**Shell Scripting:**

**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. There are different flavors of a shell, just as there are different flavors of operating systems. Each flavor of shell has its own set of recognized commands and functions.

Ex: hello.sh ----shell script save with .sh extension.

#!/bin/bash 🡪 it is called **Shebang** Constructor.

Date

echo $date

NAME="sriteja"

echo $NAME

pwd

cal

Ex2: teja.sh 🡪 input and output command

#!/bin/bash

echo "what is ur name"

read PERSON

echo " hi, $PERSON"

**Using Arrays:**

If you are using the **bash** shell, here is the syntax of array initialization −

array\_name=(value1 ... valuen)

## Accessing Array Values

After you have set any array variable, you access it as follows −

${array\_name[index]}

Here *array\_name* is the name of the array, and *index* is the index of the value to be accessed. Following is an example to understand the concept −

#!/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]}"

The above example will generate the following result –

$./test.sh

First Index: Zara

Second Index: Qadir

**Using operators:**

We will now discuss the following 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**.

Ex:- print the sum of two numbers🡪 operators.sh

#!/bin/bash

Sum=’expr 2 + 4’

Echo “Total Sum = $sum”

Output:-

Total Sum = 6

The following points need to be considered while adding -

* There must be spaces between operators and expressions. For example, 2+2 is not correct; it should be written as 2 + 2.
* The complete expression should be enclosed between **‘ ‘**, called the backtick.

A=10 and B=20

**Arthemetic Operator:**

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| + (Addition) | Adds values on either side of the operator | `expr $a + $b` will give 30 |
| - (Subtraction) | Subtracts right hand operand from left hand operand | `expr $a - $b` will give -10 |
| \* (Multiplication) | Multiplies values on either side of the operator | `expr $a \\* $b` will give 200 |
| / (Division) | Divides left hand operand by right hand operand | `expr $b / $a` will give 2 |
| % (Modulus) | Divides left hand operand by right hand operand and returns remainder | `expr $b % $a` will give 0 |
| = (Assignment) | Assigns right operand in left operand | a = $b would assign value of b into a |
| == (Equality) | Compares two numbers, if both are same then returns true. | [ $a == $b ] would return false. |
| != (Not Equality) | Compares two numbers, if both are different then returns true. | [ $a != $b ] would return true. |

**Relational Operator:**

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| **-eq** | Checks if the value of two operands are equal or not; if yes, then the condition becomes true. | [ $a -eq $b ] is not true. |
| **-ne** | Checks if the value of two operands are equal or not; if values are not equal, then the condition becomes true. | [ $a -ne $b ] is true. |
| **-gt** | Checks if the value of left operand is greater than the value of right operand; if yes, then the condition becomes true. | [ $a -gt $b ] is not true. |
| **-lt** | Checks if the value of left operand is less than the value of right operand; if yes, then the condition becomes true. | [ $a -lt $b ] is true. |
| **-ge** | Checks if the value of left operand is greater than or equal to the value of right operand; if yes, then the condition becomes true. | [ $a -ge $b ] is not true. |
| **-le** | Checks if the value of left operand is less than or equal to the value of right operand; if yes, then the condition becomes true. | [ $a -le $b ] is true. |

**Boolean Operators:**

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| **!** | This is logical negation. This inverts a true condition into false and vice versa. | [ ! false ] is true. |
| **-o** | This is logical **OR**. If one of the operands is true, then the condition becomes true. | [ $a -lt 20 -o $b -gt 100 ] is true. |
| **-a** | This is logical **AND**. If both the operands are true, then the condition becomes true otherwise false. | [ $a -lt 20 -a $b -gt 100 ] is false. |

**Conditional Statements:**

Unix Shell supports conditional statements which are used to perform different actions based on different conditions. We will now understand two decision-making statements here −

* The **if...else** statement
* The **case...esac** statement

The if...else statements

If else statements are useful decision-making statements which can be used to select an option from a given set of options.

Unix Shell supports following forms of **if…else** statement −

* [if...fi statement](https://www.tutorialspoint.com/unix/if-fi-statement.htm)
* [if...else...fi statement](https://www.tutorialspoint.com/unix/if-else-statement.htm)
* [if...elif...else...fi statement](https://www.tutorialspoint.com/unix/if-elif-statement.htm)