Lab No. 13
Using Arrays, and Functions in Shell Scripts

Objective
This lab is designed to introduce the usage of arrays and functions in shell scripting.

Activity Outcomes:
On completion of this lab students will be able to:

- Write shell scripts using array
- Write functions

Instructor Notes
As pre-lab activity, read Chapter 35 &31 from the book "The Linux Command Line", William E. Shotts, Jr.

1) Useful Concepts

Arrays
Arrays are variables that hold more than one value at a time. Arrays are organized like a table. Arrays in bash are limited to a single dimension.

Creating an Array

Array variables are named just like other bash variables, and are created automatically when they are accessed. Here is an example:

`#!/bin/bash`
`a[1]=5`
`echo "${a[1]}"`
	Out-put
	5

We can also use the declare command to declare an array. The syntax is given below

```
declare  -a   array_name
```

Adding Values to an Array
New values can be added to an array using the following syntax

```
array_name[index]=value
```

To add multiple values, we use the following syntax

```
array_name=(value1  value2 …. )
```

These values are assigned sequentially to elements of the array, starting with element zero. It is also possible to assign values to a specific element by specifying a subscript for each value:

```
array_name=([index]=value1  [index]=value2 …. )
```

Accessing Array Elements

Array elements can be accessed as follows

```
array_name=([index]=value1  [index]=value2 …. )
```

Operations on Arrays

Out-putting Entire Array: by using * or @ as index, we can output an entire array.

```
#!/bin/bash
animals=("a dog" "a cat" "a fish")
for i in ${animals[*]}
do echo $i;
done
```

Note: we can also use ${animals[@]} instead of ${animals[*]}. If we use "" marks i.e. "${animals[@]}" then contents are displayed on single line

Out-put

```
a dog
a cat
a fish
```

Determining the Number of Array Elements: we can find the total number of elements in an array by using following

```
${#array_name[@]}
```

While the length of an element can be found as

```
${#array_name[index]}
```

The following example shows the usage of these

```
#!/bin/bash
a[100]=foo
echo ${#a[@]}       # number of array elements
echo ${#a[100]}     # length of element 100
```

Out-put

```
1
3
```

Finding the Index Used by an Array: As bash allows arrays to contain "gaps" in the assignment of subscripts, it is sometimes useful to determine which elements actually exist. This can be done with a parameter expansion using the following forms:

```
${!array_name[@]}  or  ${#array_name[*]}
```

The following example shows the usage of this

```
#!/bin/bash
foo=([2]=a [4]=b [6]=c)
for i in "${!foo[@]}"
do
    echo $i
done
```

Out-put

```
2
4
6
```

Adding Elements to the End of an Array:

```
#!/bin/bash
foo=(a b c)
echo ${foo[@]}
 foo+=(d e f)
echo ${foo[@]}
```

Out-put

```
a b c

a b c d e f
```

Sorting an Array:

```
#!/bin/bash
a=(f e d c b a)
echo "Original array: ${a[@]}"
a_sorted=($(for i in "${a[@]}"; do echo
$i; done | sort))
echo "Sorted array: ${a_sorted[@]}"
```

Out-put
Original array: f e d c b a
Sorted array: a b c d e f

Deleting an Array:

```
#!/bin/bash
foo=(a b c d e f)
echo ${foo[@]}
unset foo
echo ${foo[@]}
```

Note: to delete a specific index, we can use unset 'foo[index]'

Out-put
a b c d e f

Writing Functions

A Bash function is essentially a set of commands that can be called multiple times. The purpose of a function is to help you make your bash scripts more readable and to avoid writing the same code repeatedly. Compared to most programming languages, Bash functions are somewhat limited.

The syntax for declaring a bash function is straightforward. Functions may be declared in two different formats:

```
function-name( ){
                Commands
                 }
Or
function function-name( ){
                       Commands
                      }
```

Functions can be called by name.

```
#!/bin/bash
hello_world () {
   echo 'hello, world'
}
hello_world
```

Out-put
hello world

We can define local variables within the function using the **local** keyword. To return a value, we can use return statement. Following example shows the use of return command.

```
#!/bin/bash
my_function () {
  echo "hello world"
  return 55
}

my_function
```

Out-put
hello world
55

`echo $?`	

Arguments can be passed to functions in the following way.

`#!/bin/bash`
`greeting ()`
`{`
`  echo "Hello $1"`
`}`
 	
`greeting "Ali"`
	Out-put
	Hello Ali

2) Solved Lab Activities

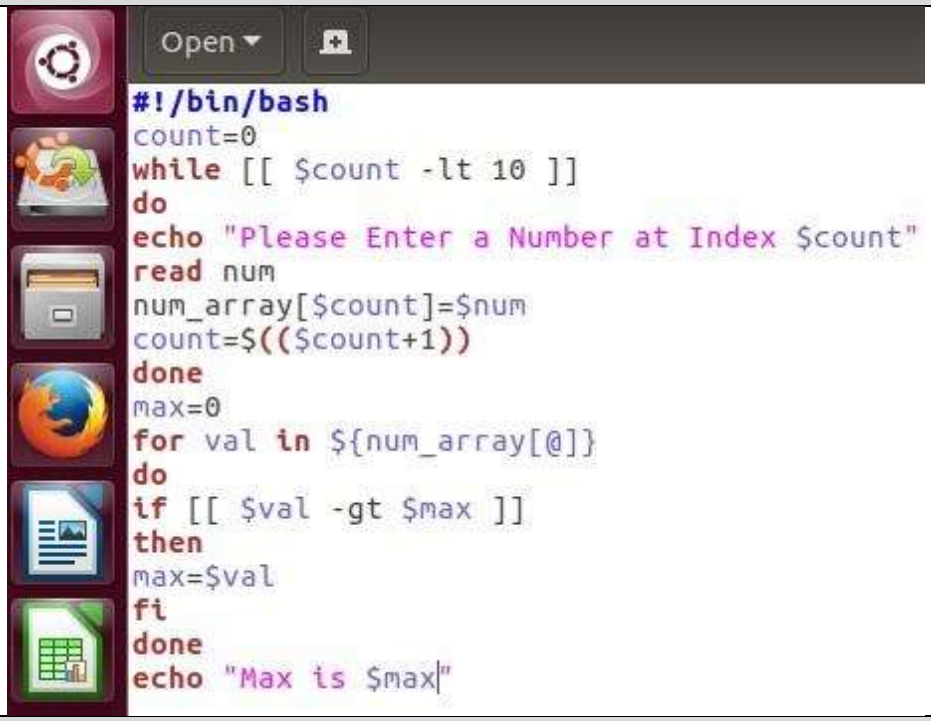
Sr.No	*Allocated Time*	*Level of Complexity*	*CLO Mapping*
1	*20*	*Medium*	*CLO-6*
2	*25*	*Medium*	*CLO-6*
3	*25*	*Medium*	*CLO-6*

Activity 1:

Write a shell script that accepts 10 numbers from user as input and finds: maximum, minimum, odd/even of them.

Solution:

Code



Out-put

```
ubuntu@ubuntu:~$ ./sh_ex
Please Enter a Number at Index 0
23
Please Enter a Number at Index 1
33
Please Enter a Number at Index 2
22
Please Enter a Number at Index 3
33
Please Enter a Number at Index 4
334
Please Enter a Number at Index 5
33
Please Enter a Number at Index 6
23
Please Enter a Number at Index 7
23
Please Enter a Number at Index 8
56
Please Enter a Number at Index 9
54
Max is 334
ubuntu@ubuntu:~$
```

Activity 2:

Write a shell script that contains two function fact( ) and is_prime( ). fact( ) function finds the factorial of a number while is_prime( ) checks whether a number is prime or not. Your script should take an integer as input from user and pass this number to fact( ) and is_prime () functions as argument.

Solution:

Code

```
#!/bin/bash
echo 'Please Enter a Number'
read num
function fact()
{
fact=1
fact_num=$1
while [[ $fact_num -gt 1 ]]
do
fact=$((fact*fact_num))
fact_num=$((fact_num-1))
done
echo "Factorial of $num is  $fact"
}

function is_prime()
{
p_num=$1
upper_limit=$((p_num/2))
count=2
ans=1

for((count=2; count<=$upper_limit; count++))
do
if [[ $((p-num%count)) -eq 0 ]]
then
ans=0
break
fi
done

if [[ $ans -eq 1 ]]
then
echo "$p_num is Prime"
else
echo "$p_num is not prime"
fi
}
fact $num
is_prime $num
```

Out-put

```
ubuntu@ubuntu:~$ ./sh_ex
Please Enter a Number
7
Factorial of 7 is  5040
7 is Prime
ubuntu@ubuntu:~$ ./sh_ex
Please Enter a Number
8
Factorial of 8 is  40320
8 is not prime
ubuntu@ubuntu:~$
```

Activity 3:

Write a shell script that contains a function play_game. This function generates a random number between 1 and 10; and keeps on asking the user to guess the number as long as user enters a number which is equal to the random number. In the end, the total number of attempts made by the user to enter correct guess is displayed.

Soltuion:

Code

```
#!/bin/bash
function play_game()
{
count=0
r_number=$((RANDOM%10))
while [[ $num -ne $r_number ]]
do
echo "Please Enter a Number Between 1 and 10"
read num
count=$((count+1))
done
echo "Congratulations ! You have Guessed Correctly in $count
}
play_game
```

Out-put

```
ubuntu@ubuntu: ~
ubuntu@ubuntu:~$ ./sh_ex
Please Enter a Number Between 1 and 10
3
Please Enter a Number Between 1 and 10
4
Please Enter a Number Between 1 and 10
5
Congratulations ! You have Guessed Correctly in 3 Attempts
ubuntu@ubuntu:~$
```

3) Graded Lab Tasks

Note: The instructor can design graded lab activities according to the level of difficult and complexity of the solved lab activities. The lab tasks assigned by the instructor should be evaluated in the same lab.

Task 1:

Write a function that finds the sum of digits in an integer

Task 2:

Write a function that finds whether an integer is a palindrome or not

Task 3:

Write a function that writes an integer in reverse (for example writes 123 as 321)