Operating System

**LAB-4**

**Name \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Roll No \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Date \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_­\_**

**Marks Obtained \_\_\_\_\_\_\_\_\_\_\_\_**

**Signature\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

UNIX/LINUX Shell programming

Array, Operator

**Objective: Understand and implement Array and Operator in Linux Shell.**

A shell variable is capable enough to hold a single value. This type of variables are called scalar variables.

Shell supports a different type of variable called an array variable that 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.

All the naming rules discussed for Shell Variables would be applicable while naming arrays.

## **Defining Array Values**

The difference between an array variable and a scalar variable can be explained as follows.

Say that you are trying to represent the names of various students as a set of variables. Each of the individual variables is a scalar variable as follows −

NAME01="Iqra"

NAME02="University"

NAME03="Computer Lab"

NAME04="Engineering Lab"

NAME05="Telecom Lab"

We can use a single array to store all the above mentioned names. Following is the simplest method of creating an array variable is to assign a value to one of its indices. This is expressed as follows −

array\_name[index]=value

Here *array\_name* is the name of the array, *index* is the index of the item in the array that you want to set, and value is the value you want to set for that item.

As an example, the following commands −

NAME[0]="Iqra"

NAME[1]="University"

NAME[2]="Computer Lab"

NAME[3]="Engineering Lab"

If you are using **ksh** shell the here is the syntax of array initiUniversityzation −

set -A array\_name value1 value2 ... valuen

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

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 the simplest example −

#!/bin/sh

NAME[0]="Iqra"

NAME[1]="University"

NAME[2]="Computer Lab"

NAME[3]="Engineering Lab"

echo "First Index: ${NAME[0]}"

echo "Second Index: ${NAME[1]}"

This would produce following result −

$./test.sh

First Index: Iqra

Second Index: University

You can access all the items in an array in one of the following ways −

${array\_name[\*]}

${array\_name[@]}

Here array\_name is the name of the array you are interested in. Following is the simplest example −

#!/bin/sh

NAME[0]="Iqra"

NAME[1]="University"

NAME[2]="Computer Lab"

NAME[3]="Engineering Lab"

echo "First Method: ${NAME[\*]}"

echo "Second Method: ${NAME[@]}"

This would produce following result −

$./test.sh

First Method: Iqra University Computer Lab Engineering Lab

Second Method: Iqra University Computer Lab Engineering Lab

**Operator**

There are various operators supported by each shell. Our tutorial is based on default shell (Bourne) so we are going to cover all the important Bourne Shell operators in the tutorial.

There are following operators which we are going to discuss −

* Arithmetic Operators.
* Relational Operators.
* Boolean Operators.
* String Operators.
* File Test Operators.

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

Here is simple example to add two numbers −

#!/bin/sh

val=`expr 2 + 2`

echo "Total value : $val"

This would produce following result −

Total value : 4

There are following points to note down −

* There must be spaces between operators and expressions for example 2+2 is not correct, where as it should be written as 2 + 2.
* Complete expression should be enclosed between **``**, called inverted commas.

## **Arithmetic Operators**

There are following arithmetic operators supported by Bourne Shell.

Assume variable a holds 10 and variable b holds 20 then −

|  |  |  |
| --- | --- | --- |
| **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 % $a` will give 0 |
| = | Assignment - Assign right operand in left operand | a=$b would assign value of b into a |
| == | EquUniversityty - Compares two numbers, if both are same then returns true. | [ $a == $b ] would return false. |
| != | Not EquUniversityty - Compares two numbers, if both are different then returns true. | [ $a != $b ] would return true. |

It is very important to note here that all the conditional expressions would be put inside square braces with one spaces around them, for example [ $a == $b ] is correct where as [$a==$b] is incorrect.

All the arithmetical calculations are done using long integers.

## **Relational Operators:**

Bourne Shell supports following relational operators which are specific to numeric values. These operators would not work for string values unless their value is numeric.

For example, following operators would work to check a relation between 10 and 20 as well as in between "10" and "20" but not in between "ten" and "twenty".

Assume variable a holds 10 and variable b holds 20 then −

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| -eq | Checks if the value of two operands are equal or not, if yes then 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 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 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 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 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 condition becomes true. | [ $a -le $b ] is true. |

It is very important to note here that all the conditional expressions would be put inside square braces with one spaces around them, for example [ $a <= $b ] is correct where as [$a <= $b] is incorrect.

## **Boolean Operators**

There are following boolean operators supported by Bourne Shell.

Assume variable a holds 10 and variable b holds 20 then −

|  |  |  |
| --- | --- | --- |
| **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 condition would be true. | [ $a -lt 20 -o $b -gt 100 ] is true. |
| -a | This is logical AND. If both the operands are true then condition would be true otherwise it would be false. | [ $a -lt 20 -a $b -gt 100 ] is false. |

## **String Operators**

There are following string operators supported by Bourne Shell.

Assume variable a holds "abc" and variable b holds "efg" then −

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| = | Checks if the value of two operands are equal or not, if yes then condition becomes true. | [ $a = $b ] is not true. |
| != | Checks if the value of two operands are equal or not, if values are not equal then condition becomes true. | [ $a != $b ] is true. |
| -z | Checks if the given string operand size is zero. If it is zero length then it returns true. | [ -z $a ] is not true. |
| -n | Checks if the given string operand size is non-zero. If it is non-zero length then it returns true. | [ -n $a ] is not false. |
| str | Check if str is not the empty string. If it is empty then it returns false. | [ $a ] is not false. |

**Task: Write bash scrip to test all Athematic and logical operators in Linux**

|  |  |  |
| --- | --- | --- |
| **Operator: Task** | **Script** | **Output** |
| **+ : add two number** |  |  |
| **- : make decrement operator** |  |  |
| **\* : Mutiply negative and positive number** |  |  |
| **/: divide any number by zero** |  |  |
| **Less than: Compare two value** |  |  |
| **Greater than: Compare two value** |  |  |
| **EquUniversityty: Compare two value** |  |  |
| **Not equal: Compare two value** |  |  |
| **And: make two simple and gate** |  |  |
| **Or: make two input or gate** |  |  |
| **Not: make 1 input not** |  |  |