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:

	Out-put

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

	Out-put

`#!/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
	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

	Out-put

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

	Out-put

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

Adding Elements to the End of an Array:

	Out-put

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