

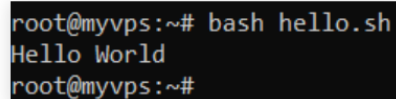
1. Hello World

Hello World is the most simple bash script to start with. We will create a new variable called **learningbash** and print out the words **Hello World**. First, open a new shell script file with a text editor of your choice:

```
nano hello.sh
```

Paste the following lines into it:

```
#!/bin/bash
#Creates a new variable with a value of "Hello World"
learningbash="Hello World"
echo $learningbash
```



```
root@myvps:~# bash hello.sh
Hello World
root@myvps:~#
```

The first line (**/bin/bash**) is used in every bash script. It instructs the operating system to use a bash interpreter as a command interpreter.

2. Echo Command

The **echo** bash command can be used to print out text as well as values of variables. In the following example, we will showcase how quotation marks affect the echo command. We will start by opening a new bash script file:

```
nano echo.sh
```

This simple bash script example will create a new variable and print it out while using different quotation marks.

```
#!/bin/bash
provider="Hostinger"
echo 'The best hosting provider is $provider'
echo "The best hosting provider is $provider"
```

```
root@myvps:~# bash echo.sh
The best hosting provider is $provider
The best hosting provider is Hostinger
root@myvps:~#
```

As you can see, if the echo bash command is used with double quotation marks “”, then the script will print out the actual value of a variable. Otherwise, if the single quotation marks “” are used, it will print out only the name of a variable.

5. Comments

Users can easily add comments to their bash scripts with the # symbol. It is extra useful if you've got a lengthy script that needs explaining on some lines.

Begin by creating a new bash script:

```
nano comments.sh
```

Then paste in the following:

```
#!/bin/bash
# Define a variable named Hostinger
provider="Hostinger"
# Print out the following text
echo 'The best hosting provider is $provider'
# Print out the following text with $provider variable value
echo "The best hosting provider is $provider"
```

```
root@myvps:~# bash comments.sh
The best hosting provider is $provider
The best hosting provider is Hostinger
root@myvps:~#
```

Keep in mind that bash comments are only visible on a text editor.

6. Get User Input

To take input from users, we'll use the **read** bash command. First, create a new bash shell file:

```
nano read.sh
```

Then, fill it with the script below:

```
#!/bin/bash
echo "What is your age?"
read age
echo "Wow, you look younger than $age years old"
```

In the above example, an age value was entered by the user. The output was then printed via the echo command.

7. Loops

A loop is an essential tool in various programming languages. To put it simply, a **bash loop** is a set of instructions that are repeated until a user-specified condition is reached. Start by creating a loop bash program:

```
nano whileloop.sh
```

Then paste in the following:

```
#!/bin/bash
n=0
while :
do
echo Countdown: $n
((n++))
done
```

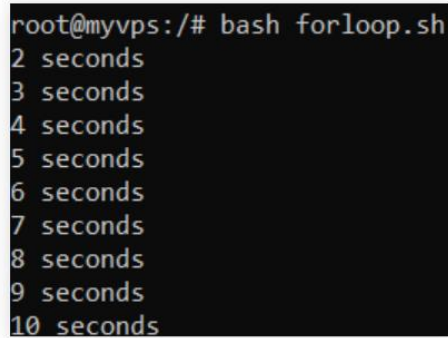
This will work as a countdown to infinity until you press **CTRL + C** to stop the script.

Now that we've tested the while loop, we can move on to the for loop. Create a bash file for it:

```
nano forloop.sh
```

It should contain the script below:

```
#!/bin/bash
for (( n=2; n<=10; n++ ))
do
echo "$n seconds"
done
```

A terminal window with a black background and white text. The prompt is 'root@myvps:/#'. The user has entered 'bash forloop.sh'. The output shows a list of numbers from 2 to 10, each followed by the word 'seconds' on a new line.

```
root@myvps:/# bash forloop.sh
2 seconds
3 seconds
4 seconds
5 seconds
6 seconds
7 seconds
8 seconds
9 seconds
10 seconds
```

The script prints out numbers from 2 to 10 while adding the **seconds** keyword to it.

8. Create an Array

A **bash array** is a data structure designed to store information in an indexed way. It is extra useful if users need to store and retrieve thousands of pieces of data fast. What makes bash arrays special is that unlike any other programming language, they can store different types of elements. For example, you can use a bash array to store both strings and numbers.

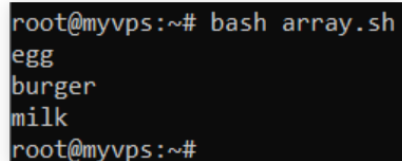
Create a new file in the current directory:

```
nano array.sh
```

Combine the freshly learned **for** loop with a new indexed array:

Combine the freshly learned **for** loop with a new indexed array:

```
#!/bin/bash
# Create an indexed array
IndexedArray=(egg burger milk)
#Iterate over the array to get all the values
for i in "${IndexedArray[@]}";do echo "$i";done
```

A terminal window with a black background and white text. The prompt is 'root@myvps:~#'. The user enters 'bash array.sh'. The script outputs three lines: 'egg', 'burger', and 'milk'. The prompt returns to 'root@myvps:~#'.

```
root@myvps:~# bash array.sh
egg
burger
milk
root@myvps:~#
```

The script iterates over the **IndexedArray** and prints out all the values.

9. Conditional Statements

The most popular and widely used conditional statement is **if**. Even though the **if** statement is easy to write and understand, it can be used in advanced shell scripts as well.

Begin with a new bash file:

```
nano if.sh
```

Paste the code below in it:

```
#!/bin/bash
salary=1000
expenses=800
#Check if salary and expenses are equal
if [ $salary == $expenses ];
then
    echo "Salary and expenses are equal"
#Check if salary and expenses are not equal
elif [ $salary != $expenses ];
then
    echo "Salary and expenses are not equal"
fi
```

This script creates two new variables and compares whether they are equal or not.

10. Functions

A bash function is a set of commands that can be reused numerous times throughout a bash script. Create a new file:

```
nano function.sh
```

Then, paste in the following code – it creates a simple Hello World function.

```
#!/bin/bash
hello () {
    echo 'Hello World!'
}
hello
```

11. Display String Length

There are a couple of ways of counting string length in bash. We'll talk about the simplest. Create a file named **stringlength.sh**:

```
nano stringlength.sh
```

Fill it with the following:

```
#!/bin/bash
# Create a new string
mystring="lets count the length of this string"
i=${#mystring}
echo "Length: $i"
```

Here, the **#** operator is used to get the length of the string variable.

12. Extract String

If users need to remove unnecessary parts from strings, they can use the Bash string extraction tools. Start by creating a new bash script:

```
nano extractstring.sh
```

The following script has 4 values, 3 of them being strings. In our example, we will extract only the number value. This can be done via the **cut** command. First, we instruct the command that each variable is separated by a comma by using the **-d** flag. Then we ask the cut command to extract the 5th value.

```
#!/bin/bash
cut -d , -f 5 <<< "Website,Domain,DNS,SMTP,5005"
```

In another example, we have a string that is mixed with some numbers. We will use **expr substr** commands to extract only the **Hostinger** text value.

```
#!/bin/bash
expr substr "458449Hostinger4132" 7 9
```

13. Find and Replace String

Another useful bash script for strings is **find and replace**. Create a file named **findreplace.sh**:

```
nano findreplace.sh
```

Then paste in the following bash script:

```
#!/bin/bash
first="I drive a BMW and Volvo"
second="Audi"
echo "${first/BMW/"$second"}"
```

```
root@myvps:~# bash findreplace.sh
I drive a Audi and Volvo
root@myvps:~#
```

14. Concatenate Strings

Concatenation is the term used for appending one string to the end of another string. Start by creating **concatenation.sh** file.

```
nano concatenation.sh
```

The most simple example would be the following:

```
#!/bin/bash
firststring="The secret is..."
secondstring="Bash"
thirdstring="$firststring$secondstring"
echo "$thirdstring"
```

The above script will connect the values of **firststring** and **secondstring** variables creating a whole new **thirdstring**.

A more advanced example would look like this:

```
#!/bin/bash
firststring="The secret is..."
firststring+="Bash"
echo "$firststring"
```

The script uses the += operator to join the strings. With this method, you can concatenate strings with only one variable.

16. Generate Factorial of Number

The factorial of a number is the result of all positive descending integers. For example, the factorial of 5 would be 120:

```
5! = 5*4*3*2*1 = 120
```

Factorial scrips are very useful for users learning about recursion. Start by creating a **.sh** file executable:

```
factorial.sh
```

The following script will ask the user to enter a number they want to get the factorial of and use a **for loop** to calculate it.


```
#!/bin/bash
echo Enter the number you want to get factorial for
read mynumber
factorial=1
for ((i=1;i<=mynumber;i++))
do
factorial=$((factorial*i))
done
echo $factorial
```

```
root@myvps:~# bash factorial.sh
Enter number you want to get factorial for
5
120
root@myvps:~#
```

17. Create Directories

It is effortless to create directories in bash unless you need to create a lot of directories quickly. In the following example, we will use the bash script to create a set of directories with the same subdirectories in each.

First, create a file named **directories.sh**:

```
nano directories.sh
```

Then paste in the following code:

```
#!/bin/bash
mkdir -p {Math,English,Geography,Arts}/{notes,examresults,portfolio}
```

The script creates 4 main directories: **Math**, **English**, **Geography**, and **Arts**. The **Notes**, **examresults**, and **portfolio** subdirectories are also created inside each.

If you were to replace the / symbol in the middle with _, the script would look like this:

```
#!/bin/bash
mkdir -p {Math,English,Geography,Arts}_{notes,examresults,portfolio}
```

Here's the output for it displaying a merge of the two directories:

```
root@myvps:~# bash directories.sh
root@myvps:~# ls
Arts_examresults Arts_notes Arts_portfolio
root@myvps:~#
```

18. Read Files

In order to read a file in bash, you will need to create a sample file first. Do so with the following command:

```
nano mysamplefile.txt
```

Fill it with some sample data:

```
Out of all scripting languages, bash is the most popular one. It allows programmers to  
run scripts effortlessly in a variety of Linux distros.
```

Then create the actual script file:

```
nano readfiles.sh
```

Fill it with the following lines:

```
#!/bin/bash  
myvalue=`cat mysamplefile.txt`  
echo "$myvalue"
```

Running the script results in this output:

```
root@myvps:~# bash readfiles.sh  
Out of all scripting languages, bash is the most popular one.  
It allows programmers to run scripts effortlessly in a variety of linux distros.  
root@myvps:~#
```

19. Print Files With Line Count

We'll print a file with its line count. Let's create it first:

```
nano cars.txt
```

In our example, we will fill it with our favorite car brands:

```
Audi  
BMW  
Bentley  
Maserati  
Seat  
Volvo
```

Save the file and create a new bash script:

```
nano printlines.sh
```

Then paste in the following code:

```
#!/bin/bash  
myfile='cars.txt'  
i=1  
while read lines; do  
echo "$i : $lines"  
i=$((i+1))  
done < $myfile
```

```
root@myvps:~# bash printlines.sh  
1 : Audi  
2 : BMW  
3 : Bentley  
4 : Maserati  
5 : Seat  
6 : Volvo  
root@myvps:~# cat cars.txt  
Audi  
BMW  
Bentley  
Maserati  
Seat  
Volvo  
root@myvps:~#
```

The file contents of **cars.txt** match the printout of the while loop script.

20. Delete Files

To delete an existing file, you can use an **if** statement to check if the file exists and instruct the bash script to remove it. Start by creating the bash script file:

```
nano deletefiles.sh
```

The following script will create a new file named **cars.txt**, and then – with the help of the if statement – check if it exists and delete it.

```
#!/bin/bash
myfile='cars.txt'
touch $myfile
if [ -f $myfile ]; then
    rm cars.txt
    echo "myfile deleted"
fi
```

21. Test if File Exists

In order to check if a given file exists, users can perform conditional tests. In this case, we'll use an **if** statement with a **-f** flag. The flag checks if a given file exists and is a regular file. Start by creating the script file:

```
nano exists.sh
```

Copy and paste the following script:

```
#!/bin/bash
MyFile=cars.txt
if [ -f "$MyFile" ]; then
    echo "$MyFile exists."
else
    echo "$MyFile does not exist."
fi
```

Running the script results in the following output:

```
root@myvps:~# bash exists.sh
cars.txt exists.
root@myvps:~#
```