What is Shell Script?

- A **shell script** is a script written for the shell such as Bash (Bourne Again Shell), Zsh, Ksh, or Sh
- Two key ingredients:
  - UNIX/LINUX commands
  - Shell programming syntax

Introduction to Shell Programming Language

Why Shell Scripting ?

- Shell programming is an essential skill to automate workflows, manage files, and perform repetitive tasks efficiently.

- Useful to create own commands

- Save lots of time on file processing.

- To automate some task of day to day life

My First Shell Script

```
$ nano myfirstscript.sh

  #! /bin/sh

  # The first example of a shell script

       directory=`pwd`

      echo Hello World!

      echo The date today is `date`

      echo The current directory is $directory
$ chmod +x myfirstscript.sh
$ ./myfirstscript.sh
Hello World!
The date today is Wed Oct 16 05:45:46 EAT 2024
The current directory is /Users/assohoun/stan
```

Shell Programming

Programming features of the UNIX/LINUX shell:

- **Shell variables**: Shell variables are symbolic names that can access values stored in memory
- **Operators:** Shell scripts support many operators, including those for performing mathematical operations
- **Logic structures**: Shell scripts support
  - sequential logic (for performing a series of commands),
  - decision logic (for branching from one point in a script to another),
  - looping logic (for repeating a command several times),
  - and case logic (for choosing an action from several possible alternatives)

Shell variables.

Variables are symbolic names that represent values stored in memory

Three different types of variables:

Global Variables: Environment and configuration variables, capitalized, such as HOME, PATH, SHELL, USERNAME, and PWD.

When you login, there will be a large number of global System variables that are already defined. These can be freely referenced and used in your shell scripts.

Local Variables

Within a shell script, you can create as many new variables as needed. Any variable created in this manner remains in existence only within that shell.

Special Variables

Reversed for OS, shell programming, etc. such as positional parameters $0, $1 …

Local variable

- Variables are symbolic names that represent values stored in memory
- variables can be defined and used in shell scripts

For example, to initialse a variable called num1 and assign it the value of 1 we would do so like this

```
num1=1
```

Note there is no space between the variable name (num1), the = and the value.

- The variable is then called using a ${} around the name.

```
echo ${num1}
```

User Input

- script input from the standard input location is done via the read command

For example:

```
echo "Please enter three nombers:”
read  nb1 nb2 nb3
echo “These numbers are :$filea  $fileb  $filec”
```

- Each read statement reads an entire line. In the above example if there are less than 3 items in the response the trailing variables will be set to blank ‘ ‘.

Three items are separated by one space.

Positional Parameters

- When a shell script is invoked with a set of command line parameters each of these parameters are copied into special variables that can be accessed

$0 This variable that contains the name of the script

$1, $2, ….. $n 1st, 2nd 3rd command line parameter

$# Number of command line parameters

$$ process ID of the shell

$@ same as $* but as a list one at a time (see for loops later)

$? Return code ‘exit code’ of the last command

For example:

Invoke : **./myscript one two hello**

During the execution of myscript variables $1 $2 $3 will contain the values one, two, hellorespectively

Defining and evaluating operators

expr supports the following operators:

arithmetic operators: +,-,*,/,%

comparison operators: <, <=, ==, !=, >=, >

boolean/logical operators: &, |

Example:

```
nano math.sh
#!/bin/sh
count=5
count=`expr $count + 1 `
echo $count
chmod +x math.sh
./math.sh
```

Arithmetic operators

var++ ,var-- , ++var , --var	post/pre increment/decrement
+ , -	add subtract
*** , / , %**	multiply/divide, remainder
******	power of
! , ~	logical/bitwise negation
& , |	bitwise AND, OR
&& ||	logical AND, OR

Arithmetic operators

var++ ,var-- , ++var , --var	post/pre increment/decrement
+ , -	add subtract
* , / , %	multiply/divide, remainder
**	power of
! , ~	logical/bitwise negation
& , |	bitwise AND, OR
&& ||	logical AND, OR

Practical 1 :

Let’s practice

Write a script that displays, upon execution, the number of parameters and each parameter provided to the script

Writing loops

- A for loop has the syntax

```
for <variable> in <list>
do
    <tasks to repeat for each item in list>
done
```

Alternatively you can place a loop all on one line using ; to separate the commands (except for the line break after the do where there is no ; included).

```
for <variable> in <list> ; do  <tasks to repeat for each item in list> ;
done
```

Practical 2 :

Let’s practice

1. Use a for loop to print the number 1 to 10 to screen
2. Use a for loop to create 3 directories which will be named run1, run2 and run3 . into each run folder (created above) and create a blank file called ‘result.txt’

If/else statements and logic (boolean) operators

Conditional control of tasks can be achieved in UNIX/LINUX using the if else statements.

```
if [[ logic check ]]; then <task if true>; else <task if false>; fi
```

For example, to check if a variable named var1 contains the word “hello” the following statement can be used

```
if [[ ${var1} == "hello" ]];then echo "it does"; else echo "it doesn't";fi
```

If/else statements and logic (boolean) operators

These statements can also be used for logic operators

!	for NOT
-a	for AND
-o	for OR
-eq	for equal (== will also usually work)
-lt	for less than
-gt	for great than

Example1 , we can check if one variable is less than the other

```
if [[ ${var1} -lt ${var2} ]];then echo "it is less"; else echo "it is not less";fi
```

Example 2 , Arithmetic can be done in such statements with ` ticks around the expr

```
if [[ `expr ${var1} + ${var2}` -eq 3 ]];then echo "the sum is 3"; else echo "the sum is not 3";fi
```

Practical 3 :

Let’s practice

1.Initialise a variable to contain a number

2.Write a statement that prints “even” if the number is a multiple of 2 and “odd” if it is not (hint, use the modulo arithmetic operator)