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:

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

and it must be an executable file.

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

You can use it like this

```
chmod u+x myscript
```

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
```

Variables:

You can set variables by using the = operator

```
name=value
```

Examples:

```
ROOM_NUMBER=237
name=Flavio
nickname="Flavio"
```

You can print a variable by using the **echo** built-in command and prepending a **$to the var name**:

```
echo $ROOM_NUMBER
echo $name
```

Length of variables:

```
#!/bin/bash
# Show the length of a variable.

a='varsha'
echo ${#a} |

b=123456
echo ${#b}
```

Output:

```
bandavarhsa.reddy@LP-2OMK2Z2 MINGW64 ~
$ ./len.sh
6
6
```

Operators:

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

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

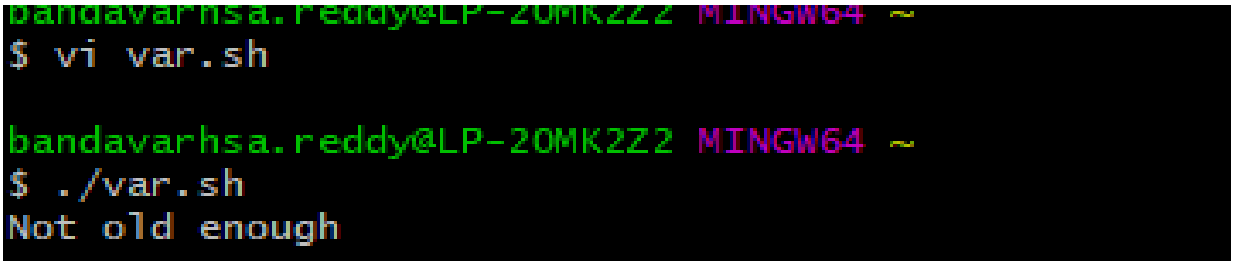
You can also use these comparison operators:

- 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"
```

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*:*

```
bandavarhsa.reddy@LP-20MK2Z2 MINGW64 ~
$ vi dog.sh

bandavarhsa.reddy@LP-20MK2Z2 MINGW64 ~
$ ./dog.sh
Your dog name is Roger
```

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:

```
#!/bin/bash

counter=1
until [ $counter -gt 10 ]
do
echo $counter
((counter++))
done

echo All done
~
~
~
```

Output:

```
bandavarhsa.reddy@LP-20MK2Z2 MINGW64 ~
$ ./until.sh
1
2
3
4
5
6
7
8
9
10
All done
```

For in:

Iterate a list and execute a command for each item

```
for item in list
do
```

```
  command
done
```

Example:

```
#!/bin/bash

names='pinky varsha shivani sravani'

for name in $names
do
echo hai
done

echo All done
~
```

Output:

```
bandavarhsa.reddy@LP-20MK2Z2 MINGW64 ~
$ ./for.sh
hai
hai
hai
hai
All done
```

Case:

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

```
case value in
  a)
    command
    #...
    ;;
```

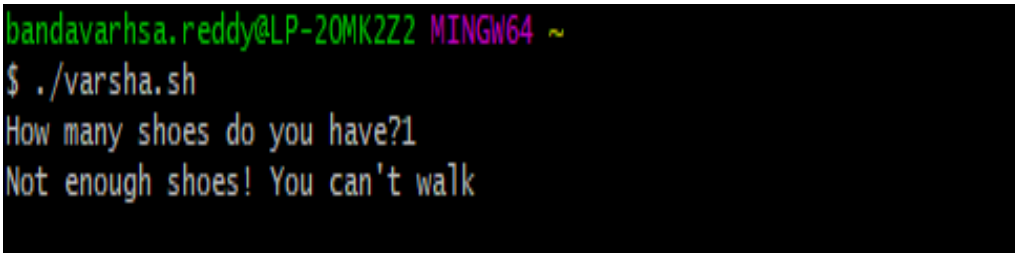
```
  b)
    command
    #...
    ;;
esac
```

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:

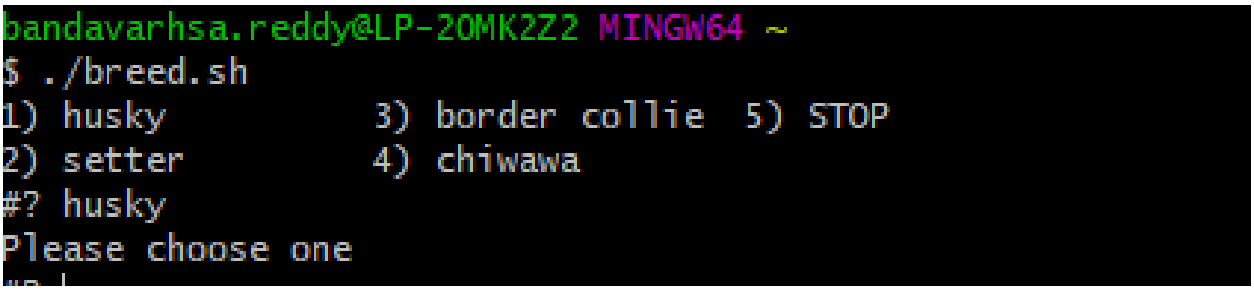
```
select item in list
do
  command
done
```

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:

```
bandavarhsa.reddy@LP-20MK2Z2 MINGW64 ~
$ ./apple.sh
Apples are apples
Apples are not oranges
```

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:

```
#!/bin/bash

function hello  {
echo Hello I am a function
}
hello
hello

~
~
```

Output:

```
bandavarhsa.reddy@LP-20MK2Z2 MINGW64 ~
$ ./fun.sh
Hello I am a function
Hello I am a function
```