Ex. No. 1(a)

BASIC UNIX COMMANDS

AIM:

To study of Basic UNIX Commands and various UNIX editors such as vi, ed, ex and EMACS.

BASIC COMMANDS:

a) Display Commands

1) **Command :** date
Purpose : To check the date and time
Syntax : $date
Example : $date

2) **Command :** month
Purpose : To display only month
Syntax : $+%m
Example : $+%m

3) **Command :** Month Name
Purpose : To display month name
Syntax : $+%h
Example : $+%h

4) **Command :** Month Day
Purpose : To display day of month
Syntax : $+%d
Example : $+%d

5) **Command :** year
Purpose : To display last two digits of years
Syntax : $+%y
Example : $+%y

6) **Command :** hour
Purpose : To display hours
Syntax : $+%H
Example : $+%H

7) **Command :** Minutes
Purpose : To display minutes
Syntax : $+%M
Example : $+%M

8) **Command :** Seconds
Purpose : To display seconds
Syntax : $+%S
Example : $+%S

b) **Command :** cal
Purpose : To display the calendar
Syntax : $cal
Example : $cal

c) **Command :** echo
Purpose : To print the message on the screen.
Syntax : $echo "text"
Example : $echo HELLO

d) **Command :** ls
Purpose : To list the files. Your files are kept in a directory.
Syntax : $ls
Example : $ls

lsls–s All files (include files with prefix)

ls–l Lodetai (provide file statistics)

ls–t Order by creation time

ls– u Sort by access time (or show when last accessed together with –l)

ls–s Order by size

ls–r Reverse order

ls–f Mark directories with **/**,executable with***** , symbolic links with @, local sockets with **=**, named pipes(FIFOs)with

ls–s Show file size

ls– h " Human Readable", show file size in Kilo Bytes & Mega Bytes (h can be used together with –l or)

e) **Command :** man
Purpose : To provide manual help on every UNIX commands.
Syntax : $man unix command
Example : $man cat

f) **Command :** who
Purpose : To displays data about all users who have logged into the

system currently.

Syntax : $who

Example : $ who -H. To show only hostname

$ who -m. To show active processes spawned by init

$ who -p. To show user's message status as

$ who -T. Show or list users logged in.

$ who -u. Show time of last system boot.

g) **Command :** whoami

Purpose : To displays about current user only

Syntax : $whoami

Example : $whoami

h) **Command :** uptime

Purpose : To tells you how long the computer has been running since its last reboot or power-off.

Syntax : $uptime

Example : $uptime

i) **Command :** uname

Purpose : To displays the system information such as hardware platform, system name and processor, OS type.

Syntax : $uname–a

Example : $uname–a

j) **Command :** hostname

Purpose : To displays and set system host name

Syntax : $ hostname

Example : $ hostname

k) **Command :** bc

Purpose : To stands for "best calculator"

Syntax : $bc

Example : $bc

l) **Command :** id

Purpose : To display the login name.

Syntax : $id

Example : $id

m) **Command :** clear

Purpose : To clear the screen.

Syntax : $clear
Example : $clear

n) **Command :** finger
Purpose : To gathers and displays information about the users which includelogin name, name of user, home directory etc..
Syntax : $finger username
Example : $finger student

FILE MANIPULATION COMMANDS

a) **Command :** cat
Purpose : To create, view, and edit files.
Syntax : CREATE :$cat>filename

VIEW: $cat filename

EDIT $cat>>filename
Example : $ cat>aaa

cat aaa

cat>> aaa

b) **Command :** concatenate
Purpose : To add two file content into new file
Syntax : $cat file1file2>file3
Example : $cat aaa bbb>ccc

c) **Command :** grep
Purpose : To search a particular word or pattern related to that word from the file.
Syntax : $grep search word filename
Example : $grep anu student

d) **Command :** rm
Purpose : To deletes a file from the file system
Syntax : $rm filename
Example : $rm student

e) **Command :** touch
Purpose : To create a blank file.
Syntax : $touch filename
Example : $touch student

f) **Command :** cp
Purpose : To copies the files or directories
Syntax : $cp source file destination file
Example : $cp student stud

g) **Command :** mv
Purpose : To rename the file or directory
Syntax : $mv old file new file
Example : $$mv student stu

h) **Command :** cut
Purpose : To cuts or pickup a given number of character or fields of the file.
Syntax : $cut<option><filename>
Example : $cut –c filename (-c cutting columns)

$cut–c1-10emp

$cut–f 3,6emp (-f cutting fields)

$ cut –f 3-6 emp

i) **Command :** wc
Purpose : To counts the number of lines, words, character in a specified file(s) with the options as –l,-w,-c
Syntax : $wc filename
Example : $$wc student -l
$$wc student -w

$$wc student -c

DIRECTORY COMMANDS

a) **Command :** mkdir
Purpose : To create a directory
Syntax : $mkdir <directory name>
Example : $mkdir student

b) **Command :** rmdir
Purpose : To delete a directory
Syntax : $rmdir <directory name>
Example : $rmdir student

c) **Command :** cd
Purpose : To change the current directory
Syntax : $cd
Example : $cd ~ (changes path to your home directory)

cd (changes parent directory)

d) **Command :** pwd (Print working Directory)
Purpose : To display the absolute pathname of current working directory.
Syntax : $pwd
Example : **$**pwd

PROCESS COMMANDS

a) **Command :** exit
Purpose : To terminate a process
Syntax : $exit
Example : $exit

b) **Command :** kill
Purpose : To terminates or send a signal to process
Syntax : $kill
Example : $kill

c) **Command :** passwd
Purpose : To create or change a password
Syntax : $passwd
Example : $passwd

d) **Command :** semicolon (;)
Purpose : To execute more than one command at a time
Syntax : $;
Example : $who; date;

FILTER COMMANDS

a) **Command :** head
Purpose : To display lines from the head(top)of a given file
Syntax : $head filename
Example : $head student

$head -2student (To display the top two lines:)

b) **Command :** tail
Purpose : To display last 10 lines of the file
Syntax : $tail filename
Example : $tail student

$tail -2filename(To display the bottom two lines)

c) **Command :** chmod

Purpose : To change the permissions of a file or directory.

Syntax : $ch mod category operation permission file

(Category–is the user type, Operation–is used to assign or remove permission, Permission–is the type of permission, File–are used to assign or remove permission all)

Example : $ch modu+rw,g+rwstudent

Assigns read and write permission for users and groups

$chmodg=rwx student

Assigns absolute permission for groups of all read, write and execute permissions

$chmodu-wx student

Removes write and execute permission for users

RESULT:

Ex. No. 1(b)

C programs to simulate UNIX commands like cp, ls, grep

AIM:

To write C programs to simulate UNIX commands like cp, ls, grep.

a) PROGRAM FOR SIMULATION OF CP UNIX COMMANDS

ALGORITHM:

STEP 1: Start the program
STEP 2: Declare the variables ch, *fp, sc=0
STEP 3: Open the file in read mode
STEP 4: Get the character
STEP 5: If ch== " " then increment sc value by one
STEP 6: Print no of spaces
STEP 7: Close the file

PROGRAM:

```
#include<fcntl.h>
#include<unistd.h>
#include<stdio.h>
main(int argc,char *argv[])
{
        FILE *fp;
        char ch;
        int sc=0;
        fp=fopen(argv[1],"r");
        if(fp==NULL)
        printf("unable to open a file",argv[1]);
        else
        {
                while(!feof(fp))
                {
                        ch=fgetc(fp);
                        if(ch==' ')
                        sc++;
                }
        printf("no of spaces %d",sc);
        printf("\n");
        fclose(fp);
        }
}
```

b) PROGRAM FOR SIMULATION OF LS UNIX COMMANDS

ALGORTIHM:

STEP 1 : Start the program
STEP 2 : Open the directory with directory object dp
STEP 3 : Read the directory content and print it.
STEP 4: Close the directory.

PROGRAM:

```
#include<stdio.h>
#include<dirent.h>
main(int argc, char **argv)
{
        DIR *dp;
        struct dirent *link;
        dp=opendir(argv[1]);
        printf("\n contents of the directory %s are \n", argv[1]);
        while((link=readdir(dp))!=0)
        printf("%s",link->d_name);
        closedir(dp);
}
```

C) PROGRAM FOR SIMULATION OF GREP UNIX COMMANDS

ALGORITHM

STEP 1: Start the program
STEP 2: Declare the variables fline[max], count=0, occurrences=0 and pointers *fp,*newline.
STEP 3: Open the file in read mode.
STEP 4: In while loop check fgets(fline,max,fp)!=NULL
STEP 5: Increment count value.
STEP 6: Check newline=strchr(fline, „\n")
STEP 7: print the count,fline value and increment the occurrence value.
STEP 8: Stop the program

PROGRAM:

```
#include<stdio.h>
#include<string.h>
#define max 1024
void usage()
{
```

```
        printf("usage:\t. /a.out filename word \n ");
    }
    int main(int argc, char *argv[])
    {
        FILE *fp;
        char fline[max];
        char *newline;
        int count=0;
        int occurrences=0;
        if(argc!=3)
        {
            usage();
            exit(1);
        }
        if(!(fp=fopen(argv[1],"r")))
        {
            printf("grep: couldnot open file : %s \n",argv[1]);
            exit(1);
        }
        while(fgets(fline,max,fp)!=NULL)
        {
            count++;
            if(newline=strchr(fline, „\n"))
            *newline="\0";
            if(strstr(fline,argv[2])!=NULL)
        {
            printf("%s: %d %s \n", argv[1],count, fline);
            occurrences++;
        }
    }
    }
```

RESULT:

Ex. No. 1(c)

SIMPLE SHELL PROGRAMS

AIM:

To write simple shell programs by using conditional, branching and looping statements.

1.Write a Shell program to check the given number is even or odd

ALGORITHM:

SEPT 1: Start the program.
STEP 2: Read the value of n.
STEP 3: Calculate „r=expr $n%2".
STEP 4: If the value of r equals 0 then print the number is even
STEP 5: If the value of r not equal to 0 then print the number is odd.

PROGRAM:

```
echo "Enter the Number"
read n
r=`expr $n % 2`
if [ $r -eq 0 ]
then
        echo "$n is Even number"
else
        echo "$n is Odd number"
fi
```

2.Write a Shell program to check the given year is leap year or not

ALGORITHM:

SEPT 1: Start the program.
STEP 2: Read the value of year.
STEP 3: Calculate „b=expr $y%4".
STEP 4: If the value of b equals 0 then print the year is a leap year
STEP 5: If the value of r not equal to 0 then print the year is not a leap year.

PROGRAM:

```
echo "Enter the year"
read y
b=`expr $y % 4`
if [ $b -eq 0 ]
then
```

```
echo "$y is a leap year"
else
echo "$y is not a leap year"
fi
```

3.Write a Shell program to find the factorial of a number

ALGORITHM:

SEPT 1: Start the program.
STEP 2: Read the value of n.
STEP 3: Calculate „i=expr $n-1".
STEP 4: If the value of i is greater than 1 then calculate „n=expr $n * $i" and „i=expr $i – 1"
STEP 5: Print the factorial of the given number.

PROGRAM:

```
echo "Enter a Number"
read n
i=`expr $n - 1`
p=1
while [ $i -ge 1 ]
do
n=`expr $n \* $i`
i=`expr $i - 1`
done
echo "The Factorial of the given Number is $n"
```

4.Write a Shell program to swap the two integers

ALGORITHM:

SEPT 1: Start the program.
STEP 2: Read the value of a,b.
STEP 3: Calculate the swapping of two values by using a temporary variable temp.
STEP 4: Print the value of a and b.

PROGRAM:

```
echo "Enter Two Numbers"
read a b
temp=$a
a=$b
b=$temp
```

```
echo "after swapping"
echo $a $b
```

5.Write a Shell program for Academic and Personal Details

ALGORITHM:

STEP 1. Get name, age, and address from the user.
STEP 2. Print that message as similar.
STEP 3. Get mark1, mark2, and mark3 from the user.
STEP 4. Print that message as similar.

PROGRAM:

```
echo –n “Enter the name”
read s
echo –n “Enter the age”
read a
echo –n “Enter the address”
read adr
echo –n “The name is $s”
echo –n “The age is $a”
echo –n “The address is $adr”
echo –n “Enter the mark1”
read m1
echo –n “Enter the mark2”
read m2
echo –n “Enter the mark3”
read m3
echo –n “The mark1 is $m1”
echo –n “The mark2 is $m2”
echo –n “The mark3 is $m3”
```

6.Write a Shell program for Greatest Among Three Numbers

ALGORITHM:

STEP 1. Start the program.
STEP 2. Enter any three numbers.
STEP 3. Read the values as a, b and c.
STEP 4. If a greater than b and greater than c, print the value of a as the Greatest number.
STEP 5. Else if b is greater than c, print the value of b as greatest number.
STEP 6. Else print the value of c as the greatest number.
STEP 7. Stop the program.

PROGRAM:

```
echo greatest of 3 numbers
echo enter the numbers
read a
read b
read c
if test $a -gt $b -a $a -gt $c
then
echo $a is greater
elif test $b -gt $c
then
echo $b is greater
else
echo $c is greater
fi
```

7.Write a Shell program to Check Prime Number

ALGORITHM:

STEP 1. Start the program
STEP 2. Input number n
STEP 3. Is i>2, repeat the following steps.
STEP 4. p=n%i.
STEP 5. Is p=0 then increment t value by 1.
STEP 6. Decrement i value by one
STEP 7. Is t value > zero then print number is prime
STEP 8. Otherwise print number is not a prime
STEP 9. Stop the program

PROGRAM:

```
echo "Enter a Number"
read n
i=`expr $n - 1`
t=0
while [ $i -ge 2 ]
do
p=`expr $n % $i`
if [ $p -eq 0 ]
then
t=`expr $t + 1`
fi
i=`expr $i - 1`
done
```

```
if [ $t -gt 0 ]
then
echo "The Number $n is not a Prime Number"
else
echo "The Number $n is a Prime Number"
fi
```

8.Write a Shell program for Sum of N Numbers

ALGORITHM:

STEP 1. Start the program
STEP 2. Get the limit number 'n' from the user
STEP 3. Initialize sum=0 and i=1
STEP 4. If I is less than or equal to n get the number to be summed from the user and increment I value by i
STEP 5. Add the number to the sum value
STEP 6. Repeat step 4 and 5
STEP 7. Print the sum value
STEP 8. Stop the program.

PROGRAM:

```
echo "enter the limit"
read n
echo "enter the $n numbers"
sum=0
i=1
while test $i -le $n
do
read num
sum=`expr $num + $sum`
i=`expr $i + 1`
done
echo "the sum of the numbers are $sum"
```

9.Write a Shell program for Fibonacci Series

ALGORITHM:

STEP 1. Input the range(n)
STEP 2. Initialize b=1,a=0,s=0
STEP 3. Do the following in a loop, until s less than or equal to n
a=b;b=s
Print the Fibonacci series value i.e.(s)
Let s=a+b
STEP 4. Stop.

PROGRAM:

```
echo "Enter the range to be displayed"
read n
a=0
b=1
s=0
echo "Fibonacci series"
while test $s -le $n
do
a=$b
b=$s
echo $s
s=`expr $a + $b`
done
```

10.Write a Shell program for Armstrong Number

ALGORITHM:

STEP 1. Start the program.
STEP 2. Get the input value(num)
STEP 3. Assign value of num to x
STEP 4. Assign value of sum equal to zero.
STEP 5. Repeat the following steps till the num greater than zero
STEP 6. Find y equal to num modulus 10.Find z equal to cube of y. Then find num equal to num divided by 10.
STEP 7. If x equal to sum then print the value is Armstrong Otherwise print the number is not armstrong
STEP 8. Stop the program.

PROGRAM:

```
echo "Enter a Number"
read num
x=$num
sum=0
while [$num -gt 0]
do
y=`expr $num % 10`
z=`expr $y \* $y \* $y`
sum=`expr $sum + $z`
num=`expr $num / 10`
done
if [$x -eq $sum]
then
```

```
echo "$x is an armstrong Number"
else
echo "$x is not an armstrong Number"
fi
```

11.Write a Shell program for Arithmetic Operations Using Switch Case

ALGORITHM:

STEP 1. Enter the input
STEP 2. Enter the choice
STEP 3. Perform corresponding arithmetic operation on the inputs
STEP 4. Display the result
STEP 5. Stop

PROGRAM:

```
echo "Performing Arithmetic Manipulation"
echo "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
echo -n "Enter the first no:"
read a
echo -n "Enter the second no:"
read b
echo "Arithmetic Operation - Menu"
echo "1.Addition"
echo "2.Subtraction"
echo "3.Multiplication"
echo "4.Division"
echo "0.Exit"
echo -n "Enter your Choice:"
read ch
case $ch in
1) echo " Option 1 Performs Addition" ;
    c=`expr $a + $b`;
    echo "The sum of $a and $b is $c";
exit ;;
2) echo " Option 2 Performs Subtraction" ;
    c=`expr $a - $b`;
    echo "The difference of $a and $b is $c";
exit ;;
3) echo " Option 3 Performs Multiplication" ;
    c=`expr $a \* $b`;
    echo "The product of $a and $b is $c";
exit ;;
4) echo " Option 4 Performs Division" ;
    c=`expr $a / $b`;
```

```
        echo "The Division of $a by $b is $c";exit ;;
        echo "You are exiting from the Arithmetic Manipulation" ;
    echo "Thankyou. Visit again"
    exit ;;
    esac
```

12.Write a Shell program for Palindrome Number

ALGORITHM:

STEP 1. Start the program.
STEP 2. Get the input value(a)
STEP 3. Assign value of a to d
STEP 4. Assign value of c equal to zero and b equal to one.
STEP 5. Repeat the following steps till the value of a greater than zero
STEP 6. Find b equal to a modulus 10.Find c equal to c *10 + b . Then find a equal to a divided by 10.
STEP 7. If d equal to c then print the value is Palindrome Otherwise print the number is not Palindrome
STEP 8. Stop the program.

PROGRAM:

```
    echo "Enter a number"
    read a
    d=$a
    c=0
    b=1
    while [ $a -gt 0 ]
    do
    b=`expr $a % 10`
    c=`expr $c \* 10 + $b`
    a=`expr $a / 10`
    done
    if [ $d -eq $c ]
    then
    echo "PALINDROME"
    else
    echo "NOT PALINDROME"
    fi
```

RESULT