Topics to cover:

1. Introduction to Shell Scripting
2. Programming practice

1. Shell Scripting

1.1 What is 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 a 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. If you have any prior experience with any programming language like Python, C/C++ etc. It would be very easy to get started with it.

1.2 Examples

The shell is, after all, a real programming language, complete with variables, control structures, and so forth. No matter how complicated a script gets, it is still just a list of commands executed sequentially.

The following script uses the **read** command which takes the input from the keyboard and assigns it as the value of the variable PERSON and finally prints it on STDOUT.

```
#!/bin/sh

echo "What is your name?"

read PERSON

echo "Hello, $PERSON"
```

1.3 Declaring variables in the script

```
#!/bin.sh

myVar=Hello
name=Sitare

echo $myVar $name
```

Output:

```
Hello Sitare
```

1.4 Adding additional delay in the script

Sometimes we want to add additional time in the script. For example, if we want a script to run after 5 seconds, we can type the following.

```
#!/bin/sh
sleep 5
echo "Sitare"
```

The above script will print “Sitare” after 5 minutes from running the script. If we want to add a delay of 1 minute, we can simplye type “sleep 60”. For 1 hour, we can type “sleep 3600”. Note that this command usually runs with seconds as the unit of time measurement. In order to explicitly specify the number of hours or minutes, you can simply type “sleep 1h” for 1 hour and “sleep 15m” for 15 minutes, respectively.

1.5 Calculating runtime of the script

We need to create a date variable to find out the amount of time required for running a shell script. Following example shows this.

```
#!/bin.sh

start=$(date +%s)
myVar=Hello
name=Sitare

sleep 1
echo $myVar $name

end=$(date +%s)

echo "Time required is: $((end-start)) seconds"
```

Output:

```
Hello Sitare
Time required is: 5 seconds
```

Note that in the above example, we intentionally added a delay of 5 minutes to show the runtime. Otherwise, the script would show 0 seconds in the output as it runs very fast for such simple scripts. This will take some amount of time for larger scripts or the scripts involving time consuming tasks. However, you can also use the following command in the script to see the run time in subseconds.

For example, if we type “date +%s%N” instead of “date +%s”, this will show time in nano seconds. Now, as nano is 10^{-9} and milli 10^{-3}, we can use the three first characters of nanoseconds to get the milliseconds. Therefore, we can type “%s%3N” instead of “%s%N” to find runtime in miliseconds. Similarly, we can use %6N for microseconds.

1.6 Working with if-else statements

When trying to understand the working of a function like if-else in a shell script, it is good to start things simple. Here, we initialize two variables a and b, then use the if-else function to check if the two variables are equal. The script should look as follows for this task.

```
#!/bin/sh

m=1
n=2

if [ $n -eq $m ]
then
        echo "Both variables are the same"
else
        echo "Both variables are different"
fi
```

Output:

```
Both variables are different
```

Similar to equality checking, we can use “-gt”, “-lt” for checking greater or less than conditions.

1.7 Working with loops

1.7.1 While loop

Here the command is evaluated and based on that the resulting loop will execute, if the command is raised to false then the loop will be terminated.

Syntax:

```
#/bin/sh
while <condition>
do
   <command 1>
   <command 2>
   <etc>
done
```

Example:

```
#/bin/sh

a=0

# lt is less than operator

#Iterate the loop until a less than 10

while [ $a -lt 10 ]
do
 # Print the values
 echo $a
 # increment the value
 a=`expr $a + 1`
done
```

Output:

```
0
1
2
```

```
3
4
5
6
7
8
9
```

As you can see, the above script will print the numbers from 0 to 9.

1.7.2 For loop

The for loop operates on lists of items. It repeats a set of commands for every item in a list. Int the s,yntax below var is the name of a variable and word1 to wordN are sequences of characters separated by spaces (words). Each time the for loop executes, the value of the variable var is set to the next word in the list of words, word1 to wordN.

Syntax:

```
#/bin/sh
for <var> in <value1 value2 ... valuen>
do
  <command 1>
  <command 2>
  <etc>
done
```

Example:

```
#!/bin/sh


for var in 0 1 2 3 4 5 6 7 8 9

do

  echo $var
```

```
done
```

Output:

```
0
1
2
3
4
5
6
7
8
9
```

As you can see, the above script will print the numbers from 0 to 9.

There is also another way of printing the contents using for loop.

```
#!/bin/sh


max=10

for i in `seq 1 $max`

do

        echo "$i"

done
```

Output:

```
1
2
3
```

```
4
5
6
7
8
9
10
```

We can also print all the file names in a directory using a for loop. For example, the following script will print all file names in the directory called “/home/vboxuser/AJP/code/java/”

```
#!/bin/sh


for FILE in /home/vboxuser/AJP/code/java/*
do
  echo "File or folder name with path: $FILE"
  echo "Only file or folder name: $(basename $FILE)"
done
```

Note that $FILE will print the file names with full path and if you use “basename” as well, it will remove the path and print only the file names. This will also print the folder names.

Output:

```
File name with path: /home/vboxuser/AJP/code/java/AreaShape.java
Only file name: AreaShape.java
File name with path: /home/vboxuser/AJP/code/java/FileReader.java
Only file name: FileReader.java
File name with path: /home/vboxuser/AJP/code/java/FileReaderCommandLine.java
Only file name: FileReaderCommandLine.java
File name with path: /home/vboxuser/AJP/code/java/Inheritance.java
Only file name: Inheritance.java
File name with path: /home/vboxuser/AJP/code/java/NumberConversion.java
Only file name: NumberConversion.java
File name with path: /home/vboxuser/AJP/code/java/Polymorphism.java
Only file name: Polymorphism.java
File name with path: /home/vboxuser/AJP/code/java/TestClass.java
Only file name: TestClass.java
File name with path: /home/vboxuser/AJP/code/java/Testing.java
```

Only file name: Testing.java
File name with path: /home/vboxuser/AJP/code/java/package_one
Only file name: package_one
File name with path: /home/vboxuser/AJP/code/java/package_two
Only file name: package_two

*****Note:** There are many many more things we can do with shell scripts. We have covered only a very small part of them. You can explore further according to your requirements.