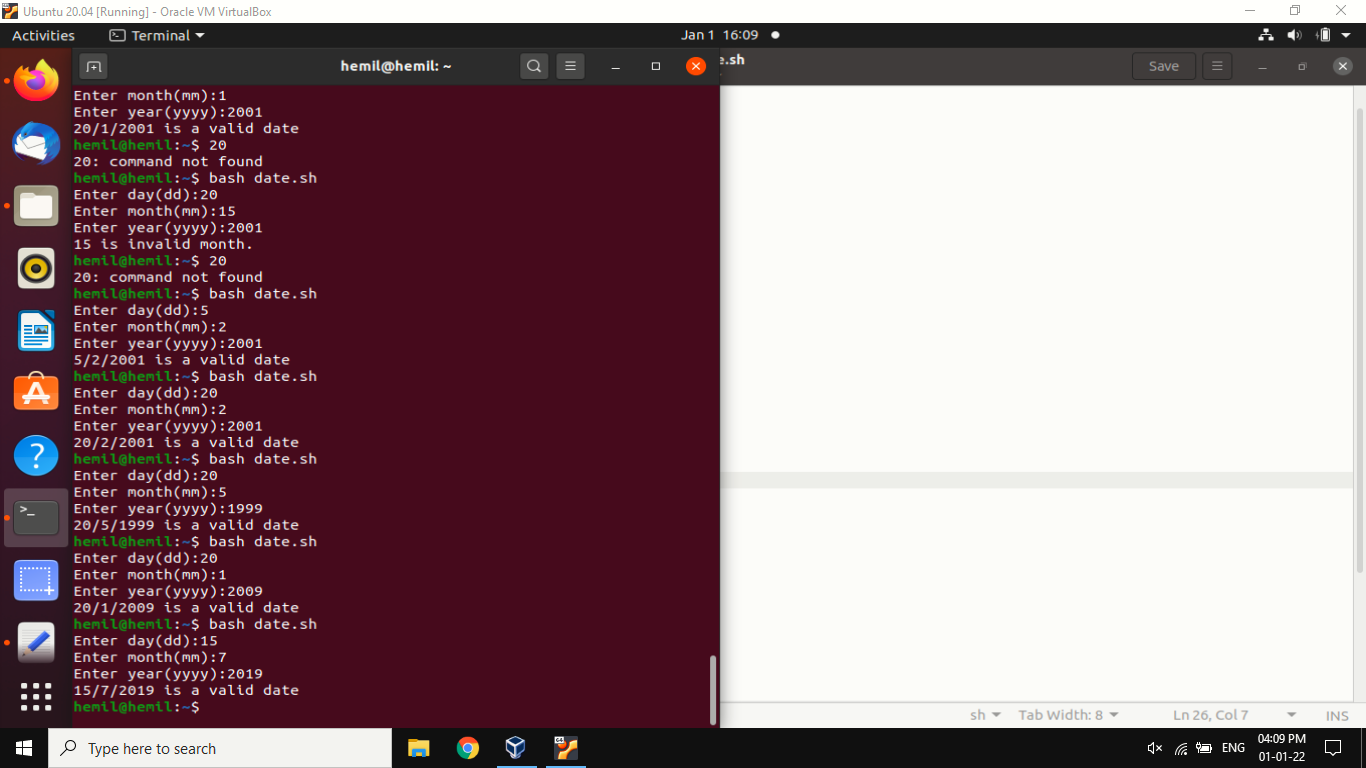
echo "$dd day is invalid"

exit 3

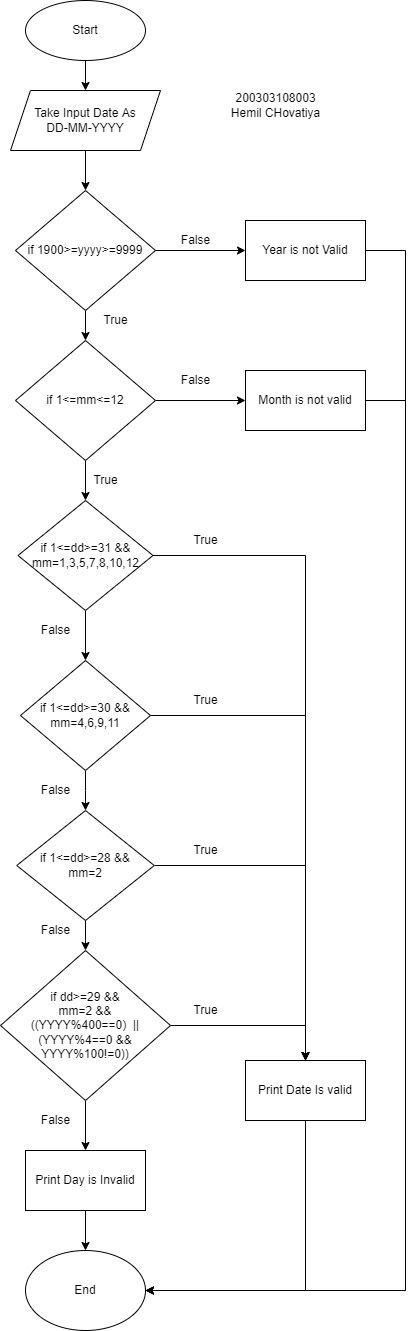
fi

echo "$dd/$mm/$yy is a valid date"

**Output:**



**Flowchart:**



**PRACTICAL 5**

**AIM: Write a shell script to check entered string is palindrome or not**

**Algorithm:**

* Input a String
* Initialize Len to zero , Flag to zero
* While String[Len] is not equal to NULL
* Increment Len
* Initialize I to zero , J to Len-1
* If va1 equal to rev
* Print Key Is a Palindrome
* else
* Print Key Is Not a Palindrome
* Stop

**Detailed Algorithm:**

Step 1:  Input S (string)

Step 2:  Len = 0 , Flag =0

Step 3:  While (S[Len] != NULL)

                        Len++

Step 4:  I = 0 , J = Len-1

Step 5:  While ( I < (Len/2)+1 )

                        If ( S[I] == S[J] )

                                 Flag=0

                        else

                                  Flag=1

                        I++ , J–

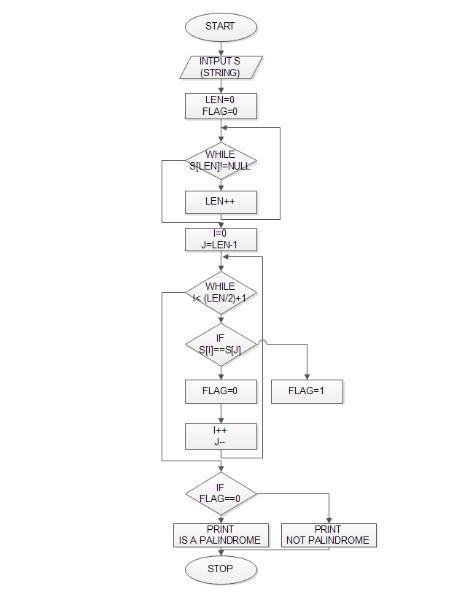
Step 6:      If ( Flag == 0 )

                               Print Key Is a Palindrome

                     else

                                Print Key Is Not a Palindrome

Step 7: End

**Flow Chart:**

**Code:**

echo Enter the value of string

read s

echo $s>temp

rvs="$(rev temp)"

if [ $s = $rvs ]

then

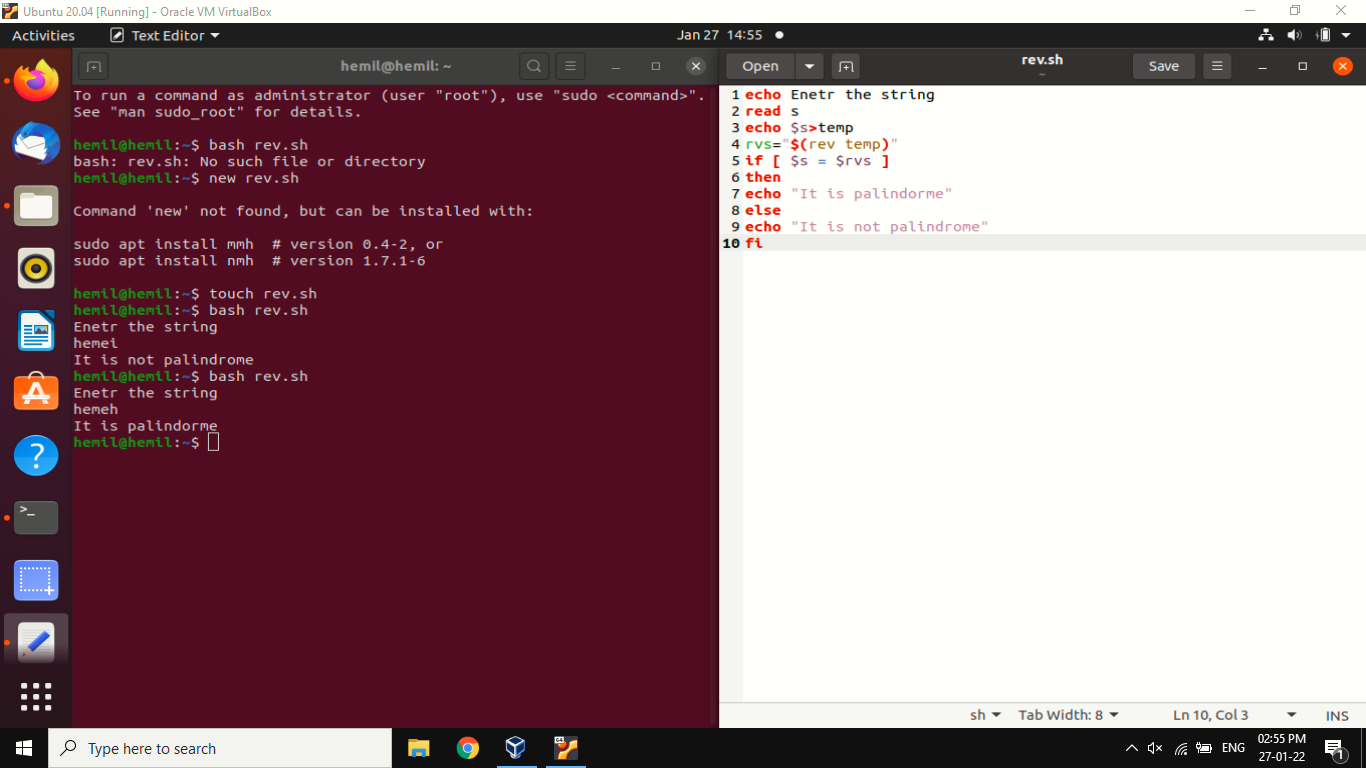
echo "it is palindrome"

else

echo " it is not a Palindrome"

fi

**Output:**

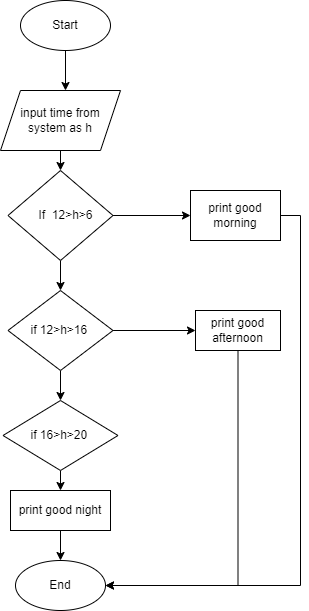


**PRACTICAL 5 B**

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

**Algorithm:**

1. Take input date from system in hour as h
2. If 12>h>6 then print good morning
3. Else if 12>h>16 then print good afternoon
4. Else if 16>h>20 then print good evening
5. Else print good night

**Flow Chart:**

**Code:**

**h**=$(date +"%H")

**if** [ **$h** -**gt** 6 -a **$h** -**le** 12 ]

**then**

echo good morning

**elif** [ **$h** -**gt** 12 -a **$h** -**le** 16 ]

**then**

echo good afternoon

**elif** [ **$h** -**gt** 16 -a **$h** -**le** 20 ]

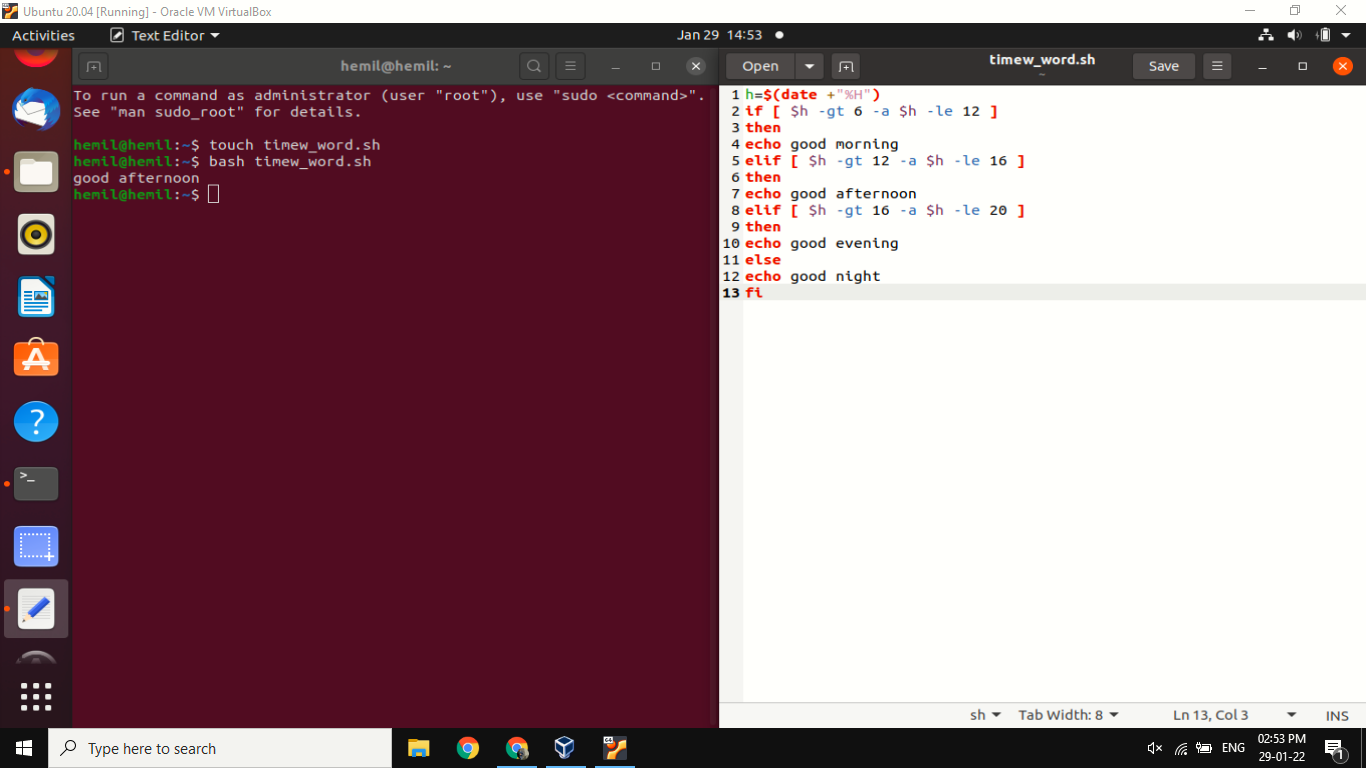
**then**

echo good evening

**else**

echo good night

**fi**

**Output:**

**PRACTICAL 6**

**AIM:** **Write a C program to create a child process. (Use of gcc compiler).**

**Code:**

#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

int main()

{

// make two process which run same

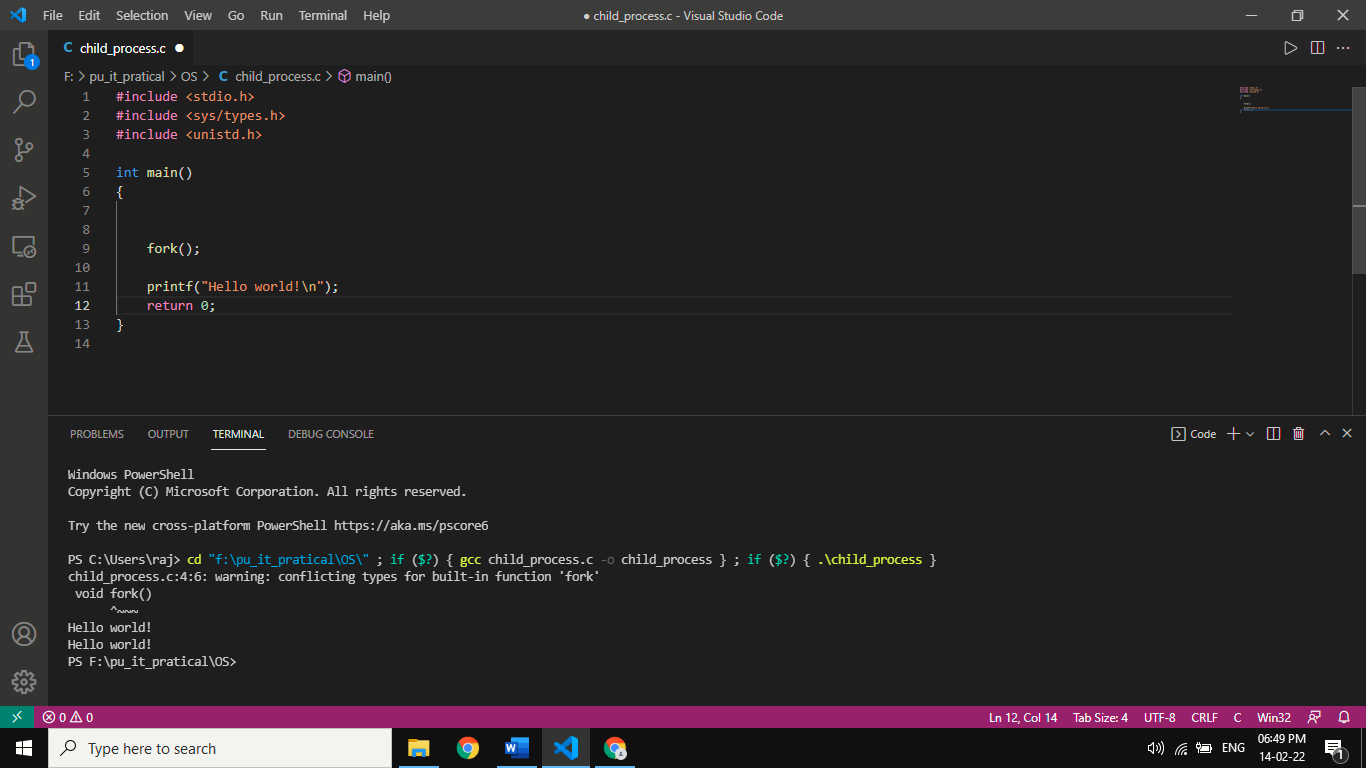
// program after this instruction

fork();

printf("Hello world!\n");

return 0;

}



**PRACTICAL 7 A**

**AIM: Finding out biggest number from given three numbers supplied as command line Arguments**

**Code:**

read a b c

if [[ $a == 0 || $b == 0 || $c == 0 ]]; then

echo "command line arguments are missing"

elif [[ $a == $b && $b == $c ]]; then

echo "All the three numbers are equal"

elif [[ $a == $b && $b > $c || $b == $c && $c > $a || $a == $c && $a > $b ]]; then

echo "I cannot figure out which number is biggest"

else

if [[ $a > $b && $a > $c ]]; then

echo "$a is Biggest number"

elif [[ $b > $a && $b > $c ]]; then

echo "$b is Biggest number"

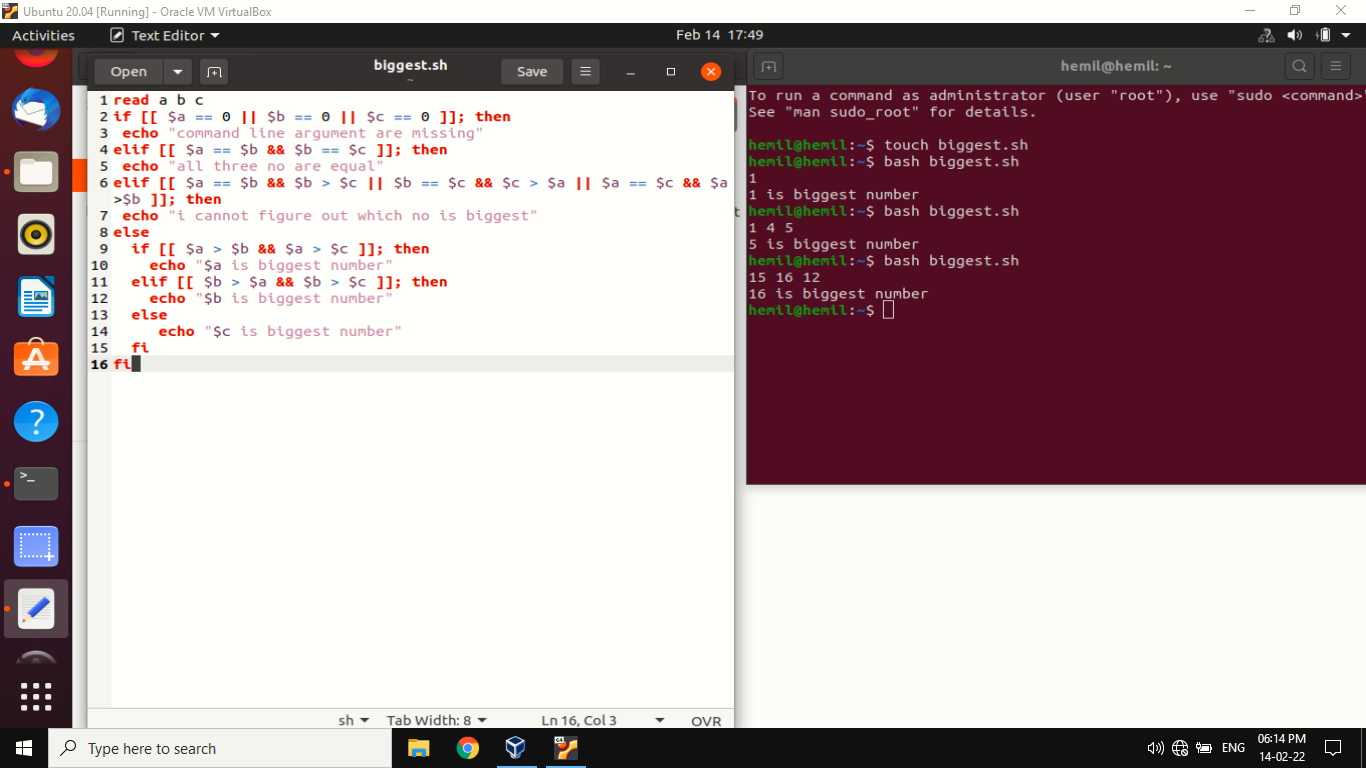
else

echo "$c is Biggest number"

fi

fi

**Output:**



**PRACTICAL 7 B**

**AIM: Printing the patterns using for loop.**

**Code:**

number=1

rows=5

for((i=rows; i>=1; i--))

do

for((j=1; j<=i; j++))

do

echo -n "$number "

number=$((number + 1))

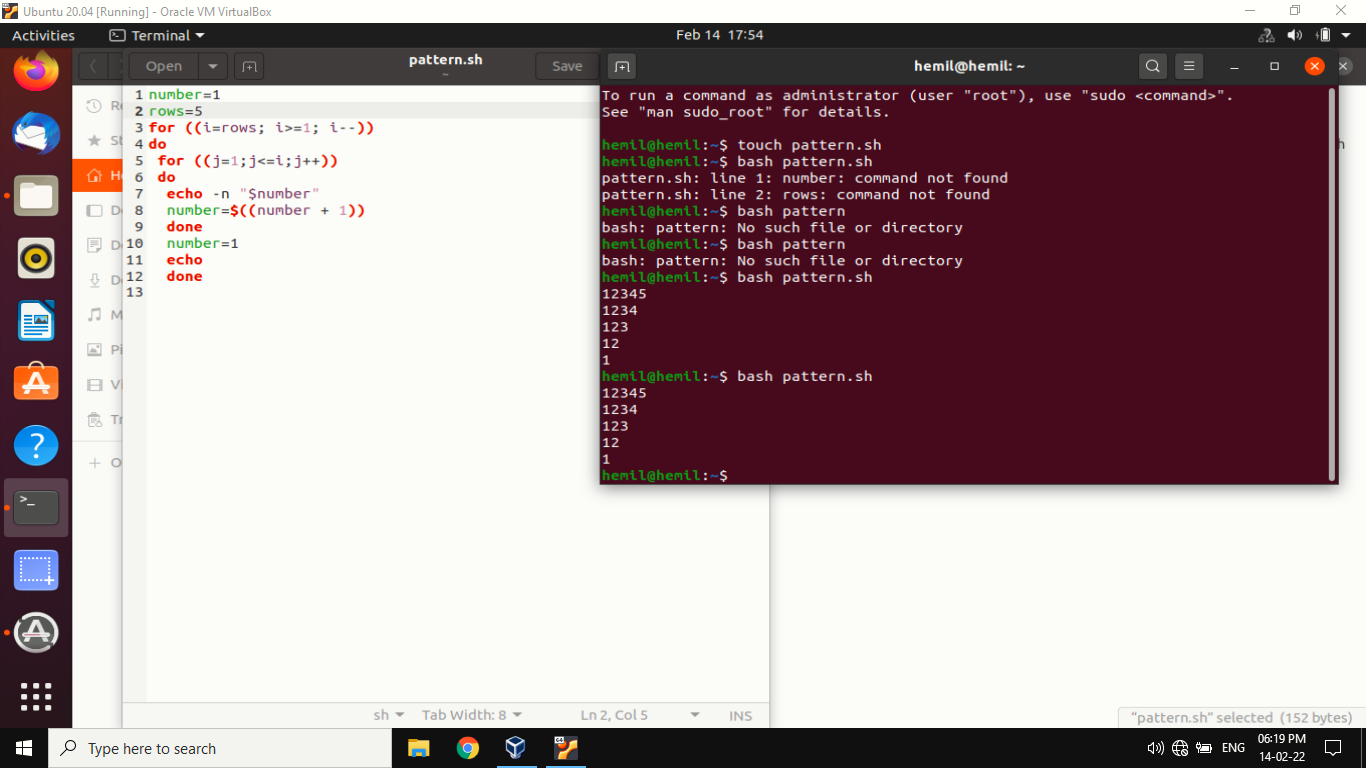
done

number=1

echo

done

**Output:**



**PRACTICAL 8**

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

**Code:**

if [ -e name.sh ]

then

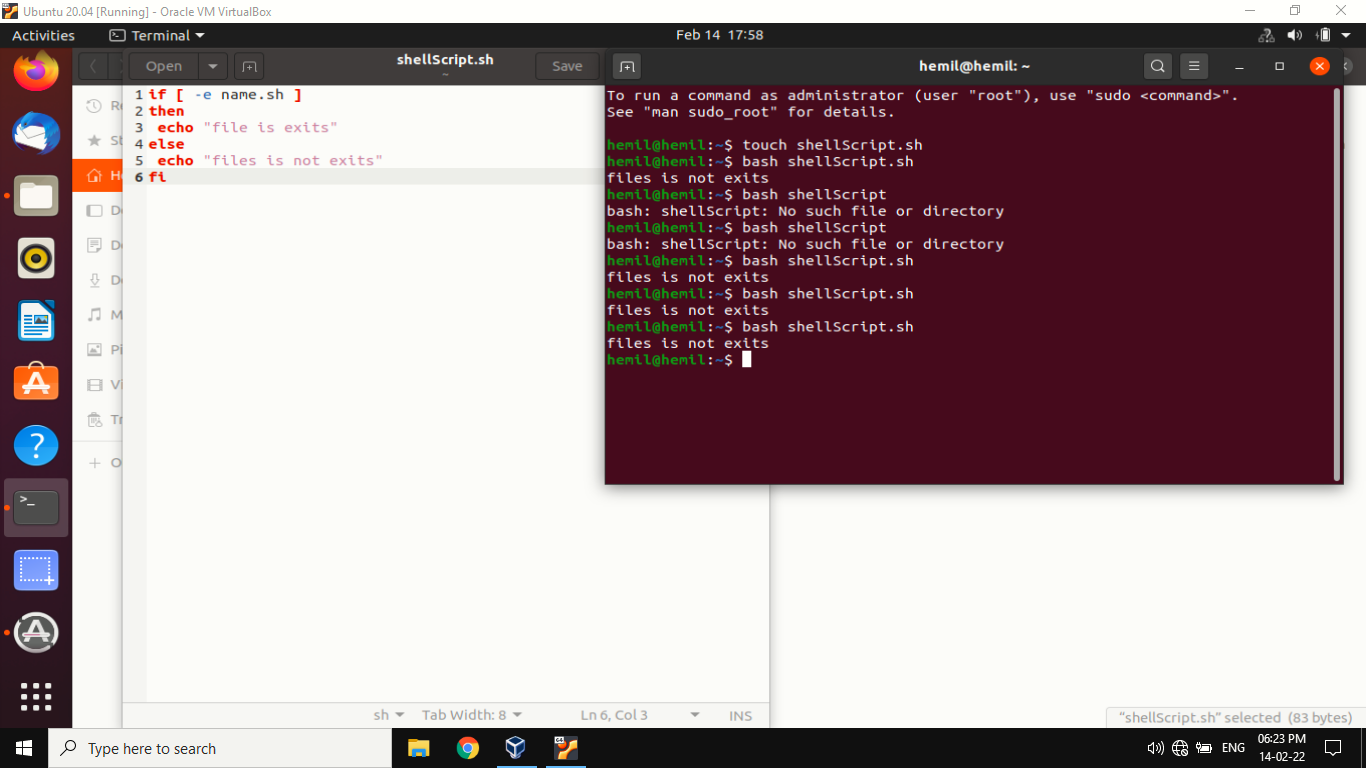
echo "file is exists "

else

echo "file is not exists"

fi

**Output:**



**PRACTICAL 9**

**AIM: Implementation of FCFS & Round Robin Algorithm.**

**Code:**

#include<stdio.h>

void findWaitingTime(int processes[], int n,int bt[], int wt[])

{

wt[0] = 0;

for (int i = 1; i < n ; i++ )

wt[i] = bt[i-1] + wt[i-1] ;

}

void findTurnAroundTime( int processes[], int n,int bt[], int wt[], int tat[])

{

for (int i = 0; i < n ; i++)

tat[i] = bt[i] + wt[i];

}

void findavgTime( int processes[], int n, int bt[])

{

int wt[n], tat[n], total\_wt = 0, total\_tat = 0;

findWaitingTime(processes, n, bt, wt);

findTurnAroundTime(processes, n, bt, wt, tat);

printf("Processes Burst time Waiting time Turn around time\n");

for (int i=0; i<n; i++)

{

total\_wt = total\_wt + wt[i];

total\_tat = total\_tat + tat[i];

printf(" %d ",(i+1));

printf(" %d ", bt[i] );

printf(" %d",wt[i] );

printf(" %d\n",tat[i] );

}

int s=(float)total\_wt / (float)n;

int t=(float)total\_tat / (float)n;

printf("Average waiting time = %d",s);

printf("\n");

printf("Average turn around time = %d ",t);

}

int main()

{

int processes[] = { 1, 2, 3};

int n = sizeof processes / sizeof processes[0];

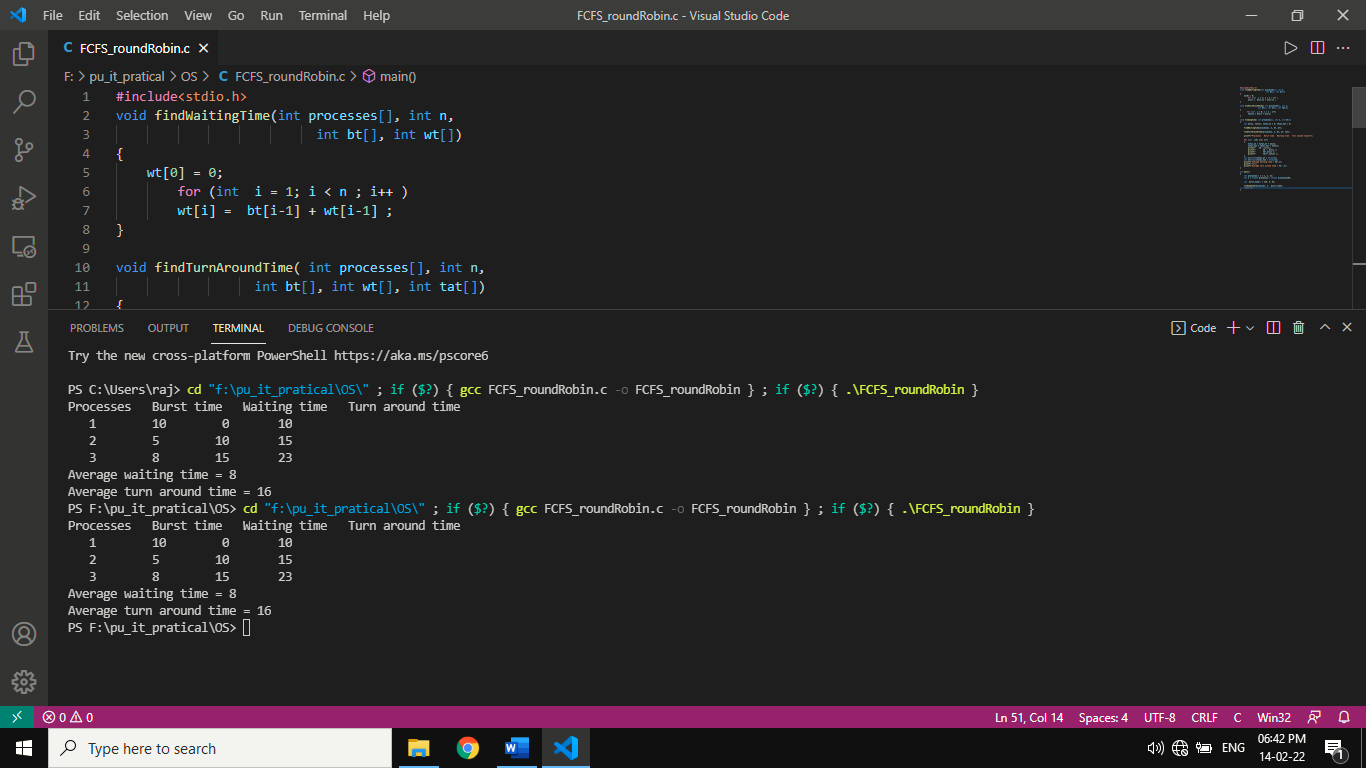
int burst\_time[] = {10, 5, 8};

findavgTime(processes, n, burst\_time);

return 0;

}

**Output:**



**PRACTICAL 10**

**AIM: Implementation of Banker Algorithm**

**Code:**

#include <stdio.h>

int main()

{

int n, m, i, j, k;

n = 5;

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 };

int f[n], ans[n], ind = 0;

for (k = 0; k < n; k++) {

f[k] = 0;

}

int need[n][m];

for (i = 0; i < n; i++) {

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

need[i][j] = max[i][j] - alloc[i][j];

}

int y = 0;

for (k = 0; k < 5; k++) {

for (i = 0; i < n; i++) {

if (f[i] == 0) {

int flag = 0;

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

if (need[i][j] > avail[j]){

flag = 1;

break;

}

}

if (flag == 0) {

ans[ind++] = i;

for (y = 0; y < m; y++)

avail[y] += alloc[i][y];

f[i] = 1;

}

}

}

}

int flag = 1;

for(int i=0;i<n;i++)

{

if(f[i]==0)

{

flag=0;

printf("The following system is not safe");

break;

}

}

if(flag==1)

{

printf("Following is the SAFE Sequence\n");

for (i = 0; i < n - 1; i++)

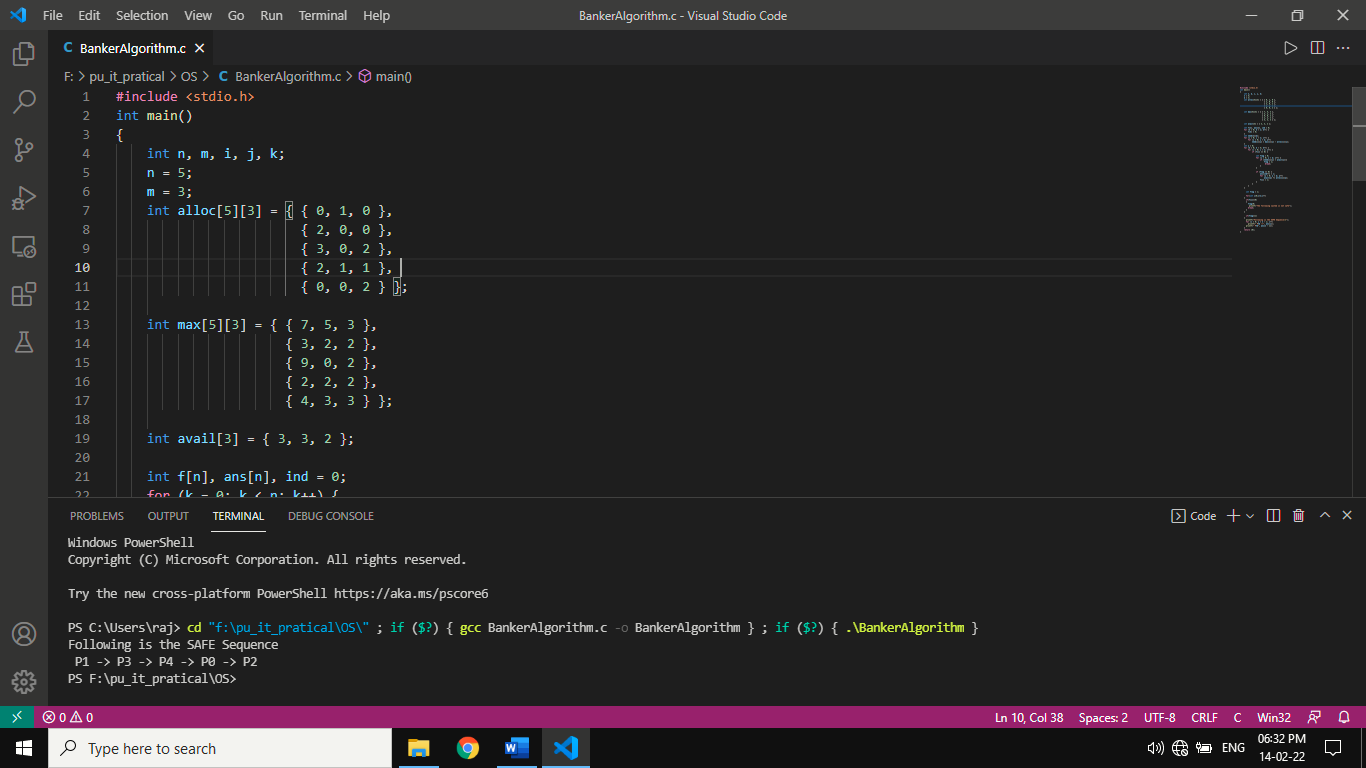
printf(" P%d ->", ans[i]);

printf(" P%d", ans[n - 1]);

}

return (0);

}

**OUTPUT:**