What Is "The Shell"?

Simply put, the shell is a program that takes commands from the keyboard and gives them to the operating system to perform. In the old days, it was the only user interface available on a Unix-like system such as Linux. Nowadays, we have *graphical user interfaces (GUIs)* in addition to command line interfaces (CLIs) such as the shell.
On most Linux systems a program called bash (which stands for Bourne Again SHell, an enhanced version of the original Unix shell program, sh, written by Steve Bourne) acts as the shell program.

What's A "Terminal?"

It's a program called a *terminal emulator*. This is a program that opens a window and lets you interact with the shell.

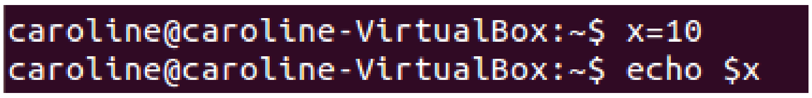
Shell Programming

- A series of shell commands can be stored in text files to form a shell program, often called a **script**

These facilities that a script provides (variables, decision control – if/then/else, loops etc) can be entered interactively at the command line as well as in scripts

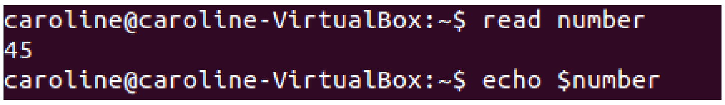
(1) Variables

- Do not need to be declared before use



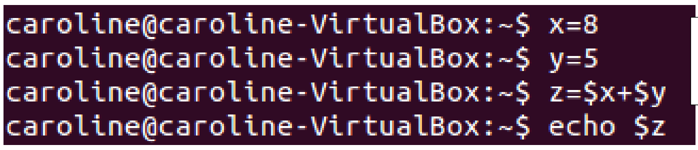
```
caroline@caroline-VirtualBox:~$ x=10
caroline@caroline-VirtualBox:~$ echo $x
```

- Use the read command to get input from the user into a variable



```
caroline@caroline-VirtualBox:~$ read number
45
caroline@caroline-VirtualBox:~$ echo $number
```

- Shell variables are untyped and are effectively treated as strings by default

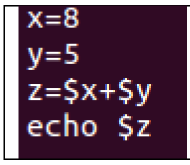
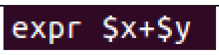
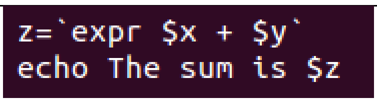


```
caroline@caroline-VirtualBox:~$ x=8
caroline@caroline-VirtualBox:~$ y=5
caroline@caroline-VirtualBox:~$ z=$x+$y
caroline@caroline-VirtualBox:~$ echo $z
```

The output is:

- So instead we need to use the **expr** command

Try out each of the following:

 x=8
y=5
z=$x+$y
echo $z
	 expr $x+$y	 `` z=`expr $x + $y` ``
		echo The sum is $z
			Now try out multiplying 6 by 4

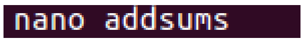
Writing a shell script

Shell scripts are short programs that are written in a shell programming language and interpreted by a shell process in the console.

You can create these programs via a text editor and execute them in a terminal window.
To successfully write a shell script, you have to do three things:

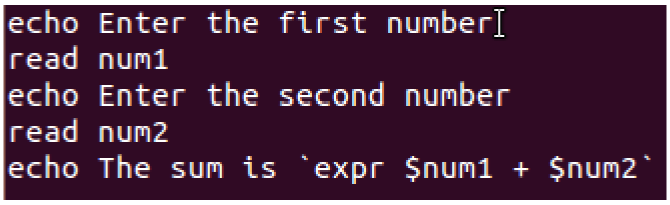
Step 1. Write a script
Step 2. Give the shell permission to execute it
Step 3. Put it somewhere the shell can find it

Step 1. Write a script: Create for example, a script called **addsums** in the nano editor:



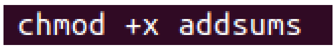
```
nano addsums
```

Add the following code:



```
echo Enter the first number
read num1
echo Enter the second number
read num2
echo The sum is `expr $num1 + $num2`
```

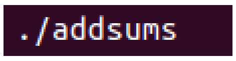
Step 2. Give the shell permission to execute it



```
chmod +x addsums
```

- what does all this mean??

Step 3. Execute it



```
./addsums
```

Again, let's (1) create another script

```
#!/bin/bash
#The famous Hello, World! program
echo "Hello, world."
```

(2) Make it executable via **chmod 755 yourFileName**
(3) Run the script
(4) Now, add the command **clear** to the top of the helloworld script and run it again to see the result

The first line of the script is important. This is a special clue given to the shell indicating what program is used to interpret the script. In this case, it is **/bin/bash**. Other scripting languages such as perl, awk, tcl, Tk, and python can also use this mechanism.

The second line is a comment. Everything that appears after a "**#**" symbol is ignored by bash. As your scripts become bigger and more complicated, comments become vital. They are used by programmers to explain what is going on so that others can figure it out.

The last line is the echo command. This command simply prints what it is given on the display.

The "755" will give you read, write, and execute permission. Everybody else will get only read and execute permission. If you want your script to be private (i.e., only you can read and execute), use "700" instead.

RECALL: $\mathbf{7_8} \equiv \underline{\quad}_2$ $\mathbf{5_8} \equiv \underline{\quad}_2$

The shell provides a rudimentary programming language with facilities for:

I. **Variables**
II. **Decision control – if/then/else, case**
III. **Loop control – while/do, until, for**
IV. **Command line arguments**
V. **Functions**

These facilities (even loops, etc) can be entered interactively at the command line as well as in scripts

- Variables are untyped and do not need to be declared in advance.
- Thus at any point in a shell script we can have a variable **assignment**; e.g.

$ **x=10** (no spaces around = sign)

- Access to the value of a variable requires the **$** symbol.

```
$ echo $x
10
```

I. Variables

Exercise 1. Create a script to allow the user enter two numbers and output the multiplication

Exercise 2. Create a script that will remove all Java files from the current directory, if the user confirms that they are sure they want to remove the files. Otherwise, simply return a comment to indicate to the user that there were no files removed

II. Decision control – if/then/else, case

Exercise 3. The case…esac command (similar to an if-else statement)

The case statement allows you to easily check pattern (conditions) and then process a command-line if that condition evaluates to true.

In other words the $variable-name is compared against the patterns until a match is found, such as when you want to take in a single digit (0-9) and return it’s word value (zero-nine) i.e. an input of 2 will output in the word “two”

NOTE: The command ;; indicates that program flow should jump to the end of the entire case statement. This is similar to break in the C programming language.

EXERCISE: try creating a script that translates a digit to a number

III. Loop control – while/do, until, for

Exercise 4. While loop

Print the numbers from 1-100

- The general format of the **while** construct is as follows

```
while [ condition ]
do
        commands
done
```

Recreate the following script that prints the numbers from 1 to 100:

```
num=1
while [ $num -le 100 ]
do
        echo $num
        num=`expr $num + 1`
done
```

Exercise 5. for loop

The general format of the **for** construct is as follows

```
for var in valuelist
do
        commands
done
```

Write the code to remove all files ending in ".class" ANS___________________________

Now, go to http://linuxcommand.org/wss0020.php and create the *ls -l* alias

V. Functions

- Writing the hello() function

Type the following command at a shell prompt:

```
hello() { echo 'Hello world!' ; }
```

- Invoking the hello() function

hello() function can be used like normal command. To execute, simply type:

```
hello
```

Passing the arguments to the hello() function

You can pass command line arguments to user defined functions. Define hello as follows:

```
hello() { echo "Hello $1, lovely sunny Thursday Feb morning!" ; }
```

You can hello function and pass an argument as follows:

```
hello Caroline
```

Sample outputs:
```
Hello Caroline, lovely sunny Thursday Feb morning!
```