Bash script

Week	Week6
Reference	https://www.freecodecamp.org/news/bash-scripting-tutorial-linux-shell-script-and-command-line-for-beginners/
Student	7ala hazeem
# Student ID	21028137

Definition of Bash scripting

A bash script is a file containing a sequence of commands that are executed by the bash program line by line.

The term "shell" refers to a program that provides a command-line interface for interacting with an operating system. Bash (Bourne-Again SHell) is one of the most commonly used Unix/Linux shells and is the default shell in many Linux distributions.

How to Create and Execute Bash scripts

Script naming conventions

By naming convention, bash scripts end with `.sh`. However, bash scripts can run perfectly fine without the `sh` extension.

Adding the Shebang

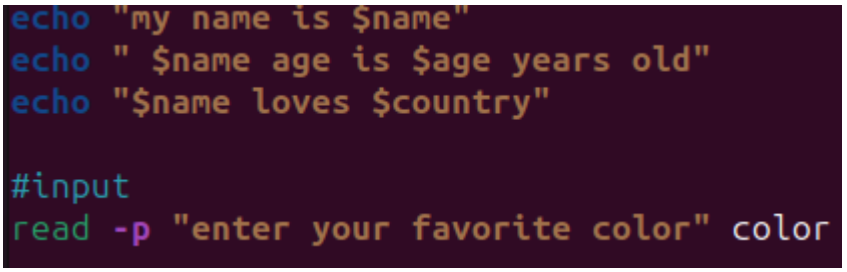
This is the first line of the script. Shebang tells the shell to execute it via bash shell. Shebang is simply an absolute path to the bash interpreter.

```
#!/bin/bash
```

Creating our first bash script

The `echo` command means to display on terminal

The `read` command reads the input and stores it in a variable



Executing the bash script

To make the script executable, assign execution rights to your user using this command:

```
chmod u+x run_all.sh
```

or

```
chmod u+x bash.sh/the file path
```

Comments in bash scripting

Comments start with a `#` in bash scripting.

Comments are very helpful in documenting the code, and it is a good practice to add them to help others understand the code.

These are examples of comments:

```
# This is an example comment
# Both of these lines will be ignored by the interpreter
```

Variables and data types in Bash

In Bash, a variable is capable of storing numeric values, individual characters, or strings of characters.

you can use and set the variable values in the following ways:

1. Assign the value directly:

```
country="Jordan"
```

2. Assign the value based on the output obtained from a program or command, using command substitution.

```
same_country=$country
```

- To access the variable value, add `$` to the variable name.

Conditional statements (if/else)

There are several ways to evaluate conditions, including `if`, `if-else`, `if-elif-else`, and nested conditionals.

```
if [[ condition ]];
then
  statement
elif [[ condition ]]; then
  statement
else
  do this by default
fi
```

- We can use logical operators such as **AND** `-a` and **OR** `-o` to make comparisons

integer comparison:

▼ eq

```
is equal to
if [ "$a" -eq "$b" ]
```

▼ ne

```
is not equal to
if [ "$a" -ne "$b" ]
```

▼ gt

```
is greater than
if [ "$a" -gt "$b" ]
```

▼ ge

```
is greater than or equal to
if [ "$a" -ge "$b" ]
```

▼ lt

```
is less than
if [ "$a" -lt "$b" ]
```

▼ le

```
is less than or equal to
if [ "$a" -le "$b" ]
```

▼ <

```
is less than (within double parentheses)
(("$a" < "$b"))
```

▼ <=

```
is less than or equal to (within double parentheses)
(("$a" <= "$b"))
```

▼ >

```
is greater than (within double parentheses)
(("$a" > "$b"))
```

▼ >=

```
is greater than or equal to (within double parentheses)
```

```
(("$a" >= "$b"))
```

string comparison

▼ =

```
is equal to
if [ "$a" = "$b" ]
```

▼ !=

```
is not equal to
if [ "$a" != "$b" ]
```

▼ example of a Bash script that uses `if`, `if-else`, and `if-elif-else` statements to determine if a user-inputted number is positive, negative, or zero:

```
#!/bin/bash
echo "Please enter a number: "
read num

if [ $num -gt 0 ]; then
  echo "$num is positive"
elif [ $num -lt 0 ]; then
  echo "$num is negative"
else
  echo "$num is zero"
fi
```