**Bash Scripting**

**Bash Single Line Comment**

**Bash Multi Line Comment**

**Bash Variables**

**Command line Arguments**

**"Read" User Input**

**Bash Date Format**

**Bash Sleep**

**Bash Arithmetic Operators**

**if Statements**

**If Else**

**For Loop**

**Example**

#!/bin/bash

-------------------------------------------------------------------------------

#! /bin/bash

# This is the basic bash script

echo " Hello World! "

==================================

Chmod +x filename.sh

./filename.sh

Or

Bash filename.sh

Output

Hello World

# Bash Comments

## Bash Single Line Comment

#!/bin/bash

#This is a single-line comment in Bash Script.

echo "Enter your name:"

read $name

echo

#echo output, its also a single line comment

echo "The current user name is $name"

#This is another single line comment

## Bash Multi Line Comment

**Method 1:**

#!/bin/bash

<<COMMENTS

This is the first comment

This is the second comment

This is the third comment

COMMENTS

echo "Hello World"

**Method 2:**

**#!/bin/bash**

**: '**

**This is the first comment**

**This is the second comment**

**This is the third comment**

**'**

**echo "Hello World"**

# Bash Variables

Variable\_name =value

1. year=2012
2. comp\_name=jtp

## Types of Bash Variables

1. System-Defined Variables
2. User-Defined Variables

**System-Defined Variables:** These are the pre-defined variables

Example:

BASH=/usr/bin/bash

BASH\_VERSION=4.2.46(2)

To know the list of these variables in your system, type the commands **set, env**, and **printenv** on the command line

#! /bin/bash

# Bash System-defined Variables

echo $HOME # Home Directory

echo $PWD # current working directory

echo $BASH # Bash shell name

echo $BASH\_VERSION # Bash shell Version

echo $LOGNAME # Name of the Login User

echo $OSTYPE # Type of OS

**User-defined Variables:** These variables are created and maintained by the user.

#! /bin/bash

# User-Defined Variables

name=Peter

ROLL\_NO=5245325

echo "The student name is $name and his Roll number is $ROLL\_NO."

## Command line Arguments

Command Line Arguments are used to make a script more dynamic by passing input to the code. We pass these arguments at the runtime of the script as the following form:

./script\_name arg1 arg2 arg3.....

In a Bash Shell, they are used with the reference of the following default-parameters or the special variables.

* **$0** specifies the name of the script to be invoked.
* **$1-$9** stores the names of the first 9 arguments or can be used as the arguments' positions.
* **$#** specifies the total number (count) of arguments passed to the script.
* **$\*** stores all the command line arguments by joining them together.
* **$@** stores the list of arguments as an array.
* **$?** specifies the process ID of the current script.
* **$$** specifies the exit status of the last command or the most recent execution process.
* **$!** shows ID of the last background job.

# Read User Input

### Syntax

read <variable\_name>

#!/bin/bash

# Read the user input

echo "Enter the the package name: "

read pkg

echo "install $pkg"

Yum $pkg -y

echo

echo "Enter other users'names: "

read name1 name2 name3

echo "$name1, $name2, $name3 are the other users."

## What will happen if we don't pass any variable with the read command?

#!/bin/bash

# using read command without any variable

echo "Enter name : "

read

echo "Name : $REPLY"

## Example 2:

In this example, we enter the input on the same PROMPT by using the **-p command line option** as follows:

read -p PROMPT <variable\_name>

#!/bin/bash

read -p "username:" user\_var

echo "The username is: " $user\_var

Output

Username:ramu

The username is: ramu

This example is to keep the input on silent mode, such that whatever be a user input on the command line will be hidden to others.

So, we pass a username and hide the password (silent mode) by using the command line options (-s, -p) commonly as follows:

read -sp PROMPT <variable\_name>

#!/bin/bash

read -p "username : " user\_var =raju

read -sp "password : " pass\_var = 12345

echo

echo "username : " $user\_var

echo "password : " $pass\_var

Echo “user provided password is” $pass\_var

Output

Username: raju

Password: \*\*\*\*\*\*\*\*

user provided password is 12345

**Example 4:** This example is to enter multiple inputs using an array. So use the -a command line option as follows:

read -a <variable\_name>

#!/bin/bash

# Reading multiple inputs using an array

echo "Enter names : "

read -a names

echo "The entered names are : ${names[0]}, ${names[1]}."

echo $names[0]

**Bash Date Format MM-DD-YYYY**

To use the date in MM-DD-YYYY format, we can use the command **date +%m-%d-%Y**.

#!/bin/bash

d=`date +%m-%d-%Y`

d=10-22-2020

echo "Date in format MM-DD-YYYY"

echo today date is $d #MM-DD-YYY

Output

Date in format MM-DD-YYYY

Today date is 10-22-2020

# Bash Sleep

* Sleep for 9 seconds, use  
  **sleep 9 or sleep 9s**
* Sleep for 0.5 seconds, use  
  **sleep 0.5 or sleep 0.5s**
* Sleep for 2 minute and 30 seconds, use  
  **sleep 2m 30s**
* Sleep for 8 hours  
  **sleep 8h**
* Sleep for 2 days, 9 hours, 5 minute and 55 seconds, use  
  **sleep 2d 9h 5m 55s**

# Bash Arithmetic Operators

There are 11 arithmetic operators which are supported by Bash Shell.

$(( 10 + 3 )), result=13

$(( 10 - 3 )), result=7

$(( 10 \* 3 )), result=30

$(( 10 / 3 )), result=3

$(( 10 \*\* 3 )), result=1000

$(( 10 % 3 )), result=1

x=10

let "x += 3"

echo $x

result=13

x=10

let "x -= 3"

echo $x

result=7

x=10

let "x \*= 3"

echo $x

result=30

x=10

let "10 /= 3"

echo $x

result=3

x=10

let "10 %= 3"

echo $x

result=1

**Syntax**

((expression))

**Method 1**

**Sum=$((10+3))**

**echo "Sum = $Sum"**

**Method 2**

**((Sum=10+3))**

**echo "total = $Sum"**

**Method 3**

**Num1=10**

**Num2=3**

**((Sum=Num1+Num2))**

**echo "Sum = $Sum"**

**Method 4**

**Num1=10**

**Num2=3**

**Sum=$((Num1+Num2))**

**echo "Sum = $Sum"**

**x=6**

**y=3**

**echo "Addition of x & y"**

**echo $(( 8 + 2 ))**

**echo "Subtraction of x & y"**

**echo $(( 8 - 2 ))**

**echo "Multiplication of x & y"**

**echo $(( 8 \* 2 ))**

**echo "Division of x by y"**

**echo $(( 8 / 2 ))**

**echo "Exponentiation of x,y"**

**echo $(( 8 \*\* 2 ))**

**echo "Modular Division of x,y"**

**echo $(( 8 % 2 ))**

**echo "Incrementing x by 5, then x= "**

**(( 8 += 5 ))**

**echo $8**

**echo "Decrementing x by 5, then x= "**

**(( 8 -= 5 ))**

**echo $8**

**echo "Multiply of x by 5, then x="**

**(( 8 \*= 5 ))**

**echo $8**

**echo "Dividing x by 5, x= "**

**echo "x=8, y=2"**

**echo "Addition of x & y"**

**echo $(( $x + $y ))**

**echo "Subtraction of x & y"**

**echo $(( $x - $y ))**

**echo "Multiplication of x & y"**

**echo $(( $x \* $y ))**

**echo "Division of x by y"**

**echo $(( $x / $y ))**

**echo "Exponentiation of x,y"**

**echo $(( $x \*\* $y ))**

**echo "Modular Division of x,y"**

**echo $(( $x % $y ))**

**echo "Incrementing x by 5, then x= "**

**(( x += 5 ))**

**echo $x**

**echo "Decrementing x by 5, then x= "**

**(( x -= 5 ))**

**echo $x**

**echo "Multiply of x by 5, then x="**

**(( x \*= 5 ))**

**echo $x**

**echo "Dividing x by 5, x= "**

**(( x /= 5 ))**

**echo $x**

**echo "Remainder of Dividing x by 5, x="**

**(( x %= 5 ))**

**echo $x**

# Bash If

## Basic if Statements

### Syntax

if [ expression ];

then

statements

fi

* For using multiple conditions with AND operator:

if [ expression\_1 ] && [ expression\_2 ];

then

statements

Fi

* For using multiple conditions with OR operator:

if [ expression\_1 ] || [ expression\_2 ];

then

statements

fi

### Example 1

#!/bin/bash

read -p " Enter number : " number

if [ $number -gt 125 ]

then

echo "Value is greater than 125"

fi

### Example 2

#!/bin/bash

# if condition is true

if [ "myfile" == "myfile" ];

then

echo "true condition"

fi

# if condition is false

if [ "myfile" == "yourfile" ];

then

echo "false condition"

fi

### Example 3

#!/bin/bash

# TRUE && TRUE

if [ 8 -gt 6 ] && [ 10 -eq 10 ];

then

echo "Conditions are true"

fi

# TRUE && FALSE

if [ "mylife" == "mylife" ] && [ 3 -gt 10 ];

then

echo "Conditions are false"

fi

|  |  |
| --- | --- |
| -d FILE | To check if FILE exists and it is a directory. |
| -e FILE | To check if FILE exists. |
| -r FILE | To check if FILE exists and the read permission is granted. |
| -s FILE | To check if FILE exists and its size is greater than zero (which means that it is not empty). |
| -w FILE | To check if FILE exists and the write permission is granted. |
| x FILE | To check if FILE exists and the execute permission is granted. |

## Nested If

You can apply as many 'if statements' as required inside your bash script.

#!/bin/bash

#Nested if statement

if [ $1 -gt 50 ]

then

echo "Number is greater than 50."

if (( $1 % 2 == 0 ))

then

echo "and it is an even number."

fi

Fi

# Bash If Else

### Bash If Else Syntax

if [ condition ];

then

<if block commands>

else

<else block commands>

fi

### Example 1

#!/bin/bash

#when the condition is true

if [ 10 -gt 3 ];

then

echo "10 is greater than 3."

else

echo "10 is not greater than 3."

fi

#when the condition is false

if [ 3 -gt 10 ];

then

echo "3 is greater than 10."

else

echo "3 is not greater than 10."

fi

# Bash For Loop

### Syntax of For Loop

for variable in list

do

commands

Done

Or

for (( expression1; expression2; expression3 ))

do

commands

done

### For Loop' Example

#!/bin/bash

#This is the basic example to print a series of numbers from 1 to 10.

for num in {1..10}

do

echo $num

done

echo "Series of numbers from 1 to 10."

Vim file.txt

Mdm

Gcm

Doc

Sms

email

Example

**#!/bin/bash**

**for i in `cat file.txt`**

**do**

**cd /webroot/chbuildbackup/**

**path=$i**

**cd $path**

**b=$(ls |wc -l)**

**4**

**if [ $b -le 3 ]**

**then**

**echo "$i directory have $b files only ,not deleted any file"**

**else**

**ls -t | tail -n 1 | xargs rm -rf**

**echo " $i directroy deleted one file"**

**fi**

**done**