**Shell Scripting** is an open-source computer program designed to be run by the Unix/Linux shell. Shell Scripting is a program to write a series of commands for the shell to execute. It can combine lengthy and repetitive sequences of commands into a single and simple script that can be stored and executed anytime which, reduces programming efforts.

A shell script is a list of commands in a computer program that is run by the Unix shell which is a command line interpreter. A shell script usually has comments that describe the steps. The different operations performed by shell scripts are program execution, file manipulation and text printing. A wrapper is also a kind of shell script that creates the program environment, runs the program etc.

A shell in a Linux operating system takes input from you in the form of commands, processes it, and then gives an output. It is the interface through which a user works on the programs, commands, and scripts. A shell is accessed by a terminal which runs it.

When you run the terminal, the Shell issues **a command prompt (usually $),** where you can type your input, which is then executed when you hit the Enter key. The output or the result is thereafter displayed on the terminal.

**Types of Shells**

There are two major types of shells in Unix. These are:

**Bourne Shell**

This is the default shell for version 7 Unix. The character $ is the default prompt for the bourne shell. The different subcategories in this shell are Korn shell, Bourne Again shell, POSIX shell etc.

**C Shell**

This is a Unix shell and a command processor that is run in a text window. The character % is the default prompt for the C shell. File commands can also be read easily by the C shell, which is known as a script.

**Capabilities of Shell Script**

The different capabilities of the shell script are −

* **Batch jobs**

Several commands that would be entered manually in a command line interface can be executed automatically using a shell script. This can be done without the user needing to trigger each command separately.

* **Programming**

There are many features in modern shell scripts that are only found in sophisticated programming languages such as arrays, variables, comments etc. Many complicated applications can be written in shell scripts using these features. But there is a problem i.e. shell script languages don’t support classes, threading etc.

* **Generalisation**

It is much more flexible to use loops, variables etc for multiple tasks in shell script. An example of this is a Unix shell script known as bash, which converts jpg images to png images.

* **Shortcuts**

There is a shortcut provided by a shell script for a system command where command options, environment settings or post processing apply. This still allows the shortcut script to act as a Unix command.

**Advantages of Shell Script**

Some of the advantages of shell script are −

* The commands and syntax of the shell script are the same as that entered at the command line. Because of this, there is no need to switch to a completely different syntax.
* It is much faster to write a code in shell script than in other programming languages. This also means that the program is easier to create and files required can be selected easily.
* Shell script can also be used to provide linkage for already existing programs.
* Shell scripting can be used by users that are not experts to modify and tailor the behaviour of their programs according to their requirements.

**Disadvantages of Shell Script**

Some of the disadvantages of shell script are −

* There may be errors in shell scripting that prove to be quite costly.
* The programs in shell script are quite slow while executing and a new process is required for every shell command executed.
* Different platforms in shell scripting may also have compatibility problems.

EXAMPLE

#!/bin/bash

echo "Welcome!! Please Enter Your Name"

read name

echo "Hello $name"

**#!/bin/bash**

A line starting with #! is used to tell the os which interpreter to use to execute the file. There are many shell interpreters like ksh, zsh(macOS), csh(C shell), ssh(secure shell), etc. The above line asks the system to use the bash interpreter.

**echo "Welcome!! Please Enter Your Name"**

The echo keyword is used to output the strings that it is being passed as arguments. It outputs onto the stdout.

**read name**

The read command is used to receive input while running the script. When we wrote read name it initialized a variable named name and stored the input in it.

**echo "Hello $name"**

The $ symbol is used to print the **value** of a variable. In the above example, it prints whatever input was given in the previous step.

**Executing the file**

Now that we have seen what a typical shell script looks like, let’s look at how to execute the file.

1. Save the file with a .sh extension
2. To execute the file, first we need to give it execute permissions.

chmod +x filepath/filename.sh

1. To execute the file, we can do it in the following ways
   * If you are using a GUI file navigation system, right-click on the file and click on run or execute.
   * If you are using the terminal, ./filename.sh will execute the script. (Make sure you are in the correct directory!)

### Variables

Variables in shell scripting are similar to variables in general programming languages; they are used as a pointer to the actual data. Variables do not have to be declared, as compared to programming languages like C, but if you try to read from an undeclared variable, then you will not get intended results.

my\_name="Rohan Reddy"

echo $my\_name

#### Naming conventions of variables

1. All caps and underscores for exported variables and constants, when they are **shared across multiple scripts**. Ex: JOB\_ID, PROCESS\_NAME
2. All lowercase and underscores for all variables that are **scoped to a single script**. Ex: max\_amount , i.
3. Use leading underscore for **private variables and functions**; can be used where functions share the same variables without clashing with similarly named variables in the code. Ex: \_debug

**When you are declaring variables, make sure there are no spaces before and after** =**.** (a=12 is correct; a = 12 throws an error.)

### Special Keywords and Variables

$PWD, this variable contains the value of the **Present Working Directory**

Arguments passed from the **command line** can be accessed by using $0, $1, $2, … notation. $0 is the name of the script itself, $1 is the first argument, $2 is the second argument and so on.

./script 1 2 3

**Note:** These variables not only work from the command line, but also with functions. You should use functions to organize your bash code just like your would in another language. Variables can be passed to functions just like you do with the command line, using $1, $2, etc.

### Control-flow Constructs

#### If-else statements

**Simple if statement**

if [expression]

statement

if

**Simple if-else statement**

if [expression]

then

command1

else

command2

if

**if-else\_if-else statement**

if [expression]

then

statement1

else [expression2]

then

statement2

else

statement3

if

**Nested if statements**

if[expression1]

then

if [expression2]

then

statement1

else

statement2

if

else

statement3

if

### Switch-Case

The switch statement is a multiway branch statement. It provides an easy way to dispatch execution to different parts of code based on the value of the expression. In shell scripting, we have the case command for this.

case "$var" in

#case 1

pattern\_1) statement\_1;;

#case 2

patter\_2) statement\_2;;

case

**Example:**

id=1

case $id in

#case 1

1) echo "ID is 1";;

#case 2

2) echo "ID is 2";;

case

### Loops

Loops are powerful tools that help us execute a set of commands repeatedly.

**While loop** executes the given commands until the given condition becomes false.

while condition

do

statement

done

**Example:**

#!/bin/sh

cars=10

while [ "$cars" -gt 0 ]

do

cars=`expr $cars - 1`

echo $cars

done

When writing shell scripts, be mindful of where spaces should exist and where there should be no spaces.

**For loop** is an efficient way of writing loops. Syntax:

for i in range

do

command

done

### Examples

We can loop over a range of numbers, loop over an array of numbers, or loop over the output from a command.

**Example 1:**

for i in 1 2 3 4 5

do

echo $i #prints 1 to 5

done

**Example 2:**

for i in {1..5}

do

echo $i #can be used to specify range

done

**Example 3:**

#!/bin/bash

for (( c=1; c<=5; c++ )) #this is one of the ways we can write loops which is similar to C-style loops

do

echo "Welcome $c times"

done

### Break & Continue

**break** and **continue** keywords can be used with the same meaning as any programming languages. break is used to stop the execution of a loop, and continue can be used to cause the loop to execute the next iteration.