INSTITUTE OF ENGINEERING & TECHNOLOGY
MOHANLAL SUKHADIA UNIVERSITY, UDAIPUR
(RAJ.)

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING
B. Tech - IV SEMESTER

Session 2022-23

Linux Shell Programming Lab
LABORATORY MANUAL

BT4CS10-CP04

Prepared by:
Shivam Chouhan

Content

Theory
Introduction to Linux Shell and Shell Scripting

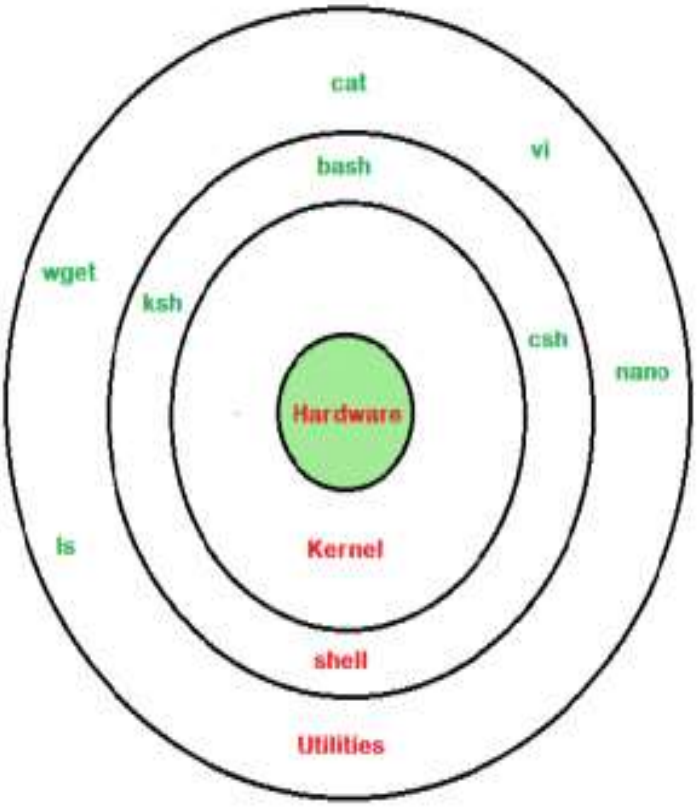
What is Kernel?

The kernel is a computer program that is the core of a computer's operating system, with complete control over everything in the system. It manages the following resources of the Linux system –

- File management
- Process management
- I/O management
- Memory management
- Device management etc.

What is Shell?

A shell is a special user program that provides an interface for the user to use operating system services. Shell accepts human-readable commands from users and converts them into something which the kernel can understand. It is a command language interpreter that executes commands read from input devices such as keyboards or from files. The shell gets started when the user logs in or starts the terminal.



Shell is broadly classified into two categories –

- Command Line Shell
- Graphical shell

Command Line Shell

Shell can be accessed by users using a command line interface. A special program called Terminal in Linux/macOS, or Command Prompt in Windows OS is provided to type in the human-readable commands such as “cat”, “ls” etc. and then it is being executed. The result is then displayed on the terminal to the user.

Graphical Shells

Graphical shells provide means for manipulating programs based on the graphical user interface (GUI), by allowing for operations such as opening, closing, moving, and resizing windows, as well as switching focus between windows.

There are several shells are available for Linux systems like –

- **BASH (Bourne Again SHell)** – It is the most widely used shell in Linux systems. It is used as default login shell in Linux systems and in macOS. It can also be installed on Windows OS.
- **CSH (C SHell)** – The C shell’s syntax and its usage are very similar to the C programming language.
- **KSH (Korn SHell)** – The Korn Shell was also the base for the POSIX Shell standard specifications etc.

Each shell does the same job but understands different commands and provides different built-in functions.

What is a terminal?

A program which is responsible for providing an interface to a user so that he/she can access the shell. It basically allows users to enter commands and see the output of those commands in a text-based interface. Large scripts that are written to automate and perform complex tasks are executed in the terminal.

Shell Scripting

Usually, shells are interactive, which means they accept commands as input from users and execute them. However, sometimes we want to execute a bunch of commands routinely, so we have to type in all commands each time in the terminal.

As a shell can also take commands as input from file, we can write these commands in a file and can execute them in shell to avoid this repetitive work. These files are called **Shell Scripts** or **Shell Programs**. Shell scripts are similar to the batch file in MS-DOS. Each shell script is saved with **`.sh`** file extension e.g., **myscript.sh.**

A shell script has syntax just like any other programming language like Python, C/C++ etc.

A shell script comprises the following elements –

- Shell Keywords – if, else, break etc.
- Shell commands – cd, ls, echo, pwd, touch etc.
- Functions
- Control flow – if..then..else, case and shell loops etc.

Why do we need shell scripts?

There are many reasons to write shell scripts:

- To avoid repetitive work and automation
- System admins use shell scripting for routine backups.
- System monitoring
- Adding new functionality to the shell etc.

Some Advantages of shell scripts

- The command and syntax are exactly the same as those directly entered in the command line, so programmers do not need to switch to entirely different syntax
- Writing shell scripts are much quicker
- Quick start
- Interactive debugging etc.

Some Disadvantages of shell scripts

- Prone to costly errors, a single mistake can change the command which might be harmful.
- Slow execution speed
- Design flaws within the language syntax or implementation
- Not well suited for large and complex task
- Provide minimal data structure unlike other scripting languages. etc.

EXPERIMENT NO. 1

Use of Basic Unix Shell Commands: ls, mkdir, rmdir, cd, cat, banner, touch, file, wc, sort, cut, grep, dd, dfspace, du, ulimit.

1. Ls : listing files

Syntax - $ls

Foo letter2

Foobar letter3

Example - $ls l* (display all files beginning with the string l)

Output – letter1 letter2 letter3

The (*) is "wildcard" character.

2. mkdir : Make Directories(use this command to create a directory)

Syntax –mkdir[directory name]

Example –mkdir easy

3.Rmdir :remove directories.(use this command to remove directory)

Syntax - $rmdir [directory name]

Example - $rmdir easy

Options:-rm –r directory name will remove all files even if directory is not empty

rmdir –r will remove directory& any present directory that are empty

rmdir –s will suppress standard error message caused by – p

4.Cd :change directories

This command is used to change the current working/present directory.

Syntax - $cd [dir name]

Example - $cd my

5.Cat :create file, display content of file, concatenate files

You can create a file without a text editor by using the cat command and ">" (Redirect output) symbol. To create a file using cat command, type

Syntax - $cat>file name

Example - $cat>colors

Red,green,blue……<CTRL_D>

To view a file we use cat in different way:

$cat colors

Output –Red,green,blue

6. banner command in linux is used to print the ASCII character string in large letter to standrad output.
Syntax:
banner text

7.The ***touch*** command is a standard command used in UNIX/Linux operating system which is used to create, change and modify timestamps of a file. Basically, there are two different commands to create a file in the Linux system which is as follows:
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.

8.Du: used disk space.

The du stands for disk usage. this command is used to show the amount of disk space consumed by one or more files or directory (or directory trees).

Syntax - $du

122 **.**/backup
425 **.**/fa/backup
2281 ./fa
5 ./check
18 ./dbf

9.Ulimit :user status

The Ulimit stand for user limit. This command shows the user space**.**

Example - $ulimit

Output – 2097151

10. wc: Print byte, word, and line counts, count the no. of bytes, whitespace-separated words, and newlines in each given FILE

Syntax - wc [*options*]... [*File*]...

Example - $cat colors

Output – 2 line 4 words 27 character colors

Options

-c	--bytes	Print only the byte counts.
	--chars	
-w	--words	Print only the word counts.
-l	--lines	Print only the newline counts

11.grep :search for a string or word in a file

Syntax - $grep option filename

Example - $grep red colors

Output – red

12. Dfspace: free disk space int MB.

The Dfspace command present in/etc directory

Example - $/etc/dfspace

Output - : Disk space: 6.30 MB available (4.59%)

Total Disk space: 6.30 MB of 135.74 MB available (4.59%)

13. Sort :sorting a file

Syntax - $sort [file name]

Example - $sort file1

Output - BcaMcaOo

15. The cut command in UNIX is a command for cutting out the sections from each line of files and writing the result to standard output. It can be used to cut parts of a line by **byte position, character and field**. Basically the cut command slices a line and extracts the text. It is necessary to specify option with command otherwise it gives error. If more than one file name is provided then data from each file is **not precedes** by its file name.

Syntax:

cut OPTION... [FILE]... The cut command in UNIX is a command for cutting out the sections from each line of files and writing the result to standard output. It can be used to cut parts of a line by **byte position, character and field**. Basically the cut command slices a line and extracts the text. It is necessary to specify option with command otherwise it gives error. If more than one file name is provided then data from each file is **not precedes** by its file name.

Experiment No:-2

Aim: Script to perform integer arithmetic operations

Script:

```
1 echo "Enter first Number"
2 read a
3 echo "Enter second Number"
4 read b
5 echo "Addition = $(($a + $b))"
6 echo "Subtraction = $(($a - $b))"
7 echo "Multiplication = $(($a * $b))"
8 echo "Division = $(($a / $b))"
9 echo "Remainder = $(($a % $b))"
```

Output:

```
ubuntu@ubuntu:~/DCCN$ ./p1.sh
Enter first Number
6
Enter second Number
2
Addition = 8
Subtraction = 4
Multiplication = 12
Division = 3
Remainder = 0
```

Experiment No:-3

Aim: Script to display first ten natural numbers

Script:

```
for((i=1;i<=10;i++))
{
echo $i
}
```

Output:

```
ubuntu@ubuntu:~/DCCN$ ./P2.sh
1
2
3
4
5
6
7
8
9
10
```

Experiment No:-4

Aim: Script to find factorial

Script:

```
1 echo "Enter the Number"
2 read num
3 fact=1
4 for((i=2;i<=num;i++))
5 {
6 fact=$((fact*i))
7 }
8 echo $fact
```

Output:

```
ubuntu@ubuntu:~/DCCN$ ./p2.sh
Enter the Number
5
120
```

Experiment No:-5

Aim: Script to check whether number is prime or not

Script:

```
echo -e "Enter the number: \c"
read n
for((i=2; i<=$n/2; i++))
do
ans=$(( n%i ))
if [ $ans -eq 0 ]
then
echo "$n is not a prime number"
exit 0
fi
done
echo "$n is a prime number"
```

Output:

```
ubuntu@ubuntu:~/DCCN$ ./p3.sh
Enter the number: 3
3 is a prime number
ubuntu@ubuntu:~/DCCN$ ./p3.sh
Enter the number: 6
6 is not a prime number
```

Experiment No:-6

Aim: Script to find sum of digits of given number

Script:

```
sum=0
echo -e "Enter the number: \c"
read n
while [ $n -gt 0 ]
do
temp=`expr $n % 10`
sum=`expr $sum + $temp`
n=`expr $n / 10`
done
echo $sum
```

Output:

```
ubuntu@ubuntu:~/DCCN$ ./P5.sh
Enter the number: 32
5
ubuntu@ubuntu:~/DCCN$ ./P5.sh
Enter the number: 65
11
```

Experiment No:-7

Aim: Script to check Armstrong number

Script:

```
echo "Enter a number: "
read c
x=$c
sum=0
r=0
n=0
while [ $x -gt 0 ]
do
r=`expr $x % 10`
n=`expr $r \* $r \* $r`
sum=`expr $sum + $n`
x=`expr $x / 10`
done
if [ $sum -eq $c ]
then
echo "It is an Armstrong Number."
else
echo "It is not an Armstrong Number."
fi
```

Output:

Enter a number:
56
It is not an Armstrong Number.

Experiment No:-8

Aim: Script to Script to calculate percentage.

Script:

```
echo -e "Enter name of student: \c"
read name
echo -e "Enter roll number of student: \c"
read roll
echo "Enter the marks of five subjects : "
read m1
read m2
read m3
read m4
read m5
let per=$m1+$m2+$m3+$m4+$m5
let per=per/5
echo "Name: $name"
echo "Roll Number: $roll"
echo "Percentage of students is $per %"
if [ $per -lt 35 ]
then
echo "Student Failed."
elif [ $m1 -ge 0 -a $m1 -lt 32 ]
then
echo "Student Failed."
elif [ $m2 -ge 0 -a $m2 -lt 32 ]
then
echo "Student Failed."
elif [ $m3 -ge 0 -a $m3 -lt 32 ]
then
echo "Student Failed."
elif [ $m4 -ge 0 -a $m4 -lt 32 ]
then
echo "Student Failed."
elif [ $m5 -ge 0 -a $m5 -lt 32 ]
then
```

```
echo "Student Failed."
elif [ $per -ge 35 -a $per -lt 45 ]
then
echo "Student passed with third class."
elif [ $per -ge 45 -a $per -lt 60 ]
then
echo "Student passed with second class."
elif [ $per -ge 60 -a $per -lt 75 ]
then
echo "Student passed with first class."
fi
```

Experiment No:-9

Aim: Script to check whether number is positive, negative or zero

Script:

```
3 echo "Enter a Number"
4 read num
5
6 if [ $num -lt 0 ]
7 then
8     echo "Negative"
9 elif [ $num -gt 0 ]
10 then
11     echo "Positive"
12 else
13     echo "Zero"
14 fi
```

Output:

Enter a Number
4
Positive

Experiment No:-10

Aim: Script to Script to check leap year

Script:

```
3 echo -e "Enter Year: \c"
4 read y
5
6 year=$y
7
8 y=$(( $y % 4 ))
9 if [ $y -eq 0 ]
10 then
11     echo "$year is Leap Year!"
12 else
13     echo "$year is not a Leap Year!"
14 fi
```

Output:

Enter Year: 2003
2003 is not a Leap Year

Experiment No:-11

Aim: Script to generate Fibonacci series

Script:

```
echo -e "Enter the number: \c"
read N
a=0
b=1
echo "The Fibonacci series is : "
for (( i=0; i<N; i++ ))
do
    echo -n "$a "
    fn=$((a + b))
    a=$b
    b=$fn
done
echo ""
```

Output:

Enter the Number: 5
The Fibonacci series is:
01123

Experiment No:-12

Aim: Script to swap two numbers

Script:

```
first=5
second=10
temp=$first
first=$second
second=$temp
echo "After swapping, numbers are:"
echo "first = $first, second = $second"
```

Output:

After swapping, numbers are:
first=10, second=5

Experiment No:-13

Aim: Script to generate pyramid using *

Script:

```
echo -e "Enter a number: \c"
read p

for((m=1; m<=p; m++))
do
    for((a=m; a<=p; a++))
    do
      echo -ne " ";
    done
    for((n=1; n<=m; n++))
    do
      echo -ne "#";
    done
for((i=1; i<m; i++))
    do
      echo -ne "#";
    done
    echo;
done
```

Output:

Enter a number:5
*
**
***
****
*****

Experiment No:-14

Aim: Script to print greatest and smallest number

Script:

```
3 echo -e "enter size of an array: \c"
4 read n
5
6 for((i=0;i<n;i++))
7 do
8 echo -e "enter $((i+1)) number: \c"
9 read nos[$i]
10 done
11 echo "number entered are"
12
13 for((i=0;i<n;i++))
14 do
15 echo ${nos[$i]}
16 done
17
18 small=${nos[0]}
19 greatest=${nos[0]}
20
21 for((i=0;i<n;i++))
22 do
23 if [ ${nos[$i]} -lt $small ]; then
24 small=${nos[$i]}
25 elif [ ${nos[$i]} -gt $greatest ]; then
26 greatest=${nos[$i]}
27 fi
28 done
29
30 echo "smallest number in an array is $small"
31 echo "greatest number in an array is $greatest"
```

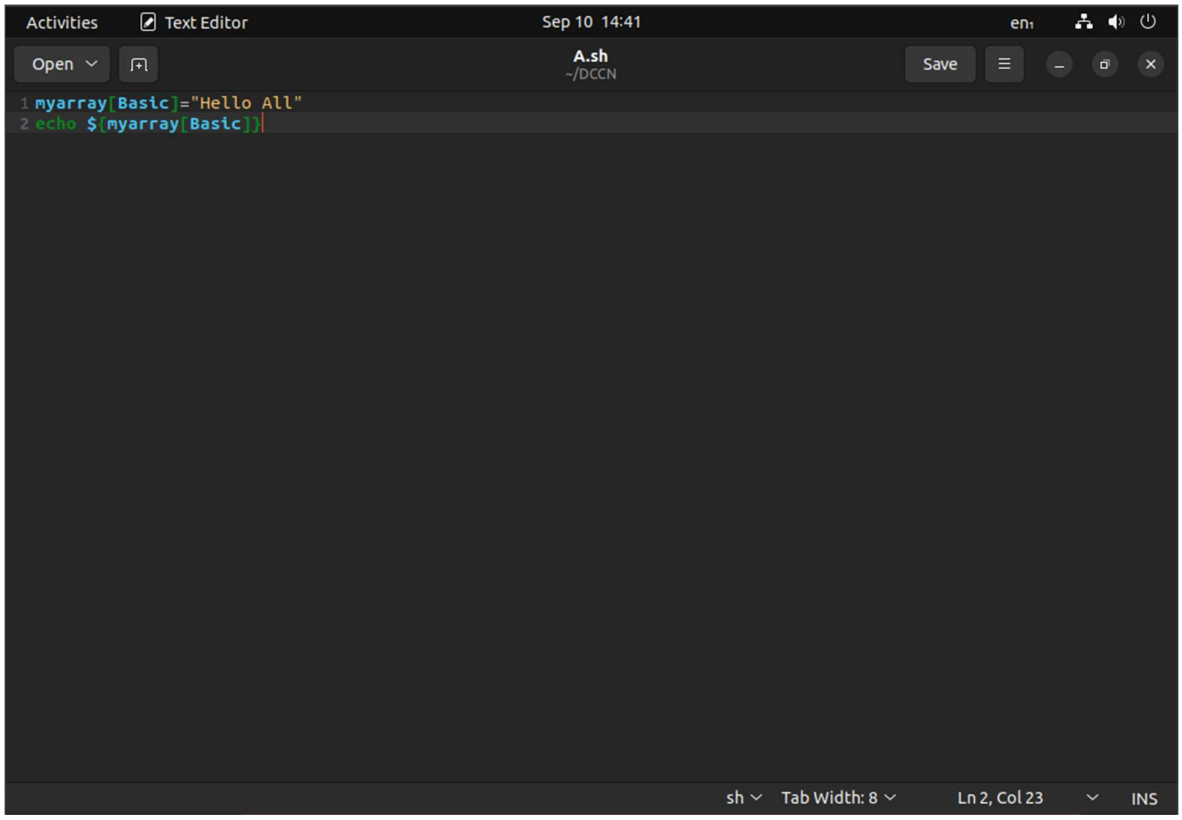
Output:

enter size of an array: 3
enter 1 number: 5
enter 2 number: 4
enter 3 number: 7
number entered are 5 4 7
smallest number in an array is 4
greatest number in an array is 7

Experiment No:-15

Aim: Introduction to arrays

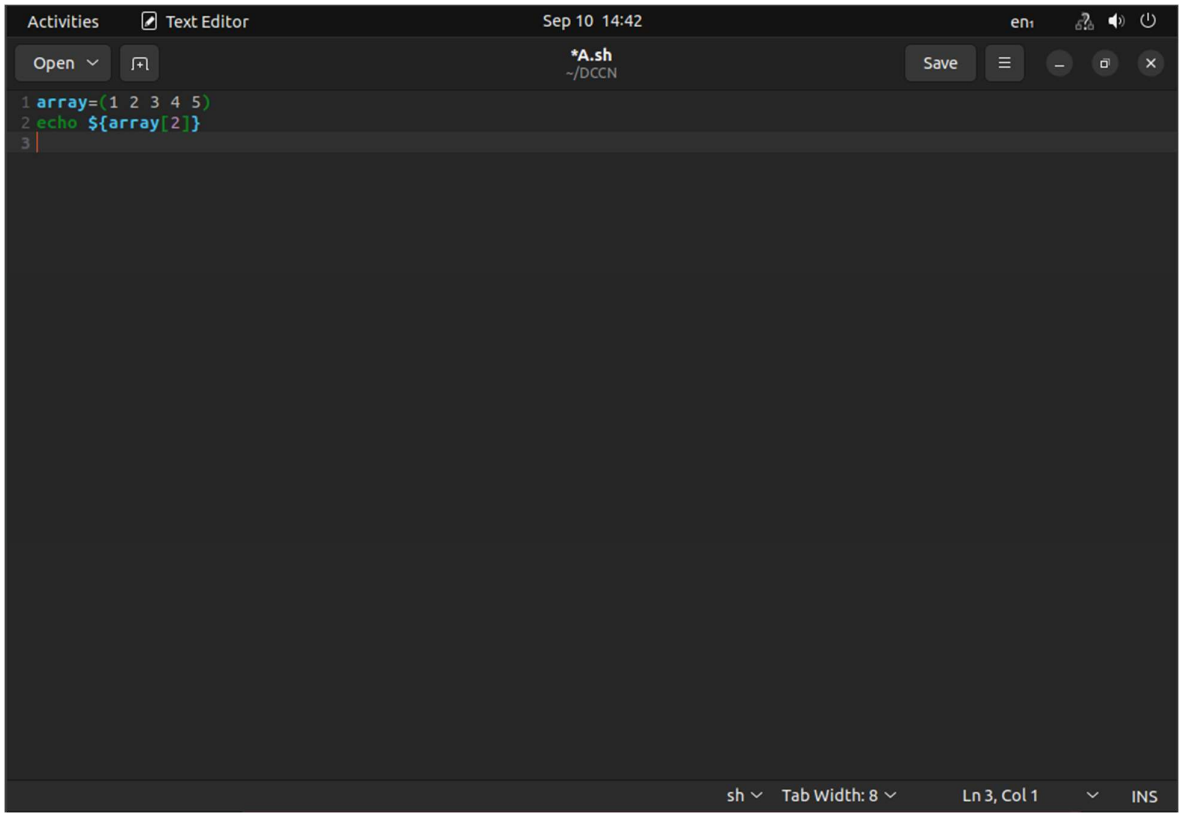
A. //Accessing array



Output:

Hello All

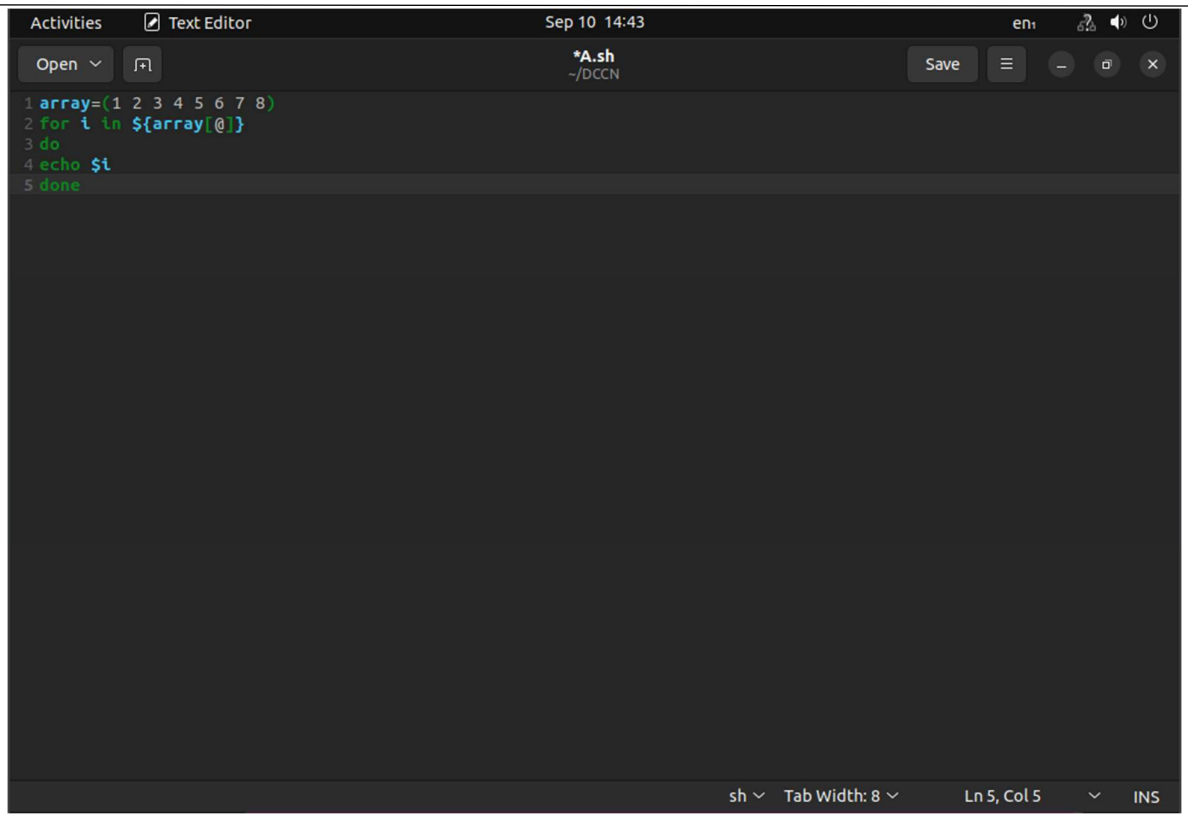
B. //Indexing array



Output :

3

C. //Reading array elements

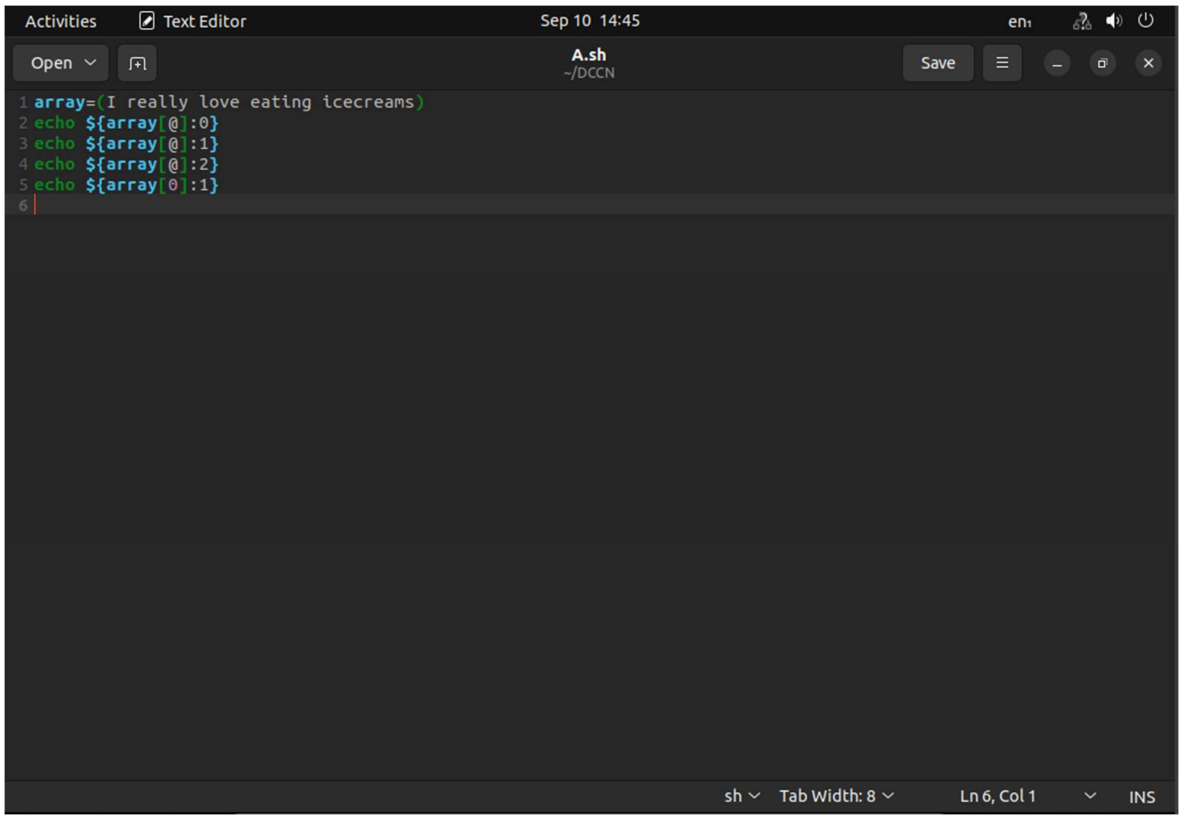


Output:

1
2
3
4
5
6
7
8

D. //Use of [@] symbol

[@] symbol is used to print all the elements at the same time. We can perform the same using [*] symbol.

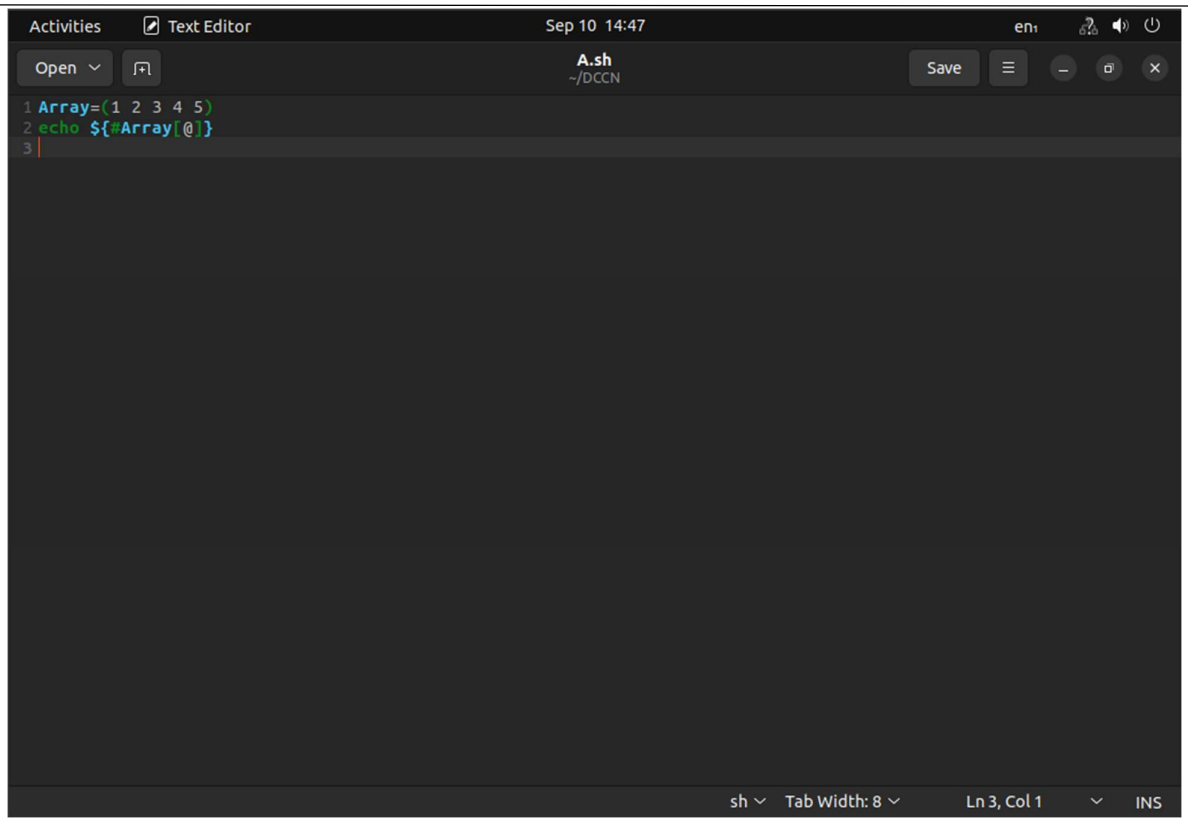


Output:

I really love eating icecream
really love eating icecream
love eating icecream
I really love eating icecream

E. //Use of [#] symbol

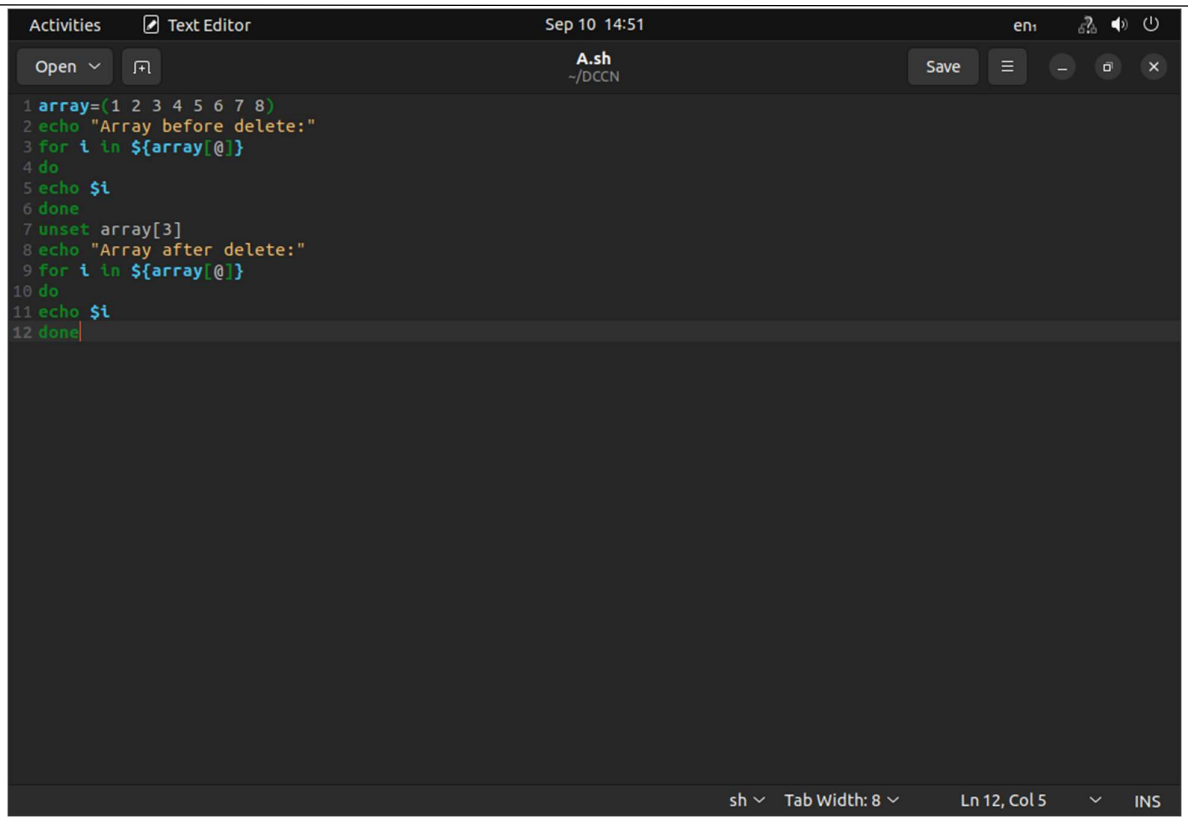
[#] symbol is used to get the count of the elements in the given array



Output:

5

F. //Use of unset



Output:

Array before delete: 1 2 3 4 5 6 7 8

Array after delete: 1 2 3 5 6 7 8

Experiment No:-16

Aim: Script to search a particular element in array

Script:

```
echo -e "Enter total no of elements: \c"
read n
i=0
echo -e "Enter the elements: \c"
while [ $i -lt $n ]
do
read a[i]
i=`expr $i + 1`
done
echo -e "Enter the element to search: \c"
read k
j=0
flag=0
while [ $j -lt $n ]
do if [ $k -eq ${a[j]} ]
then
flag=1
break
fi
j=`expr $j + 1`
done
if [ $flag -eq 1 ]
then
echo "number is found at `expr $j + 1` position"
else echo " Element is not found"
fi
```

Output:

Enter total no of elements: 3
Enter the elements: 1 2 3
Enter elements to search: 2
Number is found at 2 position

Experiment No:-17

Aim: Script to read & print elements of array

Script:

```
arr=(prakhar ankit 1 rishabh manish abhinav)

# To print all elements of array
echo ${arr[@]}
echo ${arr[*]}
echo ${arr[@]:0}
echo ${arr[*]:0}
```

Output:

```
Prakhar ankit 1 rishabh manish abhinav
Prakhar ankit 1 rishabh manish abhinav
Prakhar ankit 1 rishabh manish abhinav
Prakhar ankit 1 rishabh manish abhinav
```

Experiment No:-18

Aim: Script to find sum of all array elements

Script:

```
arr=(10 20 30 40 50)

sum=0

for i in ${arr[@]}
do
sum=`expr $sum + $i`
done

echo$sum
```

Output:

150