Practical : 1

Aim: Study of basic commands of Linux/UNIX.

Commands:

(1) whoami

To display the name of currently logged user, use whoami command in terminal. It displays current username when this command is invoked.

Syntax:

```
student@CEL4-41:~$ whoami
student
```

(2) pwd

The 'pwd' command stands for 'Print Working Directory'. The 'pwd' command prints your current working directory's path.

Syntax:

```
student@CEL4-41:~$ pwd
/home/student
```

(3) ls

This command lists the files and directories in your system. "ls" on its own lists all files in the current directory except for hidden files.

Syntax:

```
student@CEL4-41:~$ ls
01100925            bubble.c    circularqueue     Dev1.sh     f2.txt     firstfile     kam                  linkedlist.c    Pictures    secondfile
12                  c1.sh       circularqueue.c   devk        f4.txt     f.l           Kamaliya             linklist        pof1.sh     shellscript.sh
12121212            c2.c        cp                dev.txt     f5.txt     fork1.c       karan.c              linklist.c      pof.sh      string07.c
19cp005             c2.sh       csddev            Documents   f6.txt     gm.sh         kavi.c               loop.c          preet.c     t.c
adit                c3.c        d1.sh             Downloads   f7.txt     infinite.c    khushali.sh          loop_p1.c       Public      Templates
age.sh              c3.sh       d2.c              dwiti1.c    f8.txt     k1.c          krishna0129.c        loop_p2.c       queue.c     tower_of_hanoi_54.cpp
a.out               c4.sh       d2.sh             dwiti.c     file.c     k2.c          lex.yy.c             Music           rohan1.c    useoffork.c
binary_search01.cpp c5.sh       d2.txt            dwiti.txt   file.sh    k3.c          linear_search01.cpp  new.c           rohan.c     Videos
binary_search02.c   calc.sh     Desktop           EB.sh       first      k4.c          linearsearch.c       newlinklist     s1.c        VirtualBox VMs
bubble              %cd         dev               f10.txt     first.c    k5.c          linkedlist           newlinklist.c   S1.C        vyom
```

(4) cd

The 'cd' command in Linux stands for change directory. It is used to change the current directory of the terminal.

Syntax:

```
student@CEL4-41:~$ cd Nisarg
student@CEL4-41:~/Nisarg $
```

(5) mkdir

The 'mkdir' command (make directory) creates a new directory in the provided location.

Syntax:

```
student@CEL4-41:~$ mkdir Nisarg
```

(6) echo

The echo command allows users to display lines or text or string that are passed as arguments. It is commonly used in shell scripts and batch files to output status text to the screen or a file.

- Syntax: echo command which display input line in shell itself.

```
student@CEL4-41:~/vyom$ echo "Hello Nisarg"
Hello Nisarg
```

- Syntax: echo command which display input lines in files.

```
student@CEL4-41:~/Nisarg $ echo "Hello vyom" > hello.txt
```

Output:

(7) cat file.txt

This command is used to show the content of file.

Syntax:

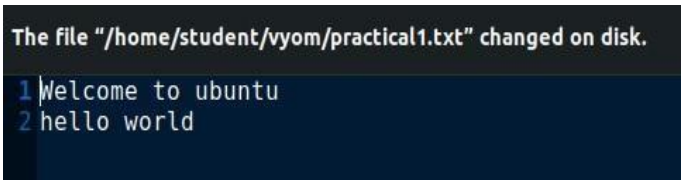
```
student@CEL4-41:~/Nisarg $ cat practical1.txt
Welcome to ubuntu
hello world
```

(8) cat >

If you want to create a new file or overwrite an existing file with new content, you can use 'cat' with the output redirection (' > ') .

Syntax:

```
student@CEL4-41:~/Nisarg $ cat > practical1.txt
Welcome to ubuntu
hello world
```



(9) ls

This command lists the files and directories in your system.

Syntax:

```
student@CEL4-41:~/Nisarg $ ls
hello.txt  practical1.txt
```

(10) cat file1.txt file2.txt

The 'cat' command can also concatenate the contents of multiple files.

Syntax:

```
student@CEL4-41:~/Nisarg $ cat hello.txt practical1.txt
Hello Nisarg
Welcome to ubuntu
hello world
```

Practical : 2

Aim: Study of Advance commands and filters of Linux/UNIX Commands:

(1) wc

The 'wc' command in linux is most powerful command which is used to count number of lines, words and characters in a file.

Syntax:

```
student@CEL4-41:~/Nisarg$ wc practical1.txt
 4  5 32 practical1.txt
```

- ➔ wc – l file.txt : This command counts only number of lines in file.
- ➔ wc – w file.txt : This command counts only number of words in file.
- ➔ wc – c file.txt : This command counts only number of characters in file.

Syntax:

```
student@CEL4-41:~/Nisarg$ wc -l practical1.txt
4 practical1.txt
student@CEL4-41:~/Nisarg$ wc -w practical1.txt
5 practical1.txt
student@CEL4-41:~/Nisarg$ wc -c practical1.txt
32 practical1.txt
```

(2) pwd

The 'pwd' command stands for 'Print Working Directory'. The 'pwd' command prints your current working directory's path.

Syntax:

```
student@CEL4-41:~/Nisarg$ pwd
/home/student/Nisarg
```

(3) cd ..

The dot dot (..) refers to the directory immediately above the current directory, its parent directory. This (..) refers to come out of present directory.

Syntax:

```
student@CEL4-41:~/Nisarg$ cd ..
student@CEL4-41:~$
```

(4) pwd

The 'pwd' command prints your current working directory's path.

Syntax:

```
student@CEL4-41:~$ pwd
/home/student
```

(5) cd

The 'cd' command in Linux stands for change directory. It is used to change the current directory of the terminal.

Syntax:

```
student@CEL4-41:~$ cd \Nisarg
bash: cd: Nisarg : No such file or directory
```

(6) date

The date command will output the date and time in the time-zone defined in our system.

Syntax:

```
student@CEL4-41:~$ date
Saturday 17 February 2024 04:19:31 PM IST
```

(7) cd --help

This command helps you to find information or formats of another commands. It is used to display help manual.

Syntax:

```
student@CEL4-41:~$ cd --help
cd: cd [-L|[-P [-e]] [-@]] [dir]
    Change the shell working directory.

    Change the current directory to DIR.  The default DIR is the value of the
    HOME shell variable.

    The variable CDPATH defines the search path for the directory containing
    DIR.  Alternative directory names in CDPATH are separated by a colon (:).
    A null directory name is the same as the current directory.  If DIR begins
    with a slash (/), then CDPATH is not used.

    If the directory is not found, and the shell option `cdable_vars' is set,
    the word is assumed to be  a variable name.  If that variable has a value,
    its value is used for DIR.

    Options:
      -L        force symbolic links to be followed: resolve symbolic
                links in DIR after processing instances of `..'
      -P        use the physical directory structure without following
                symbolic links: resolve symbolic links in DIR before
                processing instances of `..'
      -e        if the -P option is supplied, and the current working
                directory cannot be determined successfully, exit with
                a non-zero status
      -@        on systems that support it, present a file with extended
                attributes as a directory containing the file attributes

    The default is to follow symbolic links, as if `-L' were specified.
    `..' is processed by removing the immediately previous pathname component
    back to a slash or the beginning of DIR.

    Exit Status:
    Returns 0 if the directory is changed, and if $PWD is set successfully when
    -P is used; non-zero otherwise.
```

(8) date --help

This command helps you to find different/various information or formatting characters of date and time-zone.

Syntax:

```
student@CEL4-41:~$ date --help
```

```
%%   a literal %
%a   locale's abbreviated weekday name (e.g., Sun)
%A   locale's full weekday name (e.g., Sunday)
%b   locale's abbreviated month name (e.g., Jan)
%B   locale's full month name (e.g., January)
%c   locale's date and time (e.g., Thu Mar  3 23:05:25 2005)
%C   century; like %Y, except omit last two digits (e.g., 20)
%d   day of month (e.g., 01)
%D   date; same as %m/%d/%y
%e   day of month, space padded; same as %_d
%F   full date; same as %Y-%m-%d
%g   last two digits of year of ISO week number (see %G)
%G   year of ISO week number (see %V); normally useful only with %V
%h   same as %b
%H   hour (00..23)
%I   hour (01..12)
%j   day of year (001..366)
%k   hour, space padded ( 0..23); same as %_H
%l   hour, space padded ( 1..12); same as %_I
%m   month (01..12)
%M   minute (00..59)
%n   a newline
%N   nanoseconds (000000000..999999999)
```

```
%p   locale's equivalent of either AM or PM; blank if not known
%P   like %p, but lower case
%q   quarter of year (1..4)
%r   locale's 12-hour clock time (e.g., 11:11:04 PM)
%R   24-hour hour and minute; same as %H:%M
%s   seconds since 1970-01-01 00:00:00 UTC
%S   second (00..60)
%t   a tab
%T   time; same as %H:%M:%S
%u   day of week (1..7); 1 is Monday
%U   week number of year, with Sunday as first day of week (00..53)
%V   ISO week number, with Monday as first day of week (01..53)
%w   day of week (0..6); 0 is Sunday
%W   week number of year, with Monday as first day of week (00..53)
%x   locale's date representation (e.g., 12/31/99)
%X   locale's time representation (e.g., 23:13:48)
%y   last two digits of year (00..99)
%Y   year
%z   +hhmm numeric time zone (e.g., -0400)
%:z  +hh:mm numeric time zone (e.g., -04:00)
%::z  +hh:mm:ss numeric time zone (e.g., -04:00:00)
%:::z  numeric time zone with : to necessary precision (e.g., -04, +05:30)
%Z   alphabetic time zone abbreviation (e.g., EDT)
```

- Different formatting characters of date and time-zone:
  Syntax:

```
student@CEL4-41:~$ date +"%m"
02
student@CEL4-41:~$ date +%m
02
student@CEL4-41:~$ date +%M
22
student@CEL4-41:~$ date +%y
24
student@CEL4-41:~$ date +%Y
2024
student@CEL4-41:~$ date +"%Y"
2024
student@CEL4-41:~$ date +"%d"
17
student@CEL4-41:~$ date +"%D"
02/17/24
```

- Syntax for Year , Month and Day :

```
student@CEL4-41:~$ date +"Year: %Y, Month: %m, Day: %d"
Year: 2024, Month: 02, Day: 17
```

(9) mkdir --help

This command display help information for creating new directory file.

Syntax:

```
student@CEL4-41:~$ mkdir --help
Usage: mkdir [OPTION]... DIRECTORY...
Create the DIRECTORY(ies), if they do not already exist.

Mandatory arguments to long options are mandatory for short options too.
  -m, --mode=MODE   set file mode (as in chmod), not a=rwx - umask
  -p, --parents     no error if existing, make parent directories as needed
  -v, --verbose     print a message for each created directory
  -Z                   set SELinux security context of each created directory
                         to the default type
      --context[=CTX]  like -Z, or if CTX is specified then set the SELinux
                         or SMACK security context to CTX
      --help     display this help and exit
      --version  output version information and exit

GNU coreutils online help: <https://www.gnu.org/software/coreutils/>
Full documentation at: <https://www.gnu.org/software/coreutils/mkdir>
or available locally via: info '(coreutils) mkdir invocation'
```

(10) cal

The cal command in Linux is used to display a simple calendar in the terminal.

- **cal** : This command will display the calendar for the current month, highlighting the current date.

Syntax:

```
student@CEL4-41:~$ cal
   February 2024
Su Mo Tu We Th Fr Sa
                1  2  3
 4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29
```

- **cal october 2003 :** 1’his command will display the calendaí foí Octobeí 2023. You can íeplace "10" with the month and "2023" with the desiíed yeaí.

Syntax:

```
student@CEL4-41:~$ cal october 2003
    October 2003
Su Mo Tu We Th Fr Sa
             1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31
```

- **cal 2003 :** This command will show the calendars for all twelve months of the year 2003.you can display for specific year.

Syntax:

```
student@CEL4-41:~$ cal 2003
                                2003
      January                   February                  March
Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa
          1  2  3  4                     1                     1
 5  6  7  8  9 10 11   2  3  4  5  6  7  8   2  3  4  5  6  7  8
12 13 14 15 16 17 18   9 10 11 12 13 14 15   9 10 11 12 13 14 15
19 20 21 22 23 24 25  16 17 18 19 20 21 22  16 17 18 19 20 21 22
26 27 28 29 30 31     23 24 25 26 27 28     23 24 25 26 27 28 29
                                            30 31

       April                     May                       June
Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa
       1  2  3  4  5                 1  2  3   1  2  3  4  5  6  7
 6  7  8  9 10 11 12   4  5  6  7  8  9 10   8  9 10 11 12 13 14
13 14 15 16 17 18 19  11 12 13 14 15 16 17  15 16 17 18 19 20 21
20 21 22 23 24 25 26  18 19 20 21 22 23 24  22 23 24 25 26 27 28
27 28 29 30           25 26 27 28 29 30 31  29 30


        July                     August                  September
Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa
       1  2  3  4  5                 1  2      1  2  3  4  5  6
 6  7  8  9 10 11 12   3  4  5  6  7  8  9   7  8  9 10 11 12 13
13 14 15 16 17 18 19  10 11 12 13 14 15 16  14 15 16 17 18 19 20
20 21 22 23 24 25 26  17 18 19 20 21 22 23  21 22 23 24 25 26 27
27 28 29 30 31        24 25 26 27 28 29 30  28 29 30
                      31

      October                  November                  December
Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa
          1  2  3  4                     1      1  2  3  4  5  6
 5  6  7  8  9 10 11   2  3  4  5  6  7  8   7  8  9 10 11 12 13
12 13 14 15 16 17 18   9 10 11 12 13 14 15  14 15 16 17 18 19 20
19 20 21 22 23 24 25  16 17 18 19 20 21 22  21 22 23 24 25 26 27
26 27 28 29 30 31     23 24 25 26 27 28 29  28 29 30 31
```

(11) history

'history' command is used to display the history of the commands executed by the user. It provides a chronological list of previously executed commands, along with corresponding command numbers.

Syntax:

```
student@CEL4-41:~$ history
```

(12) clear

Clear is standard Unix computer OS command that is used to clear the terminal screen.

Syntax:

```
student@CEL4-41:~$ clear
```

Practical 3

Aim :- Write a shell script to generate marksheet of a student. Take 3 subjects, calculate and display total marks, percentage and Class obtained by the student.

```
echo "Enter your ROll NUMBER"
read r
echo "Enter your Name"
read name
echo "Enter MAX Marks possible"
read MAX
echo "Enter Marks in English"
read eng
echo "Enter Marks in Maths"
read mat
echo "Enter Marks in Science"
read sci

total=$(( $eng+$mat+$sci ))
echo "Total Marks Is: $total"

grandmax=$(( $MAX*3 ))
if [ $total -gt $grandmax ]
then
echo "You have entered wrong marks"
else
percentage=$(( ($total*100)/$grandmax ))
fi

echo "Mr/Ms $name your Annual Report is: $percentage%"

if [ $percentage -ge 80 ]
then
```

```
echo "Congratulations! You have passed with Distinction"
elif [ $pecentage -ge 70 ]
then
echo "Congratulations! You have passed with First Class"
elif [ $pecentage -ge 50 ]
then
echo "Congratulations! You have passed with Second Class"
elif [$pecentage -ge 33 ]
then
echo "Congratulations! You have passed with Third Class"
else
echo "Try again next time! You have Failed"
fi
```

```
jordan@Ubuntu:~/Desktop/jordan$ touch marksheet.sh
jordan@Ubuntu:~/Desktop/jordan$ ls
calc.sh  experiment.sh  factorial.sh  ff.sh  marksheet.sh  table.sh
jordan@Ubuntu:~/Desktop/jordan$ sh marksheet.sh
Enter your ROll NUMBER
55
Enter your Name
Rushabh
Enter MAX Marks possible
100
Enter Marks in English
95
Enter Marks in Maths
100
Enter Marks in Science
98
Total Marks Is: 293
Mr/Ms Rushabh your Annual Report is: 97%
Congratulations! You have passed with Distinction
jordan@Ubuntu:~/Desktop/jordan$
```

Practical 4

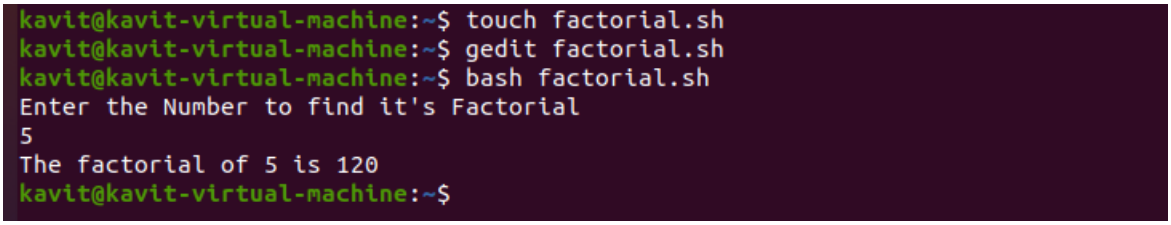
Aim :- Write a shell script to display multiplication table of given number.

```
echo "Enter the number to find it's Table"
read a
for i in `seq 1 10`
do
        echo "$a * $i = $ (( $a*$i ))"
done
```

Practical 5

Aim :- Write a shell script to find factorial of given number n.

```
echo "Enter the Number to find it's Factorial"
read n
i=1
fact=1
while [ $i -le $n ]
do
fact=$(( $fact*i ))
i=$(( $i+1 ))
done
echo "The factorial of $n is $fact"
```



Practical 6

Aim :- Write a menu driven shell script which will print the following menu and execute the given task.

a. Display calendar of current month
b. Display today's date and time
c. Display usernames those are currently logged in the system
d. Display your name at given x, y position
e. Display your terminal number

```
#!/bin/bash
echo "Welcome Back!"
echo "How can i help you?"
echo "Press 1 to Display Calendar"
echo "Press 2 to Display current DATE and TIME"
echo "Press 3 to Display current users"
echo "Press 4 to Display name at X,Y position"
echo "Press 5 to Display Terminal Number"
read i

case "$i" in

        "1") calendar=$(cal)
            echo "$calendar";;

        "2") datetime=$(date)
            echo "$datetime";;

        "3") users=$(who)
            echo "$users";;

        "4") echo -e "\033[19;20HKavit";;

        "5") terminal=$(tty)
            echo "$terminal";;
```

```
        *) echo "Please Enter Valid Numbers";;
esac
```

```
kavit@kavit-virtual-machine:~$ bash case.sh
Welcome Back!
How can i help you?
Press 1 to Display Calendar
Press 2 to Display current DATE and TIME
Press 3 to Display current users
Press 4 to Display name at X,Y position
Press 5 to Display Terminal Number
1
     April 2022
Su Mo Tu We Th Fr Sa
                1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30

kavit@kavit-virtual-machine:~$ bash case.sh
Welcome Back!
How can i help you?
Press 1 to Display Calendar
Press 2 to Display current DATE and TIME
Press 3 to Display current users
Press 4 to Display name at X,Y position
Press 5 to Display Terminal Number
3
kavit    :0           2022-04-26 09:05 (:0)
```

```
kavit@kavit-virtual-machine:~$ bash case.sh
Welcome Back!
How can i help you?
Press 1 to Display Calendar
Press 2 to Display current DATE and TIME
Press 3 to Display current users
Press 4 to Display name at X,Y position
Press 5 to Display Terminal Number
5
/dev/pts/0
```

Practical 7

Aim :- Write a shell script to read n numbers as command arguments and sort them in descending order.

```
#!/bin/bash
echo "enter the size of array"
read n
echo "enter Numbers in array:"
for (( i = 0; i < $n; i++ ))
do
read nos[$i]
done
echo " Numbers in an array before sorting are:"
for (( i = 0; i < $n; i++ ))
do
echo ${nos[$i]}
done
for (( i = 0; i < $n ; i++ ))
do
for (( j = $i; j < $n; j++ ))
do
if [ ${nos[$i]} -lt ${nos[$j]} ]; then
t=${nos[$i]}
nos[$i]=${nos[$j]}
nos[$j]=$t
fi
done
done
echo -e "\nNumbers in array after sorting are: "
for (( i=0; i < $n; i++ ))
do
echo ${nos[$i]}
done
```

```
kavit@kavit-virtual-machine:~$ bash sort.sh
Enter the size of array
5
Enter  Numbers in array:
598
69
12
4
365
  Numbers in an array before sorting are:
598
69
12
4
365

Numbers in array after sorting are:
4
12
69
365
598
kavit@kavit-virtual-machine:~$
```

Practical 8

Aim: Shell programming using filters (grep, egrep, fgrep)

The grep filter searches a file for a particular pattern of characters and displays all lines that contain that pattern. Both egrep and fgrep are derived from the base grep command. The “egrep” stands for “extended grep” while the fgrep stands for “fixed- string grep.” An egrep command is used to search for multiple patterns inside a file or other kind of data repository while frgrep is used to look for strings.

grep :- Grep is a Linux / Unix command-line tool used to search for a string of characters in a specified file.

```
kavit@kavit-virtual-machine:~$ grep printf sjf.c
    printf("Enter number of process:");
    printf("\nEnter Burst Time:\n");
        printf("p%d:", i + 1);
  printf("nProcesst\t Burst Time \t Waiting Time \t Turnaround Time");
        printf("\np%d\t\t  %d\t\t    %d\t\t\t%d", i, bt[i], wt[i], tat[i]);
    printf("\n\nAverage Waiting Time=%f", avg_wt);
    printf("\nAverage Turnaround Time=%f\n", avg_tat);
kavit@kavit-virtual-machine:~$
```

egrep :- egrep is a pattern searching command which belongs to the family of grep functions. It works the same way as grep -E does. It treats the pattern as an extended regular expression and prints out the lines that match the pattern.

```
        scanf("%d", &bt[i]);
kavit@kavit-virtual-machine:~$ egrep scanf  sjf.c
    scanf("%d", &n);
        scanf("%d", &bt[i]);
kavit@kavit-virtual-machine:~$
```

fgrep:- The fgrep (fast grep) command searches files for a character string and prints all lines that contain that string. fgrep is different from grep(1) and egrep(1) because it searches for a string, instead of searching for a pattern that matches an expression. fgrep uses a fast and compact algorithm.

```
kavit@kavit-virtual-machine:~$ fgrep int sjf.c
int main()
    int bt[20], p[20], wt[20], tat[20], i, j, n, total = 0, pos, temp;
    printf("Enter number of process:");
    printf("\nEnter Burst Time:\n");
        printf("p%d:", i + 1);
  printf("nProcesst\t Burst Time \t Waiting Time \t Turnaround Time");
        printf("\np%d\t\t  %d\t\t    %d\t\t\t%d", i, bt[i], wt[i], tat[i]);
    printf("\n\nAverage Waiting Time=%f", avg_wt);
    printf("\nAverage Turnaround Time=%f\n", avg_tat);
kavit@kavit-virtual-machine:~$
```

Practical 9

<u>AIM:</u> Write a shell script to read n numbers as command arguments and sort them in descending order.

CODE: -

```
#!/bin/bash
echo &quot;enter maximum number&quot;
read n
# taking input from user
echo &quot;enter Numbers in array:&quot;
for (( i = 0; i &lt; $n; i++ ))
do
read nos[$i]
done
#printing the number before sorting
echo &quot; Numbers in an array are:&quot;
for (( i = 0; i &lt; $n; i++ ))
do
echo ${nos[$i]}
done
# Now do the Sorting of numbers
for (( i = 0; i &lt; $n ; i++ ))
do
for (( j = $i; j &lt; $n; j++ ))
do
if [ ${nos[$i]} -lt ${nos[$j]} ]; then
t=${nos[$i]}
nos[$i]=${nos[$j]}
nos[$j]=$t
fi
done
done
# Printing the sorted number in descending order
echo -e &quot;\nSorted Numbers &quot;
for (( i=0; i &lt; $n; i++ ))
do
echo ${nos[$i]}
done
```

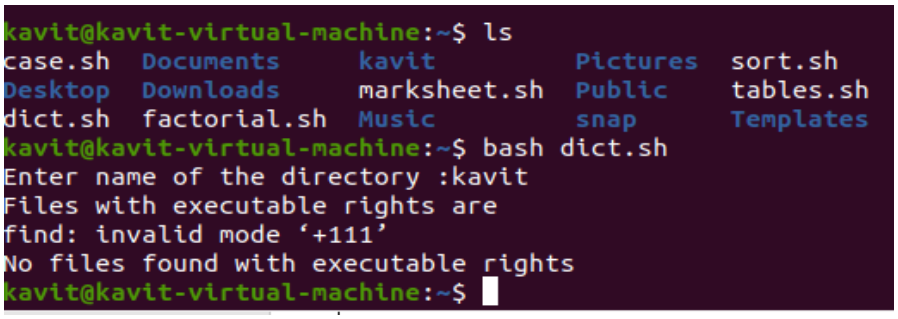
OUTPUT:

```
 Enter maximum number 5
Enter Numbers in array :
10 3 2 45 8
Numbers in array are :
10
3
2
45
8
Sorted Numbers
45
10
8
3
2
```

Practical 10

AIM: Write a shell script to display all executable files, directories and zero sized files from current directory.

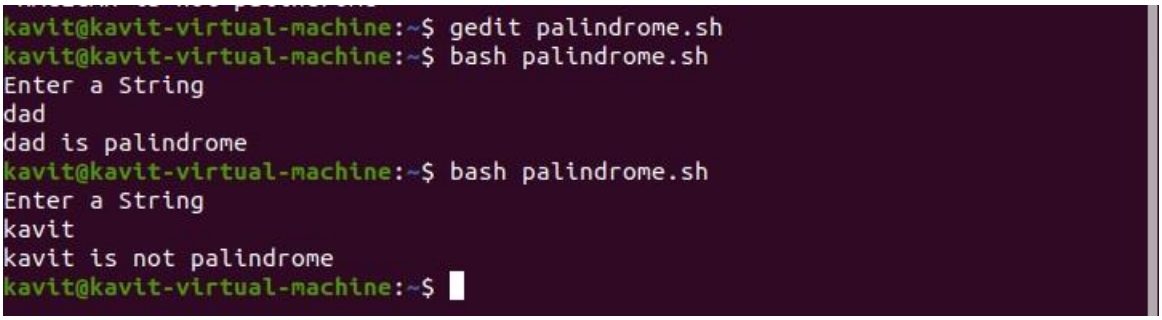
```
echo -n "Enter name of the directory :"
read directory
 if [ ! -d $directory ]
then
   echo "Directory not exist"
else
        count=0
        echo "Files with executable rights are"
       for i in $(find $directory -type f -perm +111)
       do
              echo $i
              count=$(echo count + 1 | bc -l)
       done
        if [ $count -eq 0 ]
        then
                 echo "No files found with executable rights"
        fi
fi
```



Practical 11

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

```
echo “enter a string:”
read s
reverse=””
len=${#s}
for (( i=$len-1; i>=0; i-- ))
do
        reverse=”$reverse${s:1}”
done
if [ $s == $reverse ]
then
        echo “$s is palindrome”
else
        echo “$s is not palindrome”
fi
```



Practical 12

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

```
echo "Date validator"
dd=0
mm=0
yy=0
days=0

read -p "Enter day (dd) : " dd
read -p "Enter Month (mm) : " mm
read -p "Enter Year (yyyy) : " yy
 if [ $mm -le 0 -o $mm -gt 12 ]
then
        echo "$mm is invalid month. "
        exit 1
fi
case $mm in
        1 | 3 | 5 | 7 | 8 | 10 | 12)
                days=31
                ;;
        2)
                days=28
                ;;
        4 | 6 | 9 | 11)
                days=30
                ;;
        *)
                days=-1
                ;;
esac
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"
```

```
kavit@kavit-virtual-machine:~$
kavit@kavit-virtual-machine:~$ gedit check.sh
kavit@kavit-virtual-machine:~$ bash check.sh
Date validator
Enter day (dd) : 31
Enter Month (mm) : 5
Enter Year (yyyy) : 2003
31/5/2003 is a Valid Date
kavit@kavit-virtual-machine:~$ bash check.sh
Date validator
Enter day (dd) : 32
Enter Month (mm) : 89
Enter Year (yyyy) : 0000
89 is invalid month.
kavit@kavit-virtual-machine:~$
```

Practical 13

<u>AIM :</u> Write an awk program using function,which convert each word in a given text into capital.

awk ‘{print toupper ($0) }’ n1.txt

```
kandarp@kandarp-virtual-machine:~/Desktop$ ./11.sh
HELLO WORLD
```

Practical 14

<u>AIM:</u> Write a program which demonstrate the use of fork , join and exec and wait system call.

→FORK()

```
#include<stdio.h>
int main()
{
        printf("Hi\n");
        fork();
        fork();
        printf("Hello\n");
        return 0;
}
```

OUTPUT: Hi
Hello
Hello
Hello
Hello

→JOIN()

```
vi f1.txt
vi f2.txt
join f1.txt f2.txt
```

f1.txt → 1 hello
2 hi
f2.txt→1 101
2 102

OUTPUT: 1 hello 101
2 hi 102

→WAIT()

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
	int pid=fork(),status;
	if(pid==0)
	{
		pritnf("\n child process");
		sleep(5);
		printf("\nchild process after 5 seconds");
	}
	if(pid>0)
	{
		printf("\ni am the parent ,the child is:%d",pid);
		pid=wait(&status);
		printf("\n end of process %d",pid);
		if(WIFEXITED(status))
		{
			printf("\nThe process is ended with
exit(%d).\n",WEXITSTATUS(status));
		}
		if(WIFSIGNALED(status))
		{
			printf("\nThe process is ended with
kill(%d).\n",WTERMSIG(status));
		}
	}
	exit(0);
	return 0;
}
```

OUTPUT:

child process
child process after 5 second I am the parent,the child is 3605
end of process 3605:
the process is ended with exit(0).

→EXEC()

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
        int pid=fork(),status;
        if(pid==0)
        {
                pritnf("\n child process");
                execl(("/bin/ls","ls","-l",/home/ubantu/",(char *) 0);
                perror("in exec():");
        }
        if(pid>0)
        {
                printf("\ni am the parent ,the child is:%d",pid);
                pid=wait(&status);
                printf("\n end of process %d",pid);
                if(WIFEXITED(status))
                {
                        printf("\nThe process is ended with
exit(%d).\n",WEXITSTATUS(status));
                }
                if(WIFSIGNALED(status))
                {
                        printf("\nThe process is ended with
kill(%d).\n",WTERMSIG(status));
```

```
            }
        }
        exit(0);
        return 0;
}
```

OUTPUT:

```
sudo vim exec.c
make exec
cc exec.c –o exec
./exec
I am the parent,and the child is 18264.
total 18
```

Practical 15

AIM: Write a c program to simulate FCFS CPU scheduling algorithm.

```
#include<stdio.h>
int main()
{
float avgtat=0,avgwt=0;
int n,i,max=0,ct=0,j,tat=0,wt=0,k=0;
printf("Enter the number of Processes: ");
scanf("%d",&n);
int p_id[n];
int p_AT[n];
int p_BT[n];
int p_TAT[n];
int p_WT[n];
int p_CT[n];
int arr[n];
for(i=0;i<n;i++)
{
	printf("Enter the Process ID: ");
	scanf("%d",&p_id[i]);
	printf("Enter the Process Arrival Time: ");
	scanf("%d",&p_AT[i]);
	printf("Enter the Process Burst Time: ");
	scanf("%d",&p_BT[i]);
	if(p_AT[i]>=max)
	max=p_AT[i];
}

for(i=0;i<=max;i++)
```

```
{
        for(j=0;j<n;j++)
                {
                  if(p_AT[j]==i)
                   {
                     ct=ct+p_BT[j];
                     p_CT[j]=ct;
                     p_TAT[j]=p_CT[j]-p_AT[j];
                     p_WT[j]=p_TAT[j]-p_BT[j];
                     tat+=p_TAT[j];
                     wt+=p_WT[j];
                     arr[k]=p_id[j];
                     k++;
                  }
                }
}
avgtat=(float)tat/n;
avgwt=(float)wt/n;
printf("P_ID\tP_AT\tP_BT\tP_CT\tP_WT\tP_TAT\n");
for(i=0;i<n;i++)
{

printf("%d\t%d\t%d\t%d\t%d\t%d\n",p_id[i],p_AT[i],p_BT[i],p_CT[i],p_WT[
i],p_TAT[i]);
}
printf("Average Turn Around Time is: %f\n",avgtat);
printf("Average Waiting Time is: %f\n",avgwt);

for(i=0;i<n;i++)
{
printf(" --------");
}
printf("\n");
for(i=0;i<n;i++)
```

```
printf("%d\t",arr[i]);
printf("\n");
for(i=0;i<n;i++)
{
printf(" --------");
}
printf("\n");
}
```

```
kavit@kavit-virtual-machine:~$ ./a.out
Enter the number of Processes: 5
Enter the Process ID: 1
Enter the Process Arrival Time: 2
Enter the Process Burst Time: 5
Enter the Process ID: 2
Enter the Process Arrival Time: 4
Enter the Process Burst Time: 2
Enter the Process ID: 3
Enter the Process Arrival Time: 6
Enter the Process Burst Time: 8
Enter the Process ID: 4
Enter the Process Arrival Time: 10
Enter the Process Burst Time: 2
Enter the Process ID: 5
Enter the Process Arrival Time: 12
Enter the Process Burst Time: 2
P_ID    P_AT    P_BT    P_CT    P_WT    P_TAT
1       2       5       5       -2      3
2       4       2       7       1       3
3       6       8       15      1       9
4       10      2       17      5       7
5       12      2       19      5       7
Average Turn Around Time is: 5.800000
Average Waiting Time is: 2.000000
--------------------------------------------
1       2       3       4       5
--------------------------------------------
```