Shell Scripting

What is a Shell Script?

- A shell script is a computer program designed to be run by the Unix/Linux shell
- A shell is a command-line interpreter and typical operations performed by shell scripts include file manipulation, program execution, and printing text.

What is a shell?

- A **Shell** provides you with an interface to the Unix system.
- It gathers input from you and executes programs based on that input. When a program finishes executing, it displays that program's output.
- Shell is an environment in which we can run our commands, programs, and shell scripts

Shell Types

In Unix, there are two major types of shells −

- **Bourne shell** − If you are using a Bourne-type shell, the **$** character is the default prompt.
- **C shell** − If you are using a C-type shell, the % character is the default prompt.

The Bourne Shell has the following subcategories −

- Bourne shell (sh)
- Korn shell (ksh)
- Bourne Again shell (bash)
- POSIX shell (sh)

The different C-type shells follow −

- C shell (csh)
- TENEX/TOPS C shell (tcsh)

Bourne shell was the first shell to appear on Unix systems, thus it is referred to as "the shell".

Bourne shell is usually installed as **/bin/sh** on most versions of Unix.

Steps to write and execute a script

- Open the terminal. Go to the directory where you want to create your script.
- Create a file with **.sh** extension.
- Write the script in the file using an editor.
- Make the script executable with command **chmod +x <fileName>**.
- Run the script using ./<**fileName**>.

Note: In the last step you have to mention the path of the script if your script is in other directory.

Shell Scripts - Example

We create a shell Script with a file name test.sh script. Note all the scripts would have the **.sh** extension. Before you add anything else to your script, you need to alert the system that a shell script is being started. This is done using the **shebang** construct. For example –

#!/bin/sh

- tells the system that the commands that follow are to be executed by the Bourne shell. *It's called a shebang because the # symbol is called a hash, and the ! symbol is called a bang*.

Example :

```
#!/bin/sh

echo "What is your name?"
read PERSON
echo "Hello, $PERSON"
```

Variable names in shell Script :-

The name of a variable can contain only letters (a to z or A to Z), numbers (0 to 9) or the underscore character (_).

By convention, Unix shell variables will have their names in UPPERCASE.

valid variable names	In valid variable names
_ALI TOKEN_A VAR_1 VAR_2	2_VAR -VARIABLE VAR1-VAR2 VAR_A!

The reason you cannot use other characters such as **!**, *****, or **-** is that these characters have a special meaning for the shell.

Defining Variables

variable_name=variable_value

```
Eg : NAME="Zara Ali"
```

Variables of this type are called **scalar variables**. A scalar variable can hold only one value at a time.

Shell enables you to store any value you want in a variable

Eg :

```
VAR1="Zara Ali"
VAR2=100
```

Accessing Values

To access the value stored in a variable, prefix its name with the dollar sign (**$**)

```
#!/bin/sh

NAME="Zara Ali"
echo $NAME
```

Special Variables in shell Script :-

Sr.No.	Variable & Description
1	**$0** The filename of the current script.
2	**$n** These variables correspond to the arguments with which a script was invoked. Here **n** is a positive decimal number corresponding to the position of an argument (the first argument is $1, the second argument is $2, and so on).
3	**$#** The number of arguments supplied to a script.
4	**$*** All the arguments are double quoted. If a script receives two arguments, $* is equivalent to $1 $2.
5	**$@** All the arguments are individually double quoted. If a script receives two arguments, $@ is equivalent to $1 $2.
6	**$?** The exit status of the last command executed.
7	**$$** The process number of the current shell. For shell scripts, this is the process ID under which they are executing.

8	**$!** The process number of the last background command.

Command-Line Arguments

The command-line arguments $1, $2, $3, ...$9 are positional parameters, with $0 pointing to the actual command, program, shell script, or function and $1, $2, $3, ...$9 as the arguments to the command

Using Shell Arrays

Shell supports a different type of variable called an **array variable**. This can hold multiple values at the same time. Arrays provide a method of grouping a set of variables. Instead of creating a new name for each variable that is required, you can use a single array variable that stores all the other variables.

Defining Array Values ----- array_name[index]=value

Accessing Array Values ------ ${array_name[index]}

Eg :

```
#!/bin/sh

NAME[0]="Zara"
NAME[1]="Qadir"
NAME[2]="Mahnaz"
NAME[3]="Ayan"
NAME[4]="Daisy"
echo "First Index: ${NAME[0]}"
echo "Second Index: ${NAME[1]}"
```

Shell Basic Operators

- Arithmetic Operators
- Relational Operators
- Boolean Operators
- String Operators
- File Test Operators

Bourne shell didn't originally have any mechanism to perform simple arithmetic operations but it uses external programs, either **awk** or **expr**.

The following example shows how to add two numbers

```
#!/bin/sh

val=`expr 2 + 2`
echo "Total value : $val"
```