Lab 2: Shell programming with bash

The shell is a program that provides the user with an interface to use the operating system's functions through some commands. A shell script is a program that is used to perform specific tasks. These scripts are mostly used to avoid repetitive work. You can write a script to automate a set of instructions to be executed one after the other, instead of typing in the commands one after the other n number of times.

If you are familiar with a programming language like Python or C, then it should be pretty easy to get started with shell scripting.

1 Creating a Shell Script

To create a bash script, place #!/bin/bash at the top of the file. Then, change the permissions on the file to make it executable using

```
$ chmod u+x scriptname
```

To execute the script from the current directory, run ./scriptname and pass any parameters you wish.

Below is an example bash script that prints ”Hello World”

```
#!/bin/bash
echo "Hello World"
```

Note: You can use Gedit or some other text editor in Linux and Mac for writing the code. In windows, you can use Notepad or VIM editor.

Save the above file with the extension “.sh”. Run the file with the command “./scriptname.sh”.

Note that

1. Comments in shell script start with #. It can be placed anywhere in a line; the shell ignores contents to its right. Comments are recommended but not mandatory

2. Shell variables are loosely typed i.e. not declared. Their type depends on the value assigned. Variables when used in an expression or output must be prefixed by $.

3. Output is displayed using echo statement. Any text should be within quotes.

1.1 Passing parameters to shell script

While running a command, the user can pass a variable number of parameters in the command line.

Within the command script, the passed parameters are accessible using ‘positional parameters’. These range from $0 to $9, where $0 refers to the name of the command itself, and $1 to $9 are the first through to the ninth parameter, depending on how many parameters were actually passed.

Example:

$ sh hello how to do you do

Here $0 would be assigned sh

$1 would be assigned hello

$2 would be assigned how

And so on . . .

2 Variables and Arithmetic Expressions

You can set a variable using the following command.

```
var1=7  # sets var1 to 7
var2 = "Hello World"
```

Note that there are no spaces before and after “=”.

Try creating a file with a given variable name

2.1 Adding two numbers

Two numbers can be added using an expression containing the ‘+’ command. Arithmetic operators are + - * /.
The following program shows how two numbers can be added.

```
#!/bin/bash
echo -n  Enter  the 1st  number
read n1
echo -n  Enter  the 2nd  number
read n2
sum=$(($num1 + $num2))
echo  The  sum is =  $sum
```

Exercise 1: Write a program to find the area and circumference of a circle given its radius.
Exercise 2: Write a Shell program to swap two numbers.

2.2 Concatenating Strings

The simplest way to concatenate two or more string variables is to write them one after another. Run the following commands on your shell to understand this.

```
VAR1="Hello,"
VAR2=" World"
VAR3=100
VAR4="$VAR1$VAR2$VAR3"
echo "$VAR4"
```

3 Conditional Statements

Shell supports decision-making using if statement. The if statement has the following formats. The first construct executes the statements when the condition is true. The second construct adds an optional else to the first one that has different set of statements to be executed depending on whether the condition is true or false. The last one is an elif ladder, in which conditions are tested in sequence, but only one set of statements is executed.

- Type I

```
#!/bin/bash
if [ condition ]
then
statements
fi
```

- Type II

```
#!/bin/bash
if [ condition ]
then
statements
else
```

```
statements
fi
```

- Type III

```
#!/bin/bash
if [condition ]
then
statements
elif [ condition ]
then
statements
.. . else
statements
fi
```

Operator Description

1. -eq : Equal to
2. -ne : Not equal to
3. -gt : Greater than
4. -ge : Greater than or equal to
5. -lt : Less than
6. -le : Less than or equal to
7. -a : Logical AND
8. -o : Logical OR
9. ! : Logical NOT

The following program shows how to find the smallest of 3 numbers.

```
#!/bin/bash
echo Enter 3 numbers with spaces in between
read a b c
s=$a
if [ $b -lt $s ]
then
s=$b
fi
if [ $c -lt $s ]
then
s=$c
fi
echo Smallest of $a $b $c is $s
```

Exercise 3: Write a shell program to check whether the given number is odd or even
Exercise 4: Write a program to determine whether the given year is leap year or not.

4 While Loop

One of the easiest loops to work with is while loops. They say, while an expression is true, keep executing these lines of code. They have the following format:

```
while [ <some test> ]
do
<commands>
done
```

Following is a program to print numbers from 1 to 10.

```
#!/bin/bash
# Basic while loop
counter=1
while [ $counter -le 10 ]
do
echo $counter
((counter++))
done
echo All done
```

Exercise 5: Write a shell program to determine whether the given number is prime or not.
Exercise 6: Write a shell program to create 20 files named file1.txt, file2.txt, ... , file20.txt. Write another program to delete all these files.
Exercise 7: Write a shell program to compile and run all the C files in a given folder. Hint: Use ”find”, ”gcc” and ”./a.out” commands.

The following program shows how to read a file in terminal using the while loop.

```
#!/bin/bash
read -p "Enter file name : " filename
while read line
do
echo $line
done < $filename
```

Exercise 8: Write a shell program to check whether the content of two files is exactly the same.

5 References

1. https://matt.might.net/
2. https://tecadmin.net/
3. https://www.vvitengineering.com/
4. codepoc.io/blog/unix/3776/
5. https://ryanstutorials.net/bash-scripting-tutorial/