Lab 2 - Bash Scripting and Automations

What is Bash and Bash Scripting

Bash (Bourne Again Shell) is a command language interpreter that we use in order to interact with the linux kernel. It is a command line interpreter that we use in order to interact with the linux kernel.

Bash scripting is the process of writing a set of commands in a file and then executing them in order to automate a task.

Introduction to Text Editors

In Linux, we have multiple commands that act as the `notepad` of linux. Such softwares include (but are not limited to):

- nano
- vim
- emacs

In this course, we'll be studying nano.

Nano

Nano is a command line text editor that we can use to write, read and delete data from within a file.

In order to open a file in nano, we type the following command:

```
nano <file-name>
```

Now, in order to save data into the file, we will firstly press `CTRL+S` and then, in order to exit, we need to press `CTRL+X`.

Writing your first bash script

The first line in a bash script must be `#!/bin/bash` and is called as `SHEBANG` line.

> A she-bang is set of sequence that begins with #! and then the interpreter is specified. In our case, we’ll be using /bin/bash as the interpreter.

Then, we will use `echo` command in order to print data into the stdout.

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

Now, in order to execute this file, we need to give it executable permissions. We can do that by using the `chmod` command.

```
chmod +x <file-name>
```

Now, we can execute the file by using the following command:

```
./<file-name>
```

Variables

Variables are used to store data in a program. In bash, we can declare a variable by using the following syntax:

```
# NOTE: There should be no space between the variable name and the equal sign
variable_name=value
```

Now, in order to access the value of the variable, we need to use the `$` sign before the variable name.

```
echo $variable_name
```

Variable Types

There are two types of variables in bash:

- System Variables

- User Defined Variables

System Variables

System variables are the variables that are defined by the system and are used to store system related information.

Some of the system variables are:

- `$HOME` : Stores the path to the home directory of the user
- `$PWD` : Stores the path to the current working directory
- `$BASH` : Stores the path to the bash shell
- `$BASH_VERSION` : Stores the version of the bash shell
- `$LOGNAME` : Stores the name of the user

User Defined Variables

User defined variables are the variables that are defined by the user and are used to store user related information.

Read Input from the User

In order to read input from the user, we can use the `read` command.

```
read <variable-name>
## If we want a message to be displayed before the user enters the value, we can use t
he following syntax:
read -p "Enter your name: " <variable-name>
```

Now, the value that the user enters will be stored in the variable.

Unsetting a Variable

In order to unset a variable, we can use the `unset` command.

```
unset <variable-name>
```

If-Else Statements

In order to use if-else statements in bash, we can use the following syntax:

```
if [ <condition> ]
then
  <statements>
else
  <statements>
fi
```

For Loop

In order to use for loop in bash, we can use the following syntax:

```
for <variable-name> in <list>
do
  <statements>
done
```

While Loop

In order to use while loop in bash, we can use the following syntax:

```
while [ <condition> ]
do
  <statements>
done
```

Arguments

Arguments are the values that are passed to the script when it is executed.

In order to access the arguments, we can use the following syntax:

```
$0 # Stores the name of the script
$1 # Stores the first argument
$2 # Stores the second argument
$n # Stores the nth argument
```

Exit Status

Exit status is the status that is returned by the script when it is executed.

In order to access the exit status, we can use the following syntax:

```
$? # Stores the exit status
```

Functions

Functions are the set of statements that are executed when they are called.

In order to define a function, we can use the following syntax:

```
function_name() {
  <statements>
}
```

In order to call a function, we can use the following syntax:

```
function_name
```

Class Task

Task 1

Write a bash script that takes a number as an argument and prints whether the number is even or odd. The output should be “True” or “False”. Case matters. The file must be inside `/tmp/` directory and named as `even-odd.sh`.

Task 2

Using a for loop in bash, try and ping the subnet “172.16.0.0/24” and print the IP addresses that are up. Output should be like: “172.16.0.0 = UP”

> Hint: Use ping -c 1 <ip-address> to ping the IP address once.

The file must be inside `/tmp/` directory and named as `ping.sh`.

Task 3

Create a function called `create_user` that takes two arguments: username and password. The function should create a user with the given username and password. Also, write another function called `add_to_group` that takes two arguments: username and groupname. The function should add the user to the given group. The file must be inside `/tmp/` directory and named as `user.sh`. The username, password and groupname should be provided from the command line as arguments to the script.
