

LIKE GEEKS



LINUX

Bash Script Step By Step, You Will Love It

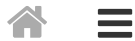
📅 February 7, 2017 👤 admin 💬 Comments(11)

Today we are going to talk about **bash script** or **shell scripting** actually, they are called **shell scripts** in general but we are going to call them bash scripts because we are going to use bash among the other Linux shells. There are zsh, tcsh , ksh and other shells, you can review the **basic Linux commands** before starting on bash script programming.

Our tutorial main points are

Bash shebang





Print messages

using variables

Environment variables

User variables

Command substitution

Math calculation

if-then statement

Nested if- Statement

Numeric Comparisons

String Comparisons

File Comparisons

So far you've seen how to use the bash shell to enter commands.

The key to bash script is the ability to enter multiple Commands and deal with the results from each command, even possibly passing the results of one command to another. The shell allows you to run multiple commands in a single step.

If you want to run two commands together or more, you can type them on the same line, separated with a semicolon

```
pwd ; whoami
```





This simple script uses just two bash shell commands.

The pwd command runs first, displaying the current working directory followed by the output of the whoami command, showing who is currently logged in user.

Using this technique, you can string together as many commands as you wish, maximum command line character count of arguments. You can determine your max args by this command

```
getconf ARG_MAX
```

Well, that's fine but it has a problem that you must enter the command at the command prompt every time you want to run it ok, what about we combine the commands into a file. And when we need to run those commands we run that file only. This is called the bash script

Now create an empty file using touch command as we discussed that in a previous post about **Linux commands**. The first line we should define which shell we will use as we know there are many shells on Linux bash is one of them

Bash Script Shebang

Here we will write bash script and in order to do that we will type

```
#!/bin/bash
```

In a normal bash script line, the pound sign (#) is used as a comment line which is not processed by the shell. However, the first line of a shell script file is a special case, and the pound sign followed by the exclamation point which is called shebang. That tells the shell what shell we will use which is bash in our case

And the shell commands are entered one per line followed by enter and you can write a comment by adding the pound sign at the beginning of the file like this





```
#!/bin/bash  
# This is a comment  
pwd  
whoami
```

You can use the semicolon and put multiple commands on the same

Line if you want to, but in bash script, you can list commands on separate lines, and this to make it simpler to read them later. The shell will process them any way.

Set Script Permission

Save the file, you are almost finished. All you need to do now is to set that file to be executable in order to be able to run it otherwise it will give you permissions denied. You can review the permissions commands on our post

```
likegeeks@likegeeks-VirtualBox ~/Desktop  
File Edit View Search Terminal Help  
likegeeks@likegeeks-VirtualBox ~/Desktop $ ./myscript  
bash: ./myscript: Permission denied  
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

```
chmod +x ./myscript
```

Then try run it by just type it in the shell

```
./myscript
```

And Yes it is executed





```
likegeeks@likegeeks-VirtualBox ~/Desktop $ chmod +x ./myscript
likegeeks@likegeeks-VirtualBox ~/Desktop $ ./myscript
/home/likegeeks/Desktop
likegeeks
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Print Messages

As we know from other posts, printing text is done by echo command

So take your knowledge to bash script and apply it

Now we edit our file and type this

```
#!/bin/bash
# our comment is here
echo "The current directory is:"
pwd
echo "The user logged in is:"
whoami
```

Look at the output

```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ ./myscript
The current directory is:
/home/likegeeks/Desktop
The user logged in is:
likegeeks
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Perfect! Now we can run commands and display text using echo command

If you don't know echo command or how to edit a file I recommend you to view previous articles about **basic Linux commands**





Variables allow you to store information in the bash script for use with other commands in the script.

Running individual commands from the bash script is good, but this has its limitations.

There are two types of variables you can use in your bash script

- Environment variables
- User variables

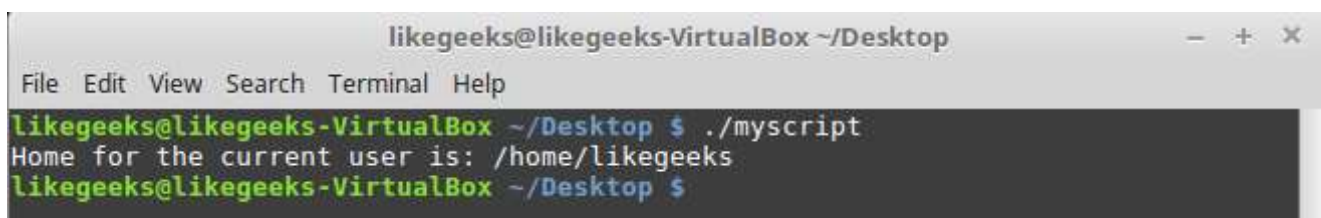
Environment Variables

Sometimes you need to interact with other data in your shell commands to process information. You can do this by using environment variables.

We talked about environment variables on another post you can check it.

```
#!/bin/bash
# display user home
echo "Home for the current user is: $HOME"
```

Notice that we were able to place the \$HOME system variable in the double quotation marks in the first string, and the shell script still know it



```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ ./myscript
Home for the current user is: /home/likegeeks
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

What if we want to print the dollar sign itself?

```
echo "I have $1 in my pocket"
```





attempted to display the variable \$1 which was not defined so how to overcome that?

By using escape characters back slash \ before the dollar sign like this

```
echo "I have \$1 in my pocket"
```

Now the bash script will print the dollar sign as it is

```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ ./myscript
I have $1 on my pocket
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

User variables

In addition to the environment variables, a bash script allows you to set and use your own variables in the script.

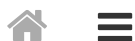
Variables defined in the shell script maintain their values till bash script execution finished.

Like system variables, user variables can be referenced using the dollar sign

```
#!/bin/bash
# testing variables
grade=5
person="Adam"
echo "$person is a good boy, he is in grade $grade"
```

```
chmod +x myscript
./myscript
```





```
likegeeks@likegeeks-VirtualBox ~/Desktop $ ./myscript
Adam is a good boy, he is in grade 5
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Command substitution

One of the best features of bash scripts is the ability to extract information from the output of a command and assign it to a variable so you can use that value anywhere in your script

There are two ways to do that

- The backtick character `
- The `$()` format

Make sure when you type backtick character it is not the single quotation mark.

You must surround the entire command line command with two backtick characters like this

```
mydir=`pwd`
```

or the other way

```
mydir=$(pwd)
```

so the script could be like this

```
#!/bin/bash
mydir=$(pwd)
echo $mydir
```

The output of the command will be stored in that variable called mydir.

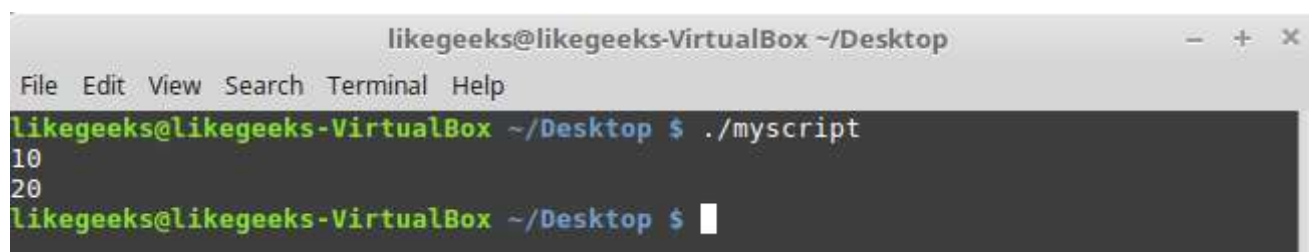
```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ ./myscript
/home/likegeeks/Desktop
likegeeks@likegeeks-VirtualBox ~/Desktop $
```




You can perform basic math calculations using `[]` format

```
#!/bin/bash
var1=$(( 5 + 5 ))
echo $var1
var2=$(( $var1 * 2 ))
echo $var2
```

Just that easy



```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ ./myscript
10
20
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

if-then statement

Bash script requires some sort of logic flow control between the commands in the script. Like if the value is greater than 5 do that else do whatever you can imagine any logic you want.

The most basic structure of if-then statement is like this

if *command*

then

commands

fi

and here is an example

```
#!/bin/bash
```



```
then  
echo "It works"  
fi
```

If the command completes successfully, the echo statement should display the text.

Let's dig deeper and use other commands we know together.

May be searching for a specific user in the users /etc/passwd and if it exists it prints that the user is present

```
#!/bin/bash  
user=likegeeks  
if grep $user /etc/passwd  
then  
echo "The user $user Exists"  
fi
```

```
likegeeks@likegeeks-VirtualBox ~/Desktop  
File Edit View Search Terminal Help  
likegeeks@likegeeks-VirtualBox ~/Desktop $ ./myscript  
likegeeks:x:1000:1000:likegeeks,,,:/home/likegeeks:/bin/bash  
The user likegeeks Exists  
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

We use grep command to search for the user in /etc/passwd file. You can check our tutorial about basic Linux commands if you don't know grep command.

If the user exists the bash script will print the message.

What if the user doesn't exist? The script will exit the execution without telling us that the user doesn't exist ok let's improve the script more

if-then-else Statement





if *command*

then

commands

else

commands

fi

if the first command runs and returns with a zero which means success it will not hit the commands after the else statement, otherwise, if the if statement returns non-zero which means the statement condition not correct, At this case the shell will hit the commands after else statement.

```
#!/bin/bash
user=anotherUser
if grep $user /etc/passwd
then
echo "The user $user Exists"
else
echo "The user $user doesn't exist"
fi
```

```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ ./myscript
The user anotherUser doesn't exist
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

We are doing good till now, keep moving





other print this else print another thing?

Well that is easy also we can achieve by nesting if statements like this

```
if command1
```

```
then
```

```
commands
```

```
elif command2
```

```
then
```

```
commands
```

```
fi
```

If the first command return zero means success it will execute the commands after it else if the second command return zero it will execute the commands after it else if none of those return zero it will execute the last commands only.

```
#!/bin/bash
user=anotherUser
if grep $user /etc/passwd
then
echo "The user $user Exists"
elif ls /home
echo "The user doesn't exist but anyway there is a directory unde
fi
```

You can imagine any scenario here may be if the user doesn't exist create it using the `useradd` command or do anything else.



You can perform numeric comparison between two numeric values using numeric comparison checks as on this list

`n1 -eq n2` Checks if n1 is equal to n2

`n1 -ge n2` Checks if n1 is greater than or equal to n2

`n1 -gt n2` Checks if n1 is greater than n2

`n1 -le n2` Checks if n1 is less than or equal to n2

`n1 -lt n2` Checks if n1 is less than n2

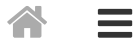
`n1 -ne n2` Checks if n1 is not equal to n2

As an example, we will try one of them and the rest is the same

Note that the comparison statement is in square brackets as shown.

```
#!/bin/bash
val1=6
if [ $val1 -gt 5 ]
then
echo "The test value $value1 is greater than 5"
else
echo "The test value $value1 is not greater than 5"
fi
```

```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ ./myscript
The test value is greater than 5
likegeeks@likegeeks-VirtualBox ~/Desktop $
```



String Comparisons

Performing comparisons on strings are so easy. The comparison functions you can use to evaluate two string values are

`str1 = str2` Checks if str1 is the same as string str2

`str1 != str2` Checks if str1 is not the same as str2

`str1 < str2` Checks if str1 is less than str2

`str1 > str2` Checks if str1 is greater than str2

`-n str1` Checks if str1 has a length greater than zero

`-z str1` Checks if str1 has a length of zero

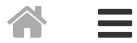
We can apply string comparison on our example

```
#!/bin/bash
user="likegeeks"
if [$user = $USER]
then
echo "The user $user is the current logged in user"
fi
```

```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ ./myscript
The user likegeeks is the current logged in user
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

One tricky note about the **greater than and less than** for string comparisons **MUST** be **escaped with the back slash** because by just using the greater-than symbol itself in the





than symbol as an output redirection. So you should do it like that

```
#!/bin/bash
val1=text
val2="another text"
if [ $val1 \> "$val2" ]
then
echo "$val1 is greater than $val2"
else
echo "$val1 is less than $val2"
fi
```

```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ ./myscript
./myscript: line 5: [: too many arguments
text is less than another text
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

It runs but if gives this warning

```
./myscript: line 5: [: too many arguments
```

Just little fix is to wrap the \$vals with double quotation, forcing it to stay as one string like this

```
#!/bin/bash
val1=text
val2="another text"
if [ "$val1" \> "$val2" ]
then
echo "$val1 is greater than $val2"
else
echo "$val1 is less than $val2"
```



```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ ./myscript
text is greater than another text
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Last tricky note about **greater than and less than** for string comparisons is when working with **uppercase and lowercase** letters. The **sort command** handles uppercase letters **opposite** to the way the **test conditions** consider them

```
#!/bin/bash
val1=Likegeeks
val2=likegeeks
if [ $val1 \> $val2 ]
then
echo "$val1 is greater than $val2"
else
echo "$val1 is less than $val2"
fi
```

```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ ./myscript
Likegeeks is less than likegeeks
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

```
sort myfile
```

likegeeks

Likegeeks





```
likegeeks@likegeeks-VirtualBox ~/Desktop $ ./myscript
Likegeeks is less than likegeeks
likegeeks@likegeeks-VirtualBox ~/Desktop $ sort ./myfile
likegeeks
Likegeeks
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Capitalized letters are treated as less than lowercase letters in test comparisons. However, the sort command does exactly the opposite.

String test comparisons use standard ASCII ordering, which means using each character's ASCII numeric value to determine the sort order.

The sort command uses the sorting order defined for the system locale language settings.

File Comparisons

This is the best and most powerful and most used comparison in bash scripting there are many file comparisons that you can do in bash script

-d file Checks if file exists and is a directory

-e file Checks if file exists

-f file Checks if file exists and is a file

-r file Checks if file exists and is readable

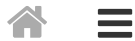
-s file Checks if file exists and is not empty

-w file Checks if file exists and is writable

-x file Checks if file exists and is executable

file1 -nt file2 Checks if file1 is newer than file2





-O file Checks if file exists and is owned by the current user

-G file Checks if file exists and the default group is the same as the current user

As they imply, you will never forget them.

Let's pick one of them and take it as an example

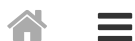
```
#!/bin/bash
mydir=/home/likegeeks
if [ -d $mydir ]
then
echo "The $mydir directory exists"
cd $ mydir
ls
else
echo "The $mydir directory does not exist"
fi
```

The screenshot shows a terminal window titled 'likegeeks@likegeeks-VirtualBox ~/Desktop'. The terminal has a menu bar with 'File', 'Edit', 'View', 'Search', 'Terminal', and 'Help'. The prompt is 'likegeeks@likegeeks-VirtualBox ~/Desktop \$'. The user has entered './myscript', and the output is 'The /home/likegeeks directory exists'. Below the output, a directory listing is shown: 'book.pdf Documents likegeeks myfile myfile3 myfile5 nohup.out Templates Desktop Downloads Music myfile2 myfile4 newfile Public Videos'. The prompt is now 'likegeeks@likegeeks-VirtualBox ~/Desktop \$'.

We are not going to type every one of them as an example but you got the idea. You just type the comparison between the square brackets as it is and complete your script normally.

There are some other advanced if-then features but let's make it on another post.





Wait for another tutorial about bash scripting so stay tuned.

Bash scripting part2

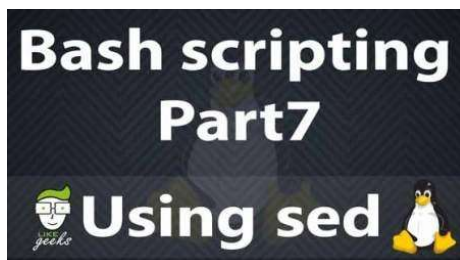
Thanks.

13

Admin

<https://likegeeks.com>

RELATED ARTICLES



LINUX

31+ Examples for sed Linux command in text manipulation

📅 February 19, 2017 👤 admin

On the previous post we've talked about bash functions and how to use it from the command line and we've seen some other cool stuff I



LINUX

Learn Linux Environment Variables Step-By-Step Easy Guide

📅 February 4, 2017 👤 admin

On the previous posts we talked about some of the basic Linux commands and today we continue our journey and we will talk about



LINUX

Main Linux Commands Easy Guide

📅 January 31, 2017 👤 admin

Main Linux Commands That You Should Know As A Beginner On previous post we discussed how to install Linux now we are going to talk about the most powerful



Today we will talk about a very useful tool for string manipulation called sed, sed Linux command is one of the most common tools [...]

Linux which is Environment Variables So what are Environment Variables and where it is and the benefit of knowing it? Well the bash shell we use to [...]

commands or shell commands for the whole documentation of Linux Commands, you can check Linux Documentation if you will use Linux [...]

0

0

0

◀ Clipboard actions copy paste in windows

Bash scripting the awesome guide Part2 ▶

Pingback: Bash Script Step By Step Awesome Guide – TechNow()

- Pingback: Bash scripting the awesome guide Part2 - Like Geeks()
- Pingback: Bash Scripting The Awesome Guide Part2 – TechNow()
- Pingback: Linux bash scripting the awesome guide part3 - Like Geeks()
- *jamesapeltier*

it is advisable to use printf in place of echo since echo behaves differently depending on platform and version. For example echo on some platforms includes a carriage return, while on others it doesn't. When in doubt, always printf.

- *likegeeks*

using printf introduces formatting and some other cool stuff that I'm going to discuss in upcoming posts.

And you are correct it behaves little different but I want to keep the material simple as possible then go the next level 😊

- Pingback: Linux Bash Scripting The Awesome Guide Part3 – TechNow()
- Pingback: Bash scripting, step by step (1) | Oddn1x: tricks with *nix()

