5 example bash scripts to help you learn Linux programming

Writing bash scripts is one of the most popular and accessible ways of programming Linux computers. These simple script examples will help you understand the process and introduce you to the basics of Bash programming.

Writing bash scripts is one of the most popular and accessible ways of programming Linux computers. These simple script examples will help you understand the process and introduce you to the basics of Bash programming.

1. How to print Hello World in Bash

For example, Hello World is a great way to learn about any programming language, and Bash is no exception.

Here's how to print "Hello World" using Bash:

1. Open the editor and start editing the new file containing the following lines of code.

2. The first line of a Bash script should always look like this:

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

Note: The shebang command (#!/bin/bash) is essential because the shell uses it to decide how to run the script. In this case it uses the Bash interpreter.

3. Any line starting with the # symbol is a comment. The shebang line is a special case, but you can use separate comments to explain your code. Add comment on line 2, for example:

```
# Print some text from a bash script
```

4. You can print to standard output using the echo command, followed by the value you want to print. Add the following line to line 3:

```
echo "Hello World"
```

5. Save the script, preferably with the .sh extension, for example, hello_world.sh. The extension is not a requirement but it is a useful convention to follow.

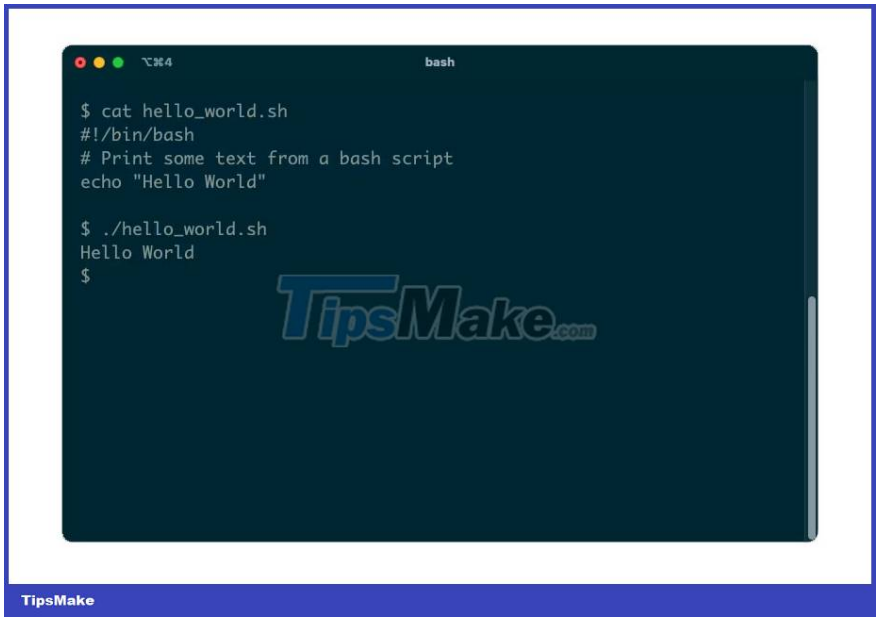
6. To run your script, make the file executable. Use the chmod ("change mode") command along with the +x ("executable") argument and the name of your shell script:

```
chmod +x hello_world.sh
```

7. Use this command to run the script from within its directory:

```
./hello_world.sh
```

8. When the script runs, it will print the text "Hello World" on your terminal:



2. Create directory by reading input

From your script, you can run any program that you would normally run on the command line. For example, you can create a new directory from your script using the mkdir command.

1. Start with the same shebang line as before:

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

2. Prompt the user to enter a folder name using the echo command as before:

```
echo "Enter new directory name:"
```

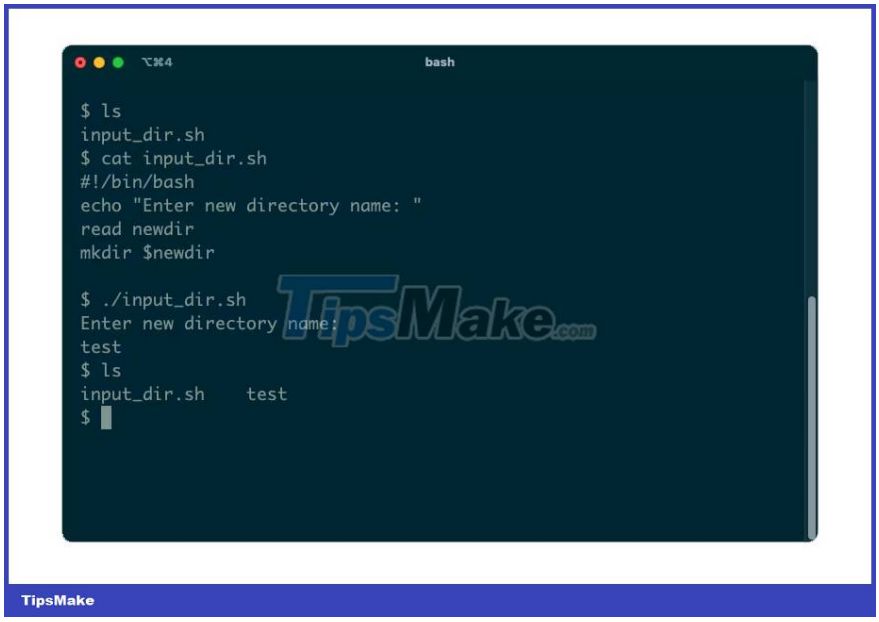
3. Use the built-in read command to fetch user input. The single argument names a variable in which the shell will store input:

```
read newdir
```

4. When you need to use the value stored in a variable, add a dollar sign ($) in front of the variable's name. You can pass the contents of an input variable as an argument to the mkdir command to create a new directory:

```
mkdir $newdir
```

5. When you run this script, it will prompt you for input. Enter a valid folder name and you will see the script create it in your current folder:



3. Create a directory using command line arguments

As an alternative to reading input data interactively, most Linux commands support arguments. You can provide an argument when running a program to control its behavior.

In your script, you can use $1 to refer to a special variable that holds the value of the first argument. $2 will refer to the second argument, etc.

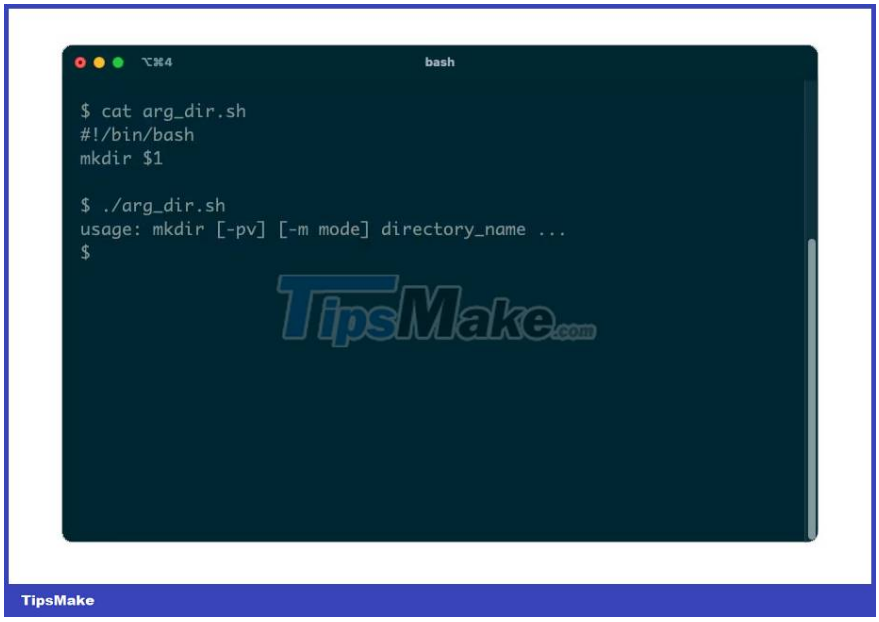
1. Create a directory using the mkdir command from the previous example. This time, however, let's use the built-in variable $1:

```
#!/bin/bash mkdir $1
```

2. Run the script, this time passing the new folder name you chose as an argument:

```
./arg_dir.sh Test
```

You may wonder what happens if you run the script without providing any arguments. Try it and see; you will get an error message starting "usage:mkdir":



Without any command line arguments, the value $1 will be empty. When your script calls mkdir, the script will not pass arguments to it and the mkdir command will return that error. To avoid this, you can check the status yourself and give a friendlier error:

1. As always, let's start with the shebang:

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

2. Before you call mkdir, check for the first argument to be empty (i.e. no arguments). You can do this with a Bash if statement that runs code based on a condition:

```
if ["$1" = ""]; then
```

3. If the first argument is empty, print an error and exit your script:

```
echo "Please provide a new directory name as the first argument" exit
```

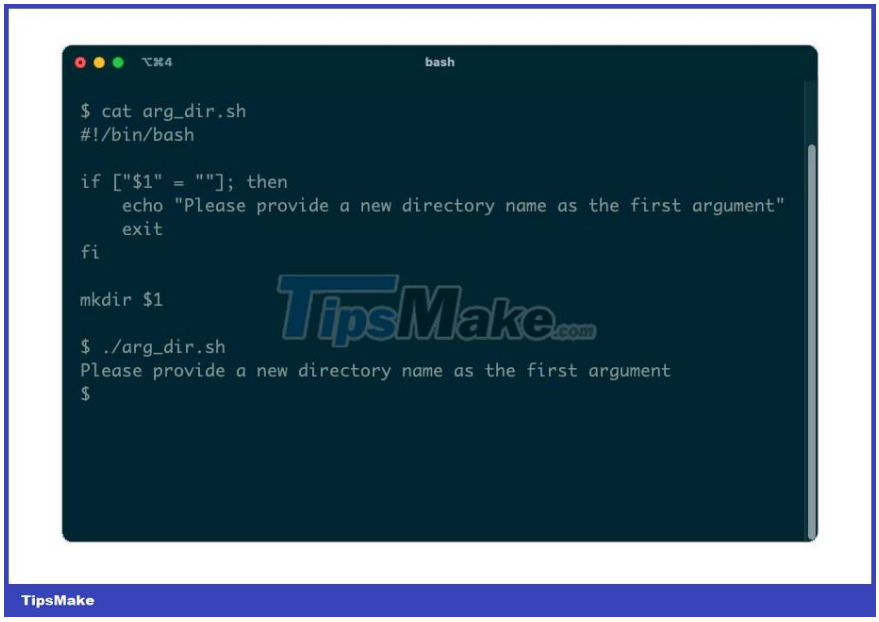
4. The somewhat strange "fi" keyword (the "reverse if") signals the end of an if statement in Bash:

```
fi
```

5. Your script can now continue as before, to handle the case when arguments are present:

```
mkdir $1
```

When you run this new version of the script, you will receive a notification if you forget to include the argument:



4. Delete files using Bash function

If you find yourself repeating the same code, consider wrapping it in a function. You can then call that function whenever you need it.

This is an example of the function to delete a certain file.

1. Start with the shebang line:

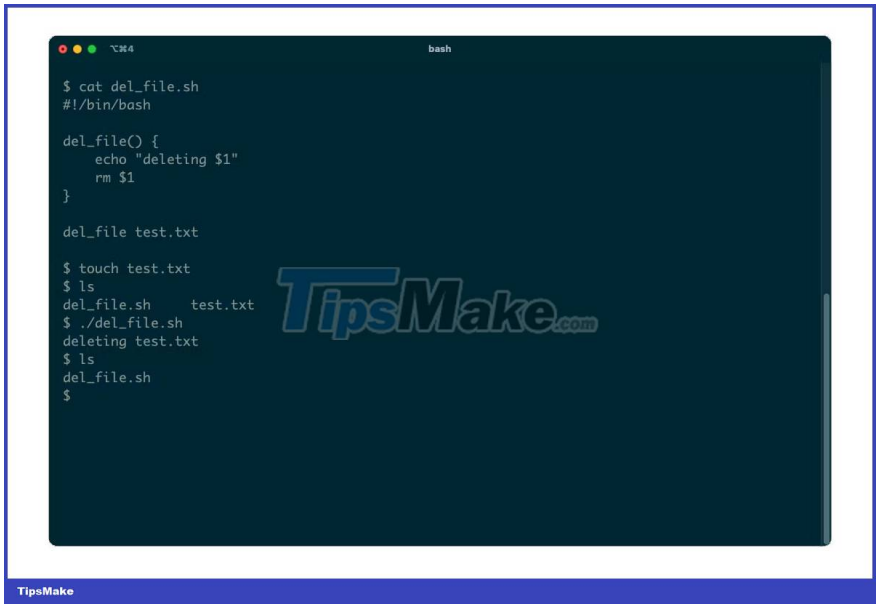
```
#!/bin/bash
```

2. Identify the function by writing its name, followed by empty parentheses and the commands inside the curly brackets:

```
del_file() { echo "deleting $1" rm $1 }
```

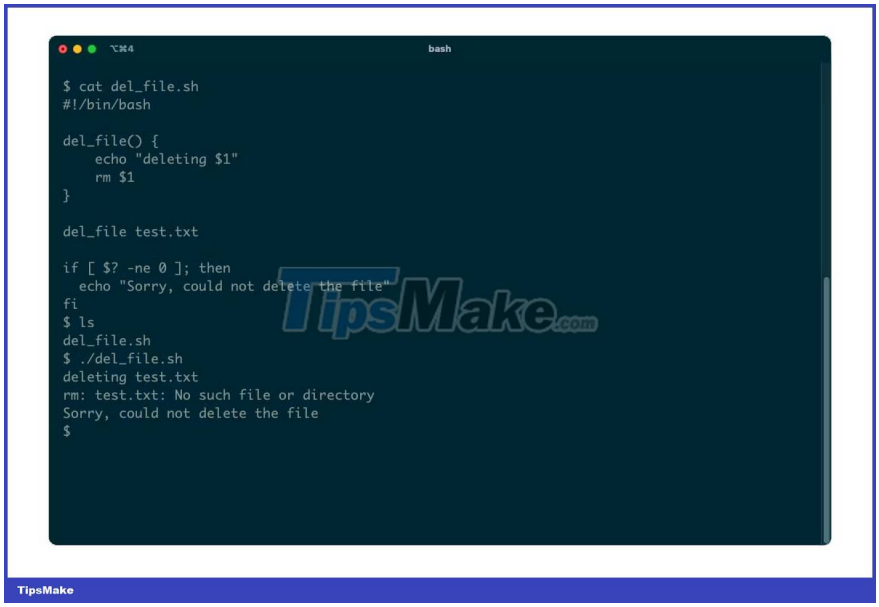
You can then call the function and pass it the name of the file to delete:

```
del_file test.txt
```



When you call a function, it sets the value $? especially with the exit status of the last command it ran. Exit status is useful for error checking; In this example you can check if the rm command was successful or not:

```
if [ $? -ne 0 ]; then echo "Sorry, could not delete the file" fi
```



5. Create a basic calculator for arithmetic calculations

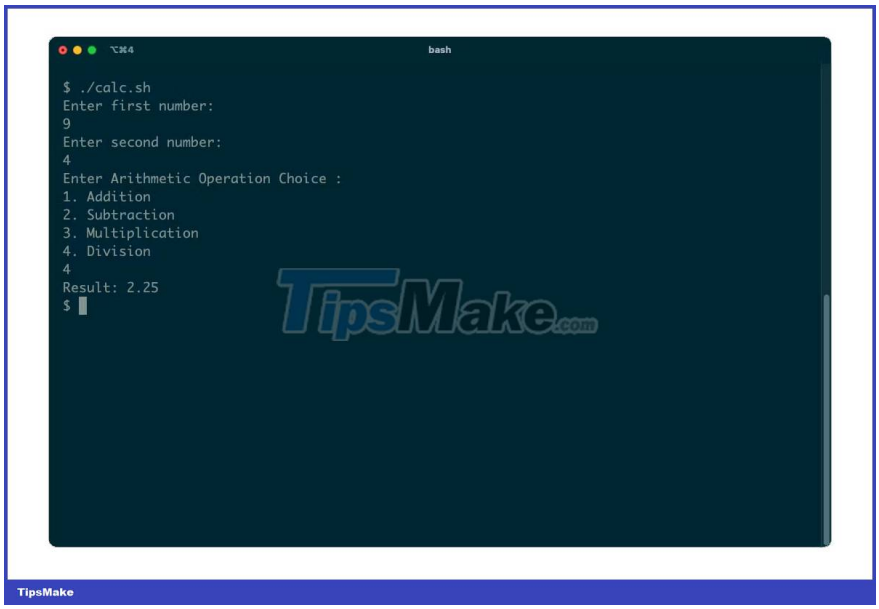
This last example illustrates a very basic calculator. When you run it, you'll enter two values, then choose an arithmetic operation to perform them.

Here is the code for calc.sh:

```
#!/bin/bash # Take operands as input echo "Enter first number: " read a echo "En
```

Note the use of case . esac which is equivalent to Bash's switch statement from other languages. It allows you to test a value - in this case the variable choice - against some fixed value and run the associated code.

This script uses the bc command to perform each calculation.



You finished reading the article "**5 example bash scripts to help you learn Linux programming**" edited by the **TipsMake** team. We hope this article has provided you with many useful tech tips and tricks. You can search for similar articles on tips and guides. Thank you for reading and for following us regularly.