After you've successfully installed Linux, you can start exploring the command line. Working in a terminal at first may seem confusing because you need to deal with operating systems using plain-text commands. However, if you master it, you will be able to automate every single bit of the whole system. Also, good knowledge of the Linux shell will help you write scripts to automate frequent actions. It is easier than using the same commands every time you need them. So, let's start by writing your first bash program!

**Echo utility**

Bash is an enhanced version of the Bourne shell. Its name stands for "Bourne Again Shell" and also can be understood as a "Born again" shell. Bash is needed to receive user commands and send them to the operating system for further processing. In addition to Bash, there are other versions of command shells. However, we will use Bash in our topics, as it is one of the most popular modern shells in the Linux environment.

Usually, the very first task is to output a text line. In Bash, there is a special utility for this. The echo command is simple but useful, though it can only output text lines to the terminal. The syntax is also easy: echo <options> <text\_line>. Let's try it! Initially, we can skip the options and just output a line in the terminal.

$ echo Hello, Linux!

Hello, Linux!

If you want to output a text with several lines, use the quotation marks:

$ echo 'Hello, Linux!' 'Here I am!'

Okay, but what if you need to output each phrase on a separate line? Then you should add a -e flag as an option into the echo command. This flag enables support for outputting escape sequences such as linefeed \n:

$ echo -e 'Hello, Linux! \nHere I am!'

Hello, Linux!

Here I am!

In case you don't want to output the linefeed after the message, add -n flag to the echo command.

Great, the first step is complete, and now you know how to output lines of text! What's next?

**Hello World!**

The next step is to write the first 'Hello, world!' script in Bash. Create the file named hello\_world.sh (you can use a text editor).

#!/usr/bin/env bash

echo 'Hello, world!'

Our script consists of two lines. The first one contains the path to the shell interpreter with which you need to execute the command. In this example, it's /usr/bin/env bash that searches the Bash executable in your environment. At the beginning of this line, there is a sequence of two symbols #!, that's called a**shebang.** It is a directive for the Linux program loader that specifies a program that will run the script. The second line is already familiar to you, it's the echo command.

Shebang defines an interpreter for the desired programming language. For example, to run a Python script add a line #!/usr/bin/env python3. You can run a script from the command line like this: ./file.py.

OK, you've got a script. How to run it?

**Run a shell script**

There are three ways to run a shell script:

* You can open the corresponding directory with your file and type bash hello\_world.sh. This way requires only one command, but do not forget to specify that you want to launch the script with Bash, because running a script this way we ignore the shebang directive.
* You can make the file executable by typing chmod +x hello\_world.sh. Then you can run it using ./hello\_world.sh also from its directory. Here, you can skip specifying Bash, but remember to make the file executable.
* You can place the file in bin by sudo cp hello\_world.sh /usr/local/bin and then use only the filename hello\_world.sh to run the script. As you can see, you may also need sudo (superuser access) to put your file in the right place. This way implies that you know how to change a directory in Bash and what sudo mode is. And the advantage of this is that the final command consists of a filename only.

Another approach is to add the directory to your PATH variable, but we've learned a lot today, so we'll tell you how to work with Linux system variables in the next topics.

If you are not familiar with chmod, cp, and sudo, don't worry – we'll explain them later in detail. For now, you just need to understand how we use them.

**Conclusion**

Congratulations! Now you know how to create shell scripts and run them in Bash! We hope you found this information useful and now you're one more step closer to understanding how Linux works.

## Sleeping program

Command sleep stops execution of program for a specified time.

Rearrange the lines to create a Bash program, which stops process execution for 7 seconds, while displaying comments about its actions.

Here is the correct order for the Bash script:

1. #!/bin/bash
2. echo “Now will be pause for 7 seconds”
3. sleep 7
4. echo “Pause is completed”

✅ This script starts by declaring the shell, then informs the user, pauses for 7 seconds, and finally prints that the pause is over.

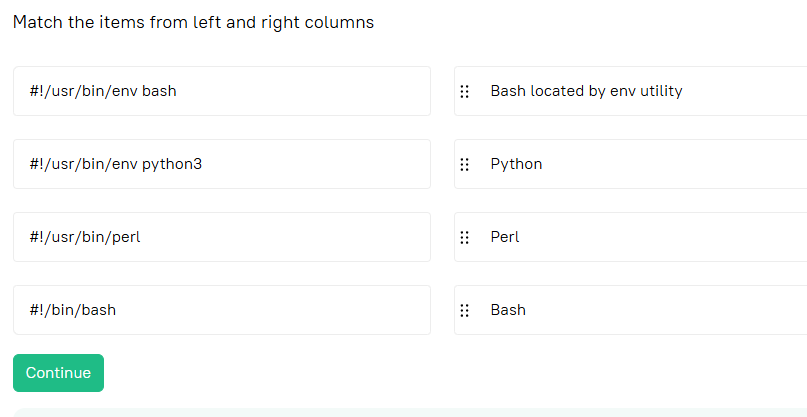
# First program in bash

## Output lines

Which command can output text lines in Bash?

Echo

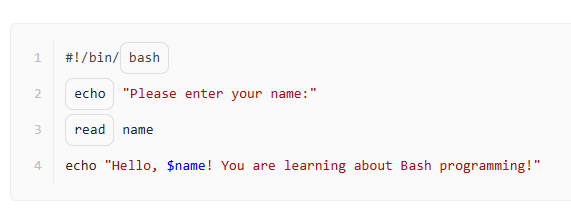
Look at the first lines of the different scripts and find for them right interpreters.



**Crafting a personalized greeting program**

You are assigned to write a simple greeting program in bash. The program should prompt the user for their name and then print out a customized message saying 'Hello' and the user's name. For practicing bash scripting, fill the blanks in the code to make this program work correctly.

Fill in the gaps with the relevant elements

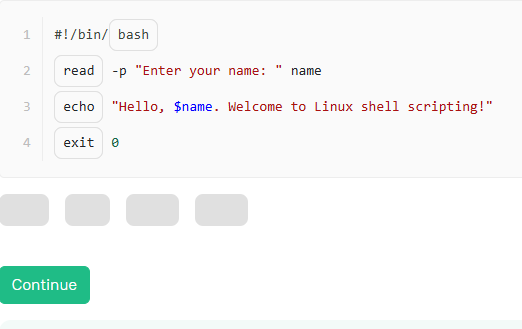


**Creating an interactive greeting script in Linux**

 Report a typo

As an upcoming system administrator, you are learning Linux and working with bash scripting. Your task is to create a simple interactive bash script that greets the user by their name. Review the following script and fill the blanks in code to make it work. The script should first prompt the user to enter their name, then output 'Hello, \. Welcome to Linux shell scripting!', replacing \ with the name entered by the user. In the end, it should terminate successfully. Remember to follow the best practices and syntax that you have learned so far.

Fill in the gaps with the relevant elements

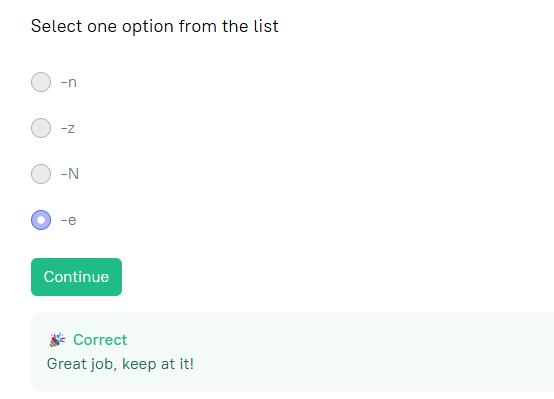


**Shebang in use**

What is shebang used for? to specify the program to run the script

**Escape sequences option**

Which echo option allows using escape sequences?



**Missing symbols**

Lina wrote the following script:

echo 'Hello, bash!'

What line should she add at the beginning of the script to tell the program loader where to look for an interpreter?

#!/usr/bin/env bash

**Your first Bash script**

 Report a typo

Write a script with \n that will display phrases in different lines. For example, for 'Hi!' and 'Hey!' the output should be the following:

Hi!

Hey!

Sample Input 1:

Sample Output 1:

Hi!

Hey!

#!/usr/bin/env bash

echo -e "Hi!\nHey!"

**Run Bash script**

Your colleague created a script file and asked you to run it. The path to the file is */tmp/run\_me.sh*, but your colleague forgot to make the file executable, so choose the right way to run the script.

Do not modify the content of the */tmp/run\_me.sh*file.

#!/usr/bin/env bash

solve() {

# add your solution here

bash /tmp/run\_me.sh

}