

# Introduction to Bash Scripting

Get acquainted with bash scripting in this mega tutorial for beginners.



Abhishek Prakash

14 Apr 2024 · 10 min read · 2 Comments

## FEATURED VIDEOS

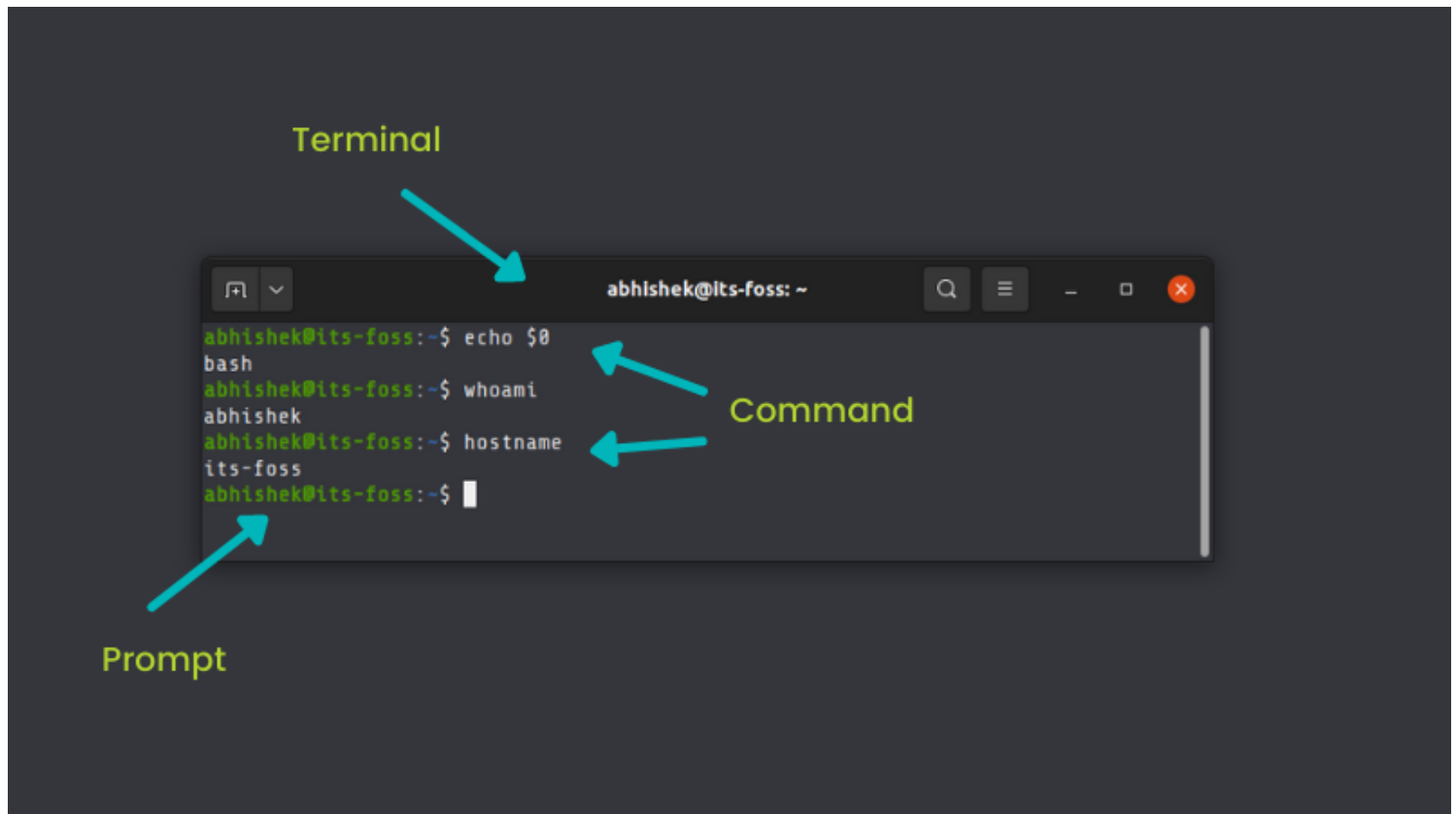
Powered by **[primis]**

Shell is the core part of Linux. It allows you to interact with the Linux kernel by using various commands like cd, ls, cat etc.

Bash is one of the many available shells for Linux. They have mostly common but not identical syntax. Bash is the most popular shell out there and is the default on most Linux distributions.

You open a terminal or SSH session and you have a shell running in it even if you don't visualize it.

Membership



When you type a command, it is interpreted by the shell. If the command and syntax are correct, it will be executed otherwise you'll see an error.

## Why bash scripts when you can just run Linux commands?

You can enter the commands directly in the terminal and they will be executed.

```
abhishek@itsfoss:~$ echo "hello world"
hello world
```

And the same can be done in a script as well:

```
abhishek@itsfoss:~$ cat >> script.sh
#!/bin/bash
```

```
echo "hello world"  
abhishek@itsfoss:~$ bash script.sh  
hello world
```

Why do you need shell scripts then? Because you don't have to type the same command again and again. You just run the shell script.

Also, if you have complicated logic in your script, typing it all in the terminal won't be a good idea.

For example, if you enter the command below, it will work. But it is not easy to understand and typing it again and again (or even searching for it in the bash history) is a pain.

```
if [ $(whoami) = 'root' ]; then echo "root"; else echo "not root"; fi
```

Instead, you can put in a shell script so that it is easier to understand and run it effortlessly:

```
#!/bin/bash  
  
if [ $(whoami) = 'root' ]; then  
    echo "You are root"  
else  
    echo "You are not root"  
fi
```



This was still simple. Imagine a complicated script with fifty or a hundred lines!

## What will you learn?

There are nine sections in this bash scripting tutorial. You'll learn to:

- Create and run your first bash shell script
- Use variables
- Pass arguments and accept user inputs in your bash scripts
- Perform mathematical calculations
- Manipulate strings
- Use conditional statements like if-else
- Use for, while and until loops
- Create functions

All the sections will give you a brief example. If you wish, you can extend on the section by visiting the detailed chapter for each section. These chapters also contain practice exercises.

## 1. Writing your first bash shell script

Create a new file named `hello.sh`:

```
nano hello.sh
```

This will open the nano editor in the terminal. Enter the following lines to it:

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

```
echo "Hello World"
```





Save and exit the nano editor by pressing the Ctrl+X key.

Now, you can run the bash shell script in the following manner:

```
bash hello.sh
```

And you should see the following output:

```
Hello World
```

Another way is to give the script execute permission first:

```
chmod u+x hello.sh
```

And then run it in this manner:

```
./hello.sh
```



writing longer scripts. However, you need to switch to the directory where the script is saved to run it.

Congratulations! You just ran your first bash script.



### Bash Basics #1: Create and Run Your First Bash Shell Script

Start learning bash scripting with this new series. Create and run your first bash shell script in the first chapter.

 It's FOSS • Abhishek Prakash

## 2. Using variables in bash scripts

Variables are declared in the following manner:

```
var=some_value
```





And then the variable is accessed like this:

```
$var
```

- ❑ There must not be a space before and after `=` while declaring variable.

Let's modify the previous script by adding a variable.

```
#!/bin/bash  
  
message="Hello World"  
  
echo $message
```

The output will still remain the same if you run this script:

```
Hello World
```





## Bash Basics #2: Use Variables in Bash Scripts

In this chapter of the Bash Basics series, learn about using variables in Bash scripts.

 It's FOSS • Abhishek Prakash

### 3. Passing arguments to bash script

You can pass arguments to a bash script while running it in the following manner:

```
./my_script.sh arg1 arg2
```

Inside the script, you can use \$1 for the 1st argument, \$2 for the 2nd argument and so on. \$0 is a special variable that holds the name of the script being executed.

Now, create a new shell script named `arguments.sh` and add the following lines to it:

```
#!/bin/bash

echo "Script name is: $0"
echo "First argument is: $1"
echo "Second argument is: $2"
```

Make it executable and r





```
abhishek@itsfoss:~$ ./argument.sh abhishek prakash
Script name is: ./argument.sh
First argument is: abhishek
Second argument is: prakash
```

Here's a quick look at the special variables:

Special Variable	Description
\$0	Script name
\$1, \$2...\$9	Script arguments
\${n}	Script arguments from 10 to 255
\$#	Number of arguments
\$@	All arguments together
\$\$	Process id of the current shell
\$_	Process id of the last executed command
\$_	Exit status of last executed command

You can also make your bash script interactive by accepting user input from the keyboard.



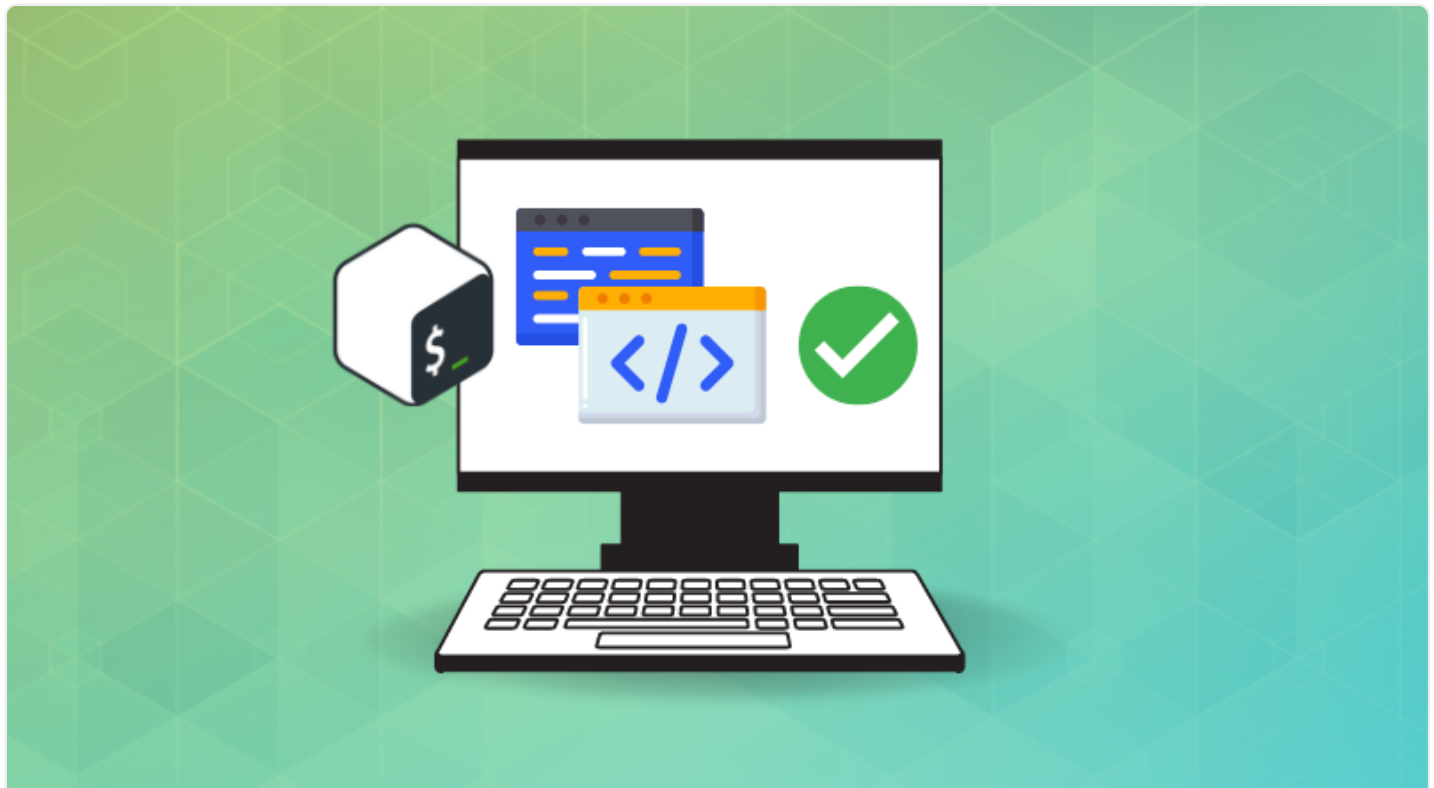
For that, you'll have to use the read command. You can also use `read -p` command to prompt the user for the keyboard input without echo command.

```
#!/bin/bash

echo "What is your name, stranger?"
read name
read -p "What's your full name, $name? " full_name
echo "Welcome, $full_name"
```

Now if you run this script, you'll have to enter the 'arguments' when you are prompted for it.

```
abhishek@itsfoss:~$ ./argument.sh
What is your name, stranger?
abhishek
What's your full name, abhishek? abhishek prakash
Welcome, abhishek prakash
```



Learn how to pass arguments to bash scripts and make them interactive in this chapter of the Bash Basics series.

 It's FOSS • Abhishek Prakash

## 4. Perform arithmetic operation

The syntax for arithmetic operations in the bash shell is this:

```
$( arithmetic_operation )
```

Here's the list of the arithmetic operations you can perform in bash:

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Integer division (without decimal)
%	Modulus division (only remainder)
**	Exponentiation (a to the power b)

Here's an example of performing summation and subtraction in bash script:

```
#!/bin/bash

read -p "Enter first number: " num1
read -p "Enter second number: " num2
```

```
sum=$(( $num1+$num2 ))
```



```
sub=$(( $num1-$num2 ))  
echo "The summation of $num1 and $num2 is $sum"  
echo "The subtraction of $num2 from $num1 is $sub"
```

You can execute the shell script with random number of your choice.

There is a big problem if you try the division. Bash only works with integers. It doesn't have the concept of decimal numbers by default. And thus you'll get 3 as the result of 10/3 instead of 3.333.

For floating point operations, you'll have to use the bc command in this manner:



```
#!/bin/bash

num1=50
num2=6

result=$(echo "$num1/$num2" | bc -l)

echo "The result is $result"
```

And this time, you'll see accurate result.

```
The result is 8.33333333333333333333
```



## Bash Basics Series #4: Arithmetic Operations

In the fourth chapter of the series, learn to use basic mathematics in Bash.

 It's FOSS • Abhishek Prakash



## 5. Using arrays in bash scripts

Instead of using multiple variables, you can use arrays in bash to store values in the same category.

You can declare an array like this:

```
distros=(Ubuntu Fedora SUSE "Arch Linux" Nix)
```

To access an element, use:

```
${array_name[N]}
```

Like most other programming languages, the array index starts at 0.

You can display all the elements of an array like this:

```
${array[*]}
```

And get the array length like this:

```
${#array_name[@]}
```





## Bash Basics Series #5: Using Arrays in Bash

Time to use arrays in bash shell scripts in this chapter. Learn to add elements, delete them and get array length.

 It's FOSS • Abhishek Prakash

## 6. Basic string operation in Bash

Bash is capable of performing a number of string operations.

You can get the string length in this manner:

```
${#string}
```

Join two strings:

```
str3=$str1$str2
```



Extract a substring by providing the starting position of the substring and its length:

```
${string:$pos:$len}
```

Here's an example:

You can also replace a portion of the given string:

```
${string/substr1/substr2}
```

And you can also delete





```
`${string/substring}`
```



## Bash Basics Series #6: Handling String Operations

In this chapter of the Bash Basics series, learn to perform various common string operations like extracting, replacing and deleting substrings.

 It's FOSS • Abhishek Prakash

## 7. Use conditional statements in Bash

You can add conditional logic to your bash scripts by using if or if-else statements. These statements end with `fi`.

The syntax for a single if statement is:

```
if [ condition ]; then  
    your code  
fi
```



Pay attention to the use `[ ... ];` and `then` .

The syntax for if-else statement is:

```
if [ expression ]; then
    ## execute this block if condition is true else go to next
elif [ expression ]; then
    ## execute this block if condition is true else go to next
else
    ## if none of the above conditions are true, execute this block
fi
```

Here's a sample bash script that uses if-else statement:

```
#!/bin/bash
read -p "Enter the number: " num
```



```
mod=$(( $num%2 ))

if [ $mod -eq 0 ]; then
    echo "Number $num is even"
else
    echo "Number $num is odd"
fi
```

Run it and you should see a result like this:

The `-eq` is called test condition or conditional operator. There are many such operators to give you different types of comparison:

Here are the test condition operators you can use for numeric comparison:

Condition	Equivalent to true when
<code>\$a -lt \$b</code>	<code>\$a &lt; \$b</code> ( <code>\$a</code> is less than <code>\$b</code> )
<code>\$a -gt \$b</code>	<code>\$a &gt; \$b</code> ( <code>\$a</code> is greater than <code>\$b</code> )
<code>\$a -le \$b</code>	<code>\$a &lt;= \$b</code> ( <code>\$a</code> is less or equal than <code>\$b</code> )



Condition	Equivalent to true when
<code>\$a -ge \$b</code>	<code>\$a &gt;= \$b</code> ( <code>\$a</code> is <b>g</b> reater or <b>e</b> qual than <code>\$b</code> )

If you are comparing strings, you can use these test conditions:

Condition	Equivalent to true when
<code>"\$a" = "\$b"</code>	<code>\$a</code> is same as <code>\$b</code>
<code>"\$a" == "\$b"</code>	<code>\$a</code> is same as <code>\$b</code>
<code>"\$a" != "\$b"</code>	<code>\$a</code> is different from <code>\$b</code>
<code>-z "\$a"</code>	<code>\$a</code> is empty

There are also conditions for file type check:

Condition	Equivalent to true when
<code>-f \$a</code>	<code>\$a</code> is a file
<code>-d \$a</code>	<code>\$a</code> is a directory
<code>-L \$a</code>	<code>\$a</code> is a link

- Pay special attention to space. There must be space between the opening and closing brackets and the conditions. Similarly, space must be before and after the conditional operators (`-le`, `==` etc).





## Bash Basics Series #7: If Else Statement

If this, then that else something else. Doesn't make sense? It will after you learn about the if-else statements in bash shell scripting.



It's FOSS • Abhishek Prakash

## 8. Using loops in bash scripts

Bash support three types of loops: for, while and until.

Here's an example of the **for loop**:

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

```
for num in {1..10}
```



```
echo $num
done
```

Run it and you'll see the following output:

```
1
2
3
4
5
6
7
8
9
10
```

If you take the previous example, it can be rewritten using the **while loop** like this:

```
#!/bin/bash

num=1
while [ $num -le 10 ]; do
    echo $num
    num=$(( $num + 1 ))
done
```

And the same can be rewritten using the **until loop**:

```
#!/bin/bash

num=1
until [ $num -gt 10 ]; do
    echo $num
    num=$(( $num + 1 ))
done
```



- The while and until loop are pretty similar. The difference is that while loop runs as long as the condition is true and until loop runs as long as the condition is false.



## Bash Basics #8: For, While and Until Loops

In the penultimate chapter of the Bash Basics series, learn about for, while and until loops.

 It's FOSS • Abhishek Prakash

## 9. Using functions in bash scripts

Bash shell does support the use of functions so that you don't have to write the same piece of code again and again.

Here's the generic syntax for declaring a bash function:

```
function_name() {
```



```
commands
```

```
}
```

Here's a sample bash script that uses function with arguments:

```
#!/bin/bash

sum() {
    sum=$(( $1 + $2 ))
    echo "The sum of $1 and $2 is: $sum"
}

echo "Let's use the sum function"
sum 1 5
```

If you run the script, you'll see the following output:

```
Let's use the sum function
The sum of 1 and 5 is: 6
```







## Bash Basics Series #9: Functions in Bash

Learn all about functions in the final chapter of the Bash Basics series.

 It's FOSS • Abhishek Prakash

## Where to go from here?

This is just a glimpse. This bash scripting tutorial is just a primer. There is a lot more to bash scripting and you can explore it slowly and gradually.

The GNU bash reference is an excellent online resource to clear your bash doubts.

### Bash Reference Manual

Bash Reference Manual



##Apart from that, you can download this free book to learn more bash stuff that is not covered here:

[Download Bash Beginner Guide](#)

Once you have enough k

 ed bash scripting with

this free book:

[Download Adavanced Bash Scripting Guide](#)

Both of these books are at least a decade old but you can still use them to learn bash.

I hope you like this tutorial as the starting point of your bash script learning. Please provide your feedback in the comments section.

Guides



## ABOUT THE AUTHOR



### Abhishek Prakash

Created It's FOSS 11 years ago to share my Linux adventures. Have a Master's degree in Engineering and years of IT industry experience. Huge fan of Agatha Christie detective mysteries



## Featured

[Teaching Linux to Software Developers With This Book](#)





## 7 Sudo Tips and Tweaks for Linux Users

## 11 YouTube Channels Linux Users Should Explore



### Latest

## Switching Between Intel and Nvidia Graphics Cards on Ubuntu

15 May 2024



## Testing the Surfshark VPN GUI App on Linux

14 May 2024

## Finding the Fastest Arch Linux Mirrors

12 May 2024

### Become a Better Linux User







With the FOSS Weekly Newsletter, you learn useful Linux tips, discover applications, explore new distros and stay updated with the latest from Linux world

**Subscribe**



What's your reaction?



 13  
  3  
  0  
  1  
  0  
  1

superb   love   wow   sad   laugh   angry

## 2 Comments

12 ONLINE

LOGIN

We welcome and encourage civil, courteous, and community-friendly comments.

Write your comment...

Newest



Dharrun Singh 8 months ago

LOVED BY ITSFOSS

Helpful! Appreciate it a lot!

0 0 Reply

Abhishek **Plus Member** 8 months ago

Glad you liked it :)

0 0 Reply

## READ NEXT

Complete Guide to Installing Linux on Chromebook >

How To Install and Use Conky in Ubuntu Linux >

Getting Started With Ubuntu >

How to Check if an Array is empty in Bash? >

Understanding Ubuntu's Repository System >



# Become a Better Linux User

With the FOSS Weekly Newsletter, you learn useful Linux tips, discover applications, explore new distros and stay updated with the latest from Linux world

**SUBSCRIBE**

Making You a Better Linux User

**Subscribe**

## Navigation

- News
  - Newsletter
  - Quizzes & Puzzles
  - Resources
  - Community
  - About
  - Contact
  - Policies
- Linux Server Side
  - En Español
  - Feedback





## Resources

[Distro Resources](#) □

[Guides](#) □

[Courses](#) □

## Social

[Facebook](#)

[Twitter](#)

[RSS](#)

[Instagram](#)

[Telegram](#)

[Youtube](#)

---

©2024 It's FOSS. Hosted on Digital Ocean & Published with Ghost & Rinne.

[System](#) ⌵

