****

**Contents**

1. **Introduction to bash scripts**
2. **Variables**

* **Length of variables**

1. **Operators**
2. **Control statements**
3. **Functions**

* **Passing arguments**

1. **User Interface**

**1.Bash scripting:**

Shell scripting is an powerful way to automate tasks that you regularly execute on your computer.

**Basics:**

Scripts are stored in files. You can give any name and extension to a shell script, it does not matter. The important thing is that it must start with a “shebang” on the first line:



and it must be an executable file.

A file is set as executable using chmod, an utility command.

You can use it like this



to make the myscript file executable for your user. Now you can execute the script if you are in the same folder by calling it ./myscript, or using the full path to it.

**Comments:**

Comments are one of the most important things when writing programs. A line starting with the # symbol is a comment

**#!/bin/bash**

# this is a comment

A comment can start at the end of a line too:

**#!/bin/bash**

echo ok # this is a comment

**2.Variables:**

You can set variables by using the = operator

name=value

**Examples:**

ROOM\_NUMBER=237

name=Flavio

nickname="Flavio"

## **Length of variables**:

## 

**Output:**

## 

## **3.Operators:**

## Bash implements some arithmetic operators commonly used across programming languages:

* Add (+)
* Subtract(-)
* Multiply(\*)
* Divide(/)

You can also use these comparison operators:

* lower than(-lt)
* greater than(-gt)
* lower or equal than(-le)
* greater or equal than(-ge)
* equal to(-eq)
* not equal to(-ne)

**Example:**

**#!/bin/bash**

age=23

minimum=18

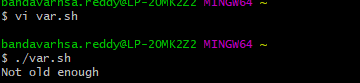
if test $age -ge $minimum

then

echo "Not old enough"

fi

**Output:**



You can print anything to the screen using the **echo** command:

**#!/bin/bash**

echo "test"

## 4.Control structures:

## In you can use several control structures which you might be familiar with:

**If/else statements:**

**Simpleif:**

if condition

then

command

fi

**if then else :**

if condition

then

command

else

anothercommand

fi

**Nested if - then - else:**

if condition

then

command

elif

anothercommand

else

yetanothercommand

fi

You can keep the else on the same line by using a semicolon:

if condition ; then

command

fi

**Example:**

**#!/bin/bash**

DOGNAME=Roger

if [ "$DOGNAME" == "" ]; then

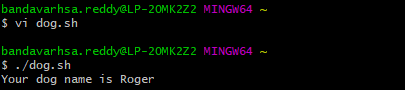
echo "Not a valid dog name!"

else

echo "Your dog name is $DOGNAME"

fi

#### **Output**:



#### **While loop:**

While condition resolves to a true value, run command

while condition

do

command

done

#### **Until:**

Until condition resolves to a true value, run command

until condition

do

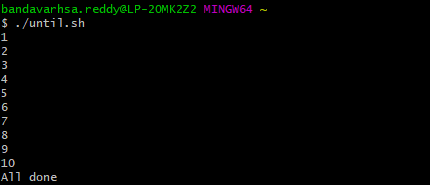
command

Exadone

#### **Example:**

#### 

#### **Output:**



#### **For in:**

Iterate a list and execute a command for each item

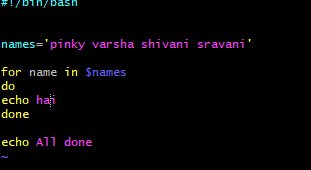
for item in list

do

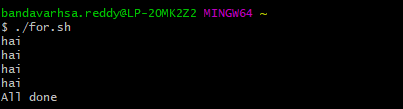
command

done

### **Example:**



### **Output:**



### **Case:**

### A case control structure lets you pick different routes depending on a value.

case value in

a)

command

#...

;;

b)

command

#...

;;

esac

Like with fi, esac ends the case structure and as you can notice it’s case spelled backwards.We add a double semicolon after each case.

**Example:**

**#!/bin/bash**

read -p "How many shoes do you have?" value

case $value in

0|1)

echo "Not enough shoes! You can't walk"

;;

2)

echo "Awesome! Go walk!"

#...

;;

\*)

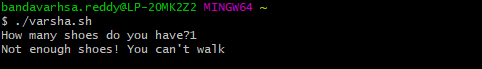
echo "You got more shoes than you need"

#...

;;

esac

### **Output:**



### **Select:**

A select structure shows the user a menu of choices to choose:

select item in list

do

command

done

**Example:**

**#!/bin/bash**

select breed in husky setter "border collie" chiwawa STOP

do

if [ "$breed" == "" ]; then

echo Please choose one;

else

break

fi

done

## Output:

## 

## Testing conditions:

I used the term condition in the above control structures. You can use the test Bash built-in command to check for a condition and return a true (0) or false value (not 0).

**Example:**

**#!/bin/bash**

if test "apples" == "apples"

then

echo Apples are apples

fi

if ! test "apples" == "oranges"

then

echo Apples are not oranges

fi

## Output:

## 

## **5.Functions:**

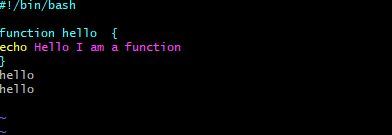
Just like in JavaScript or in any other programming language, you can create little reusable pieces of code, give them a name, and call them when you need.Bash calls them functions.

You define a function with

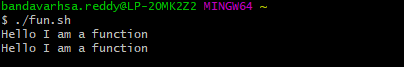
function name {

}

**Example:**



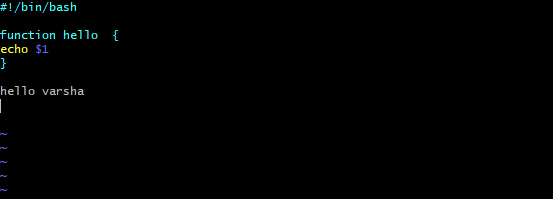
Output:



### **Passing Arguments:**

It is often the case that we would like the function to process some data for us. We may send data to the function in a similar way to passing command line arguments to a script. We supply the arguments directly after the function name. Within the function they are accessible as **$1, $2, etc**.

**Input:**

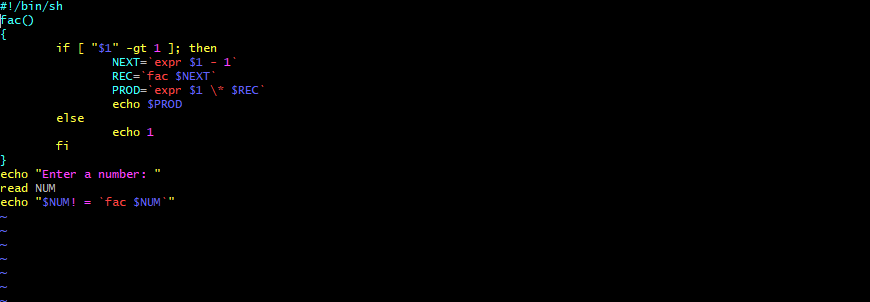


**Output:**

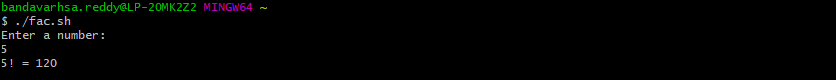


**Example:**

**Input:**



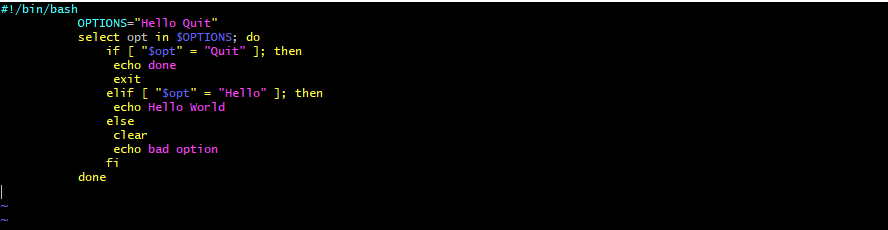
**Output:**



**6.User Interface:**

For Bash scripts I like to think about the layout and structure of the commands inside the script as well. Bash scripts are often small tools used to automate tedious and repetitive tasks. They are always readable by the end user and often modified to suit changing requirements. Therefore the ease with which the user (often yourself) may modify and extend the script is also very important.

**Input:**



**Output:**

