**Batch script and Shell script:**

**Batch script**: In Windows, the batch file is a file that stores commands in a serial order. Command line interpreter takes the file as an input and executes in the same order.

A batch file is simply a text file saved with the .bat file extension. It can be written using Notepad or any other text editor.

A simple batch file will be

ECHO OFF

ECHO hello world

PAUSE

After saving it with .bat extension. Double click it to run the file. It prints shows

helloworld

In above script, ECHO off cleans up the console by hiding the commands from being printed at the prompt, ECHO prints the text “GeeksforGeeks” to the screen, and then waits for the user to press a key so program can be ceased.

**Some basic commands of batch file**

**ECHO** – Prints out the input string. It can be ON or OFF, for ECHO to turn the echoing feature on or off. If ECHO is ON, the command prompt will display the command it is executing.

**cls** – Clears the command prompt screen.

TITLE: Changes the title text displayed on top of prompt window.

EXIT – To exit the Command Prompt.

PAUSE – Used to stop the execution of Windows batch file.

:: – Add a comment in the batch file.

COPY – Copy a file or files

Types of “batch” files in windows

INI (\*.ini) – Initialization file. These set the default variables for the system and programs.

CFG (\*.cfg) – These are the configuration files.

SYS (\*.sys) – System files, can sometimes be edited, mostly compiled machine code in new versions.

COM (\*.com) – Command files. These are the executable files for all the DOS commands. In early versions there was a separate file for each command. Now, most are inside COMMAND.COM.

CDM (\*.cmd) – These were the batch files used in NT operating systems.

Suppose we need to list down all the files/directory names inside a particular directory and save it to a text file, so batch script for it will be,

@echo off

Rem Listing all the files in the directory Program files

dir "C:\Program Files" > C:\geeks\_list.txt

echo "Done!"

Now when we run this batch script, it will create a file name geeks\_list.txt in your C:\ directory, displaying all the files/folder names in C:\Program Files

Another useful batch script that can be written to diagnose your network and check performance of it.

:: This batch file checks for network connection problems.

ECHO OFF

:: View network connection details

ipconfig /all

:: Check if geeksforgeeks.com is reachable

ping geeksforgeeks.com

:: Run a traceroute to check the route to geeksforgeeks.com

tracert geeksforgeeks.com

PAUSE

This script gives information about the current network and some network packet information. ‘ipconfig /all’ helps to view the network information and ‘ping’ & ‘tracert’ to get each packet info.

**Difference between Ping and Traceroute**

In computer networks, *data is sent in small blocks known as packets. Each packet is transmitted individually and may also follow different route to reach the destination. Once all these packets of the original message reach the destination, they are re-assembled to form the original message*. But, sometimes, it may happen that the web server is down, network congestion or some other technical glitch is there, that may prevent the message from reaching the destination. To diagnose such congestions and network failures, we use two common programs namely Ping and Traceroute.

**Ping** – *It is a utility that helps one to check if a particular IP address is accessible or not*. *Ping works by sending a packet to the specified address and waits for the reply. It also measures round trip time and reports errors*.

Ping is also used in checking if the computers on a local network are active. For this, the user has to go in command prompt and type : ping 127.0.0.1, and if the address is active,

> ping www.google.com

The IP address 127.0.0.1 is the address of the local host and would receive a ping reply even if the sender is not connected to internet.

**Traceroute** – *It is utility that traces a packet from your computer to the host, and will also show the number of steps (hops) required to reach there, along with the time by each step. Traceroute works by sending the packets of data with low survival time (Time to Live – TTL) which specifies how many steps (hops) can the packet survive before it is returned*. When a packet can’t reach the final destination and expires at an intermediate step, that node returns the packet and identifies itself. So, by increasing the TTL gradually, Traceroute is able to identify the intermediate hosts. If any of the hops come back with “Request timed out”, it denotes network congestion and a reason for slow loading Web pages and dropped connections.

> tracert ww.google.com

The main difference between Ping and Traceroute is that Ping is a quick and easy utility to tell if the specified server is reachable and how long will it take to send and receive data from the server whereas Traceroute finds the exact route taken to reach the server and time taken by each step (hop).

**Shell script**: can be used for various purposes, such as executing a shell command, running multiple commands together, customizing administrative tasks, performing task automation etc. So knowledge of bash programming basics is important for every Linux user.

Create and Execute First BASH Program:

You can ***run bash script from the terminal*** or ***by executing any bash file***. Run the following command from the terminal to execute a very simple bash statement. The output of the command will be ‘Hello World’.

$ echo "Hello World"

Open any editor to create a bash file. Here, nano editor is used to create the file and filename is set as ‘First.sh’

$ nano First.sh

Add the following bash script to the file and save the file.

#!/bin/bash

echo "Hello World"

You can run bash file by two ways. One way is by using bash command and another is by setting execute permission to bash file and run the file. Both ways are shown here.

$ bash First.sh

Or,

$ chmod a+x First.sh

$ ./First.sh

Use of echo command:

You can use echo command with various options. Some useful options are mentioned in the following example. When you use ‘echo’ command without any option then a newline is added by default. ‘-n’ option is used to print any text without new line and ‘-e’ option is used to remove backslash characters from the output. Create a new bash file with a name, ‘echo\_example.sh’ and add the following script.

#!/bin/bash

echo "Printing text with newline"

echo -n "Printing text without newline"

echo -e "\nRemoving \t backslash \t characters\n"

Run the file with bash command.

$ bash echo\_example.sh

Use of comment:

‘#’ symbol is used to add single line comment in bash script. Create a new file named ‘comment\_example.sh’ and add the following script with single line comment.

#!/bin/bash

# Add two numeric value

((sum=25+35))

#Print the result

echo $sum

Run the file with bash command.

$ bash comment\_example.sh

Use of Multi-line comment:

You can use multi line comment in bash in various ways. A simple way is shown in the following example. Create a new bash named, ‘multiline-comment.sh’ and add the following script. Here, ‘:’ and “ ’ ” symbols are used to add multiline comment in bash script. This following script will calculate the square of 5.

#!/bin/bash

: '

The following script calculates

the square value of the number, 5.

'

((area=5\*5))

echo $area

Run the file with bash command.

$ bash multiline-comment.sh

Using While Loop:

Create a bash file with the name, ‘while\_example.sh’, to know the use of while loop. In the example, while loop will iterate for 5 times. The value of count variable will increment by 1 in each step. When the value of count variable will 5 then the while loop will terminate.

#!/bin/bash

valid=true

count=1

while [ $valid ]

do

echo $count

if [ $count -eq 5 ];

then

break

fi

((count++))

done

Run the file with bash command.

$ bash while\_example.sh

Using For Loop:

The basic for loop declaration is shown in the following example. Create a file named ‘for\_example.sh’ and add the following script using for loop. Here, for loop will iterate for 10 times and print all values of the variable, counter in single line.

#!/bin/bash

for (( counter=10; counter>0; counter-- ))

do

echo -n "$counter "

done

printf "\n"

Run the file with bash command.

$ bash for\_example.sh

Get User Input:

‘read’ command is used to take input from user in bash. Create a file named ‘user\_input.sh’ and add the following script for taking input from the user. Here, one string value will be taken from the user and display the value by combining other string value.

#!/bin/bash

echo "Enter Your Name"

read name

echo "Welcome $name to LinuxHint"

Run the file with bash command.

$ bash user\_input.sh

Using if statement:

You can use if condition with single or multiple conditions. Starting and ending block of this statement is define by ‘if’ and ‘fi’. Create a file named ‘simple\_if.sh’ with the following script to know the use if statement in bash. Here, 10 is assigned to the variable, n. if the value of $n is less than 10 then the output will be “It is a one digit number”, otherwise the output will be “It is a two digit number”. For comparison, ‘-lt’ is used here. For comparison, you can also use ‘-eq’ for equality, ‘-ne’ for not equality and ‘-gt’ for greater than in bash script.

#!/bin/bash

n=10

if [ $n -lt 10 ];

then

echo "It is a one digit number"

else

echo "It is a two digit number"

fi

Run the file with bash command.

$ bash simple\_if.sh

Using if statement with AND logic:

Different types of logical conditions can be used in if statement with two or more conditions. How you can define multiple conditions in if statement using AND logic is shown in the following example. ‘&&’ is used to apply AND logic of if statement. Create a file named ‘if\_with\_AND.sh’ to check the following code. Here, the value of username and password variables will be taken from the user and compared with ‘admin’ and ‘secret’. If both values match then the output will be “valid user”, otherwise the output will be “invalid user”.

!/bin/bash

echo "Enter username"

read username

echo "Enter password"

read password

if [[ ( $username == "admin" && $password == "secret" ) ]]; then

echo "valid user"

else

echo "invalid user"

fi

Run the file with bash command.

$ bash if\_with\_AND.sh

Using if statement with OR logic:

‘||’ is used to define OR logic in if condition. Create a file named ‘if\_with\_OR.sh’ with the following code to check the use of OR logic of if statement. Here, the value of n will be taken from the user. If the value is equal to 15 or 45 then the output will be “You won the game”, otherwise the output will be “You lost the game”.

#!/bin/bash

echo "Enter any number"

read n

if [[ ( $n -eq 15 || $n -eq 45 ) ]]

then

echo "You won the game"

else

echo "You lost the game"

fi

Run the file with bash command.

$ bash if\_with\_OR.sh

Using else if statement:

The use of else if condition is little different in bash than other programming language. ‘elif’ is used to define else if condition in bash. Create a file named, ‘elseif\_example.sh’ and add the following script to check how else if is defined in bash script.

#!/bin/bash

echo "Enter your lucky number"

read n

if [ $n -eq 101 ];

then

echo "You got 1st prize"

elif [ $n -eq 510 ];

then

echo "You got 2nd prize"

elif [ $n -eq 999 ];

then

echo "You got 3rd prize"

else

echo "Sorry, try for the next time"

fi

Run the file with bash command.

$ bash elseif\_example.sh

Using Case Statement:

Case statement is used as the alternative of if-elseif-else statement. The starting and ending block of this statement is defined by ‘case’ and ‘esac’. Create a new file named, ‘case\_example.sh’ and add the following script. The output of the following script will be same to the previous else if example.

#!/bin/bash

echo "Enter your lucky number"

read n

case $n in

101)

echo echo "You got 1st prize" ;;

510)

echo "You got 2nd prize" ;;

999)

echo "You got 3rd prize" ;;

\*)

echo "Sorry, try for the next time" ;;

esac

Run the file with bash command.

$ bash case\_example.sh

Get Arguments from Command Line:

Bash script can read input from command line argument like other programming language. For example, $1 and $2 variable are used to read first and second command line arguments. Create a file named “command\_line.sh” and add the following script. Two argument values read by the following script and prints the total number of arguments and the argument values as output.

#!/bin/bash

echo "Total arguments : $#"

echo "1st Argument = $1"

echo "2nd argument = $2"

Run the file with bash command.

$ bash command\_line.sh Linux Hint

Get arguments from command line with names:

How you can read command line arguments with names is shown in the following script. Create a file named, ‘command\_line\_names.sh’ and add the following code. Here, two arguments, X and Y are read by this script and print the sum of X and Y.

#!/bin/bash

for arg in "$@"

do

index=$(echo $arg | cut -f1 -d=)

val=$(echo $arg | cut -f2 -d=)

case $index in

X) x=$val;;

Y) y=$val;;

\*)

esac

done

((result=x+y))

echo "X+Y=$result"

Run the file with bash command and with two command line arguments.

$ bash command\_line\_names X=45 Y=30

Combine String variables:

You can easily combine string variables in bash. Create a file named “string\_combine.sh” and add the following script to check how you can combine string variables in bash by placing variables together or using ‘+’ operator.

#!/bin/bash

string1="Linux"

string2="Hint"

echo "$string1$string2"

string3=$string1+$string2

string3+=" is a good tutorial blog site"

echo $string3

Run the file with bash command.

$ bash string\_combine.sh

Get substring of String:

Like other programming language, bash has no built-in function to cut value from any string data. But you can do the task of substring in another way in bash that is shown in the following script. To test the script, create a file named ‘substring\_example.sh’ with the following code. Here, the value, 6 indicates the starting point from where the substring will start and 5 indicates the length of the substring.

#!/bin/bash

Str="Learn Linux from LinuxHint"

subStr=${Str:6:5}

echo $subStr

Run the file with bash command.

$ bash substring\_example.sh

Add Two Numbers:

You can do the arithmetical operations in bash in different ways. How you can add two integer numbers in bash using double brackets is shown in the following script. Create a file named ‘add\_numbers.sh’ with the following code. Two integer values will be taken from the user and printed the result of addition.

#!/bin/bash

echo "Enter first number"

read x

echo "Enter second number"

read y

(( sum=x+y ))

echo "The result of addition=$sum"

Run the file with bash command.

$ bash add\_numbers.sh

Create Function:

How you can create a simple function and call the function is shown in the following script. Create a file named ‘function\_example.sh’ and add the following code. You can call any function by name only without using any bracket in bash script.

#!/bin/bash

function F1()

{

echo 'I like bash programming'

}

F1

Run the file with bash command.

$ bash function\_example.sh

Create function with Parameters:

Bash can’t declare function parameter or arguments at the time of function declaration. But you can use parameters in function by using other variable. If two values are passed at the time of function calling then $1 and $2 variable are used for reading the values. Create a file named ‘function|\_parameter.sh’ and add the following code. Here, the function, ‘Rectangle\_Area’ will calculate the area of a rectangle based on the parameter values.

#!/bin/bash

Rectangle\_Area() {

area=$(($1 \* $2))

echo "Area is : $area"

}

Rectangle\_Area 10 20

Run the file with bash command.

$ bash function\_parameter.sh

Pass Return Value from Function:

Bash function can pass both numeric and string values. How you can pass a string value from the function is shown in the following example. Create a file named, ‘function\_return.sh’ and add the following code. The function, greeting() returns a string value into the variable, val which prints later by combining with other string.

#!/bin/bash

function greeting() {

str="Hello, $name"

echo $str

}

echo "Enter your name"

read name

val=$(greeting)

echo "Return value of the function is $val"

Run the file with bash command.

$ bash function\_return.sh

Make Directory:

Bash uses ‘mkdir’ command to create a new directory. Create a file named ‘make\_directory.sh’ and add the following code to take a new directory name from the user. If the directory name is not exist in the current location then it will create the directory, otherwise the program will display error.

#!/bin/bash

echo "Enter directory name"

read newdir

`mkdir $newdir`

Run the file with bash command.

$ bash make\_directory.sh

Make directory by checking existence:

If you want to check the existence of directory in the current location before executing the ‘mkdir’ command then you can use the following code. ‘-d’ option is used to test a particular directory is exist or not. Create a file named, ‘directory\_exist.sh’ and add the following code to create a directory by checking existence.

#!/bin/bash

echo "Enter directory name"

read ndir

if [ -d "$ndir" ]

then

echo "Directory exist"

else

`mkdir $ndir`

echo "Directory created"

fi

Run the file with bash command.

$ bash directory\_exist.sh

Read a File:

You can read any file line by line in bash by using loop. Create a file named, ‘read\_file.sh’ and add the following code to read an existing file named, ‘book.txt’.

#!/bin/bash

file='book.txt'

while read line; do

echo $line

done < $file

Run the file with bash command.

$ bash read\_file.sh

Run the following command to check the original content of ‘book.txt’ file.

$ cat book.txt

Delete a File:

‘rm’ command is used in bash to remove any file. Create a file named ‘delete\_file.sh’ with the following code to take the filename from the user and remove. Here, ‘-i’ option is used to get permission from the user before removing the file.

#!/bin/bash

echo "Enter filename to remove"

read fn

rm -i $fn

Run the file with bash command.

$ ls

$ bash delete\_file.sh

$ ls

Append to File:

New data can be added into any existing file by using ‘>>’ operator in bash. Create a file named ‘append\_file.sh’ and add the following code to add new content at the end of the file. Here, ‘Learning Laravel 5’ will be added at the of ‘book.txt’ file after executing the script.

#!/bin/bash

echo "Before appending the file"

cat book.txt

echo "Learning Laravel 5">> book.txt

echo "After appending the file"

cat book.txt

Run the file with bash command.

$ bash append\_file.sh

Test if File Exist:

You can check the existence of file in bash by using ‘-e’ or ‘-f’ option. ‘-f’ option is used in the following script to test the file existence. Create a file named, ‘file\_exist.sh’ and add the following code. Here, the filename will pass from the command line.

#!/bin/bash

filename=$1

if [ -f "$filename" ]; then

echo "File exists"

else

echo "File does not exist"

fi

Run the following commands to check the existence of the file. Here, book.txt file exists and book2.txt is not exist in the current location.

$ ls

$ bash file\_exist.sh book.txt

$ bash file\_exist.sh book2.txt

Send Email:

You can send email by using ‘mail’ or ‘sendmail’ command. Before using these commands, you have to install all necessary packages. Create a file named, ‘mail\_example.sh’ and add the following code to send the email.

#!/bin/bash

Recipient=”admin@example.com”

Subject=”Greeting”

Message=”Welcome to our site”

`mail -s $Subject $Recipient <<< $Message`

Run the file with bash command.

$ bash mail\_example.sh

Get Parse Current Date:

You can get the current system date and time value using `date` command. Every part of date and time value can be parsed using ‘Y’, ‘m’, ‘d’, ‘H’, ‘M’ and ‘S’. Create a new file named ‘date\_parse.sh’ and add the following code to separate day, month, year, hour, minute and second values.

#!/bin/bash

Year=`date +%Y`

Month=`date +%m`

Day=`date +%d`

Hour=`date +%H`

Minute=`date +%M`

Second=`date +%S`

echo `date`

echo "Current Date is: $Day-$Month-$Year"

echo "Current Time is: $Hour:$Minute:$Second"

Run the file with bash command.

$ bash date\_parse.sh

Wait Command:

wait is a built-in command of Linux that waits for completing any running process. wait command is used with a particular process id or job id. If no process id or job id is given with wait command then it will wait for all current child processes to complete and returns exit status. Create a file named ‘wait\_example.sh’ and add the following script.

#!/bin/bash

echo "Wait command" &

process\_id=$!

wait $process\_id

echo "Exited with status $?"

Run the file with bash command.

$ bash wait\_example.sh

Sleep Command:

When you want to pause the execution of any command for specific period of time then you can use sleep command. You can set the delay amount by seconds (s), minutes (m), hours (h) and days (d). Create a file named ‘sleep\_example.sh’ and add the following script. This script will wait for 5 seconds after running.

#!/bin/bash

echo “Wait for 5 seconds”

sleep 5

echo “Completed”

Run the file with bash command.

$ bash sleep\_example.sh