



# Bash

Bash for anyone

Author: Vadym Savenko @2024



## Content:

1. About Bash (p. 3)
2. Commands (p. 3 — 5)
3. Variables (p. 5 — 6)
4. Math (p. 6 — 7)
5. Conventation (p. 7)
6. Conditions (p. 8 — 9)
7. Circuit (p. 9 — 10)
8. Functions (p. 10 — 11)
9. Opening files (p. 12)
10. Dictionary of base variables Linux (p. 13)
11. Dictionary of base commands Linux (p. 14 — 15)

# 1. About Bash

Bash was developed as an improved version of the original Bourne Shell (sh), including new features and enhancements. It is compatible with most commands and scripts written for sh, but also provides many additional features such as improved string handling, array operations, and advanced input/output stream management. It is primarily used as a command interpreter on Linux distributions. Typically, a file for Bash projects has a .sh format, but files may not necessarily have this extension. It is sufficient to enter the command in the terminal: "sh /path/to/file". However, if you do not want to type such a command every time, you can use the command: "chmod +x /path/to/file". Now, by entering "./the\_file" or "/path/to/file" in the console, the system will understand that this is a Bash project and needs to be executed with the command: "sh /path/to/file".

## 2. Commands

To write Bash code, you can use any code editor. The code that displays text in the console:

```
echo "Hello World"
```

As a result, the console will display:

```
Hello World
```

Where "Hello World" was written, you can specify data of different values. There is also a command that takes data from the user. It looks like this:

```
read var
```

Where "var" was written, it indicated that a variable "var" is being created. The discussion about variables will come later.

Let's create a code that will display our entered first and last name using the commands we learned:

```
echo "Your first name:"  
read firstName  
echo "Your last name:"  
read lastName  
echo "Your are: $firstName $lastName"
```

If variables were written with a dollar sign (\$), it must be present if we're using it (if we're creating variables, then the dollar sign should not be included).

Upon running this code, you will be prompted to enter your first and last name, and after entering your data, you will get the result in the terminal indicating who you are.

You can also create comments in the code. This is meant to skip the execution of a command. Comments are created using a hash symbol (#) at the beginning of the command line:

```
# echo Comment
```

There are other commands as well, but this book will focus on the fundamental codes for Linux. The dictionary with basic commands is on p. 15.

## 3. Variables

At the p. 4 was mention about variables. Sometimes there are situations when variables should have their own data, not the ones entered by the user. Here's an example of creating variables:

```
var1="Hi" # Data type: text
var2=50 # Data type: number
var3=(50 "Hi" 30) # Data type: array
echo "Vars: 1st=$var1; 2nd=$var2; 3rd=${var3[2]}"
```

As noted, all variables have a value that has its own data type. In Bash, there are 3 data types:

- **Texts:** where any set of characters is stored, they are stored within double quotes (" ").
- **Numbers:** where only numbers are stored.
- **Arrays:** where a collection of different values with different data types is stored.

If we know how to represent variables of numeric and text data types, arrays also have an index, which is an integer stored in square brackets ([]). The index is the sequence number of the element, where the first element starts not from 1 (since it's already the second element), but from 0.

If the index is less than 0 (for example: -1, -2, -3, and so on), it means that the elements are taken in reverse order. For example: [-1] - the last element; [-2] - the second-to-last element; and so on.

You can also add numeric and text variables to an array. Here's an example:

```
var=50  
array=($var "Hi" 30)
```

In Python, after each element (except the last one), you need to put a comma and space, confirming the creation of an element in the list. For Bash, as an array, it's sufficient to just put a space.

## 4. Math

To perform mathematical operations in Bash, commands like `let`, arithmetic expansion `$(( ))`, or the `expr` command are used. The data type of variables should be numeric. Here's the code for working with mathematical operations:

```
let "a = 5 + 3"  
b=$((5 * 3))  
c=$(expr 10 - 3)
```

If we multiply text data by a number, it will result in the repetition of the text data. If we add text data to other text data, it will concatenate the text data. In addition to multiplication, addition, and subtraction, there are other arithmetic operations. Here's an example code with other arithmetic actions is located on page 7.

Example of code:

```
quotient=$((first_number / second_number)) # Division
power=$(echo "$first_number^$second_number" | bc) # Exponentiation
sqrt_first=$(echo "scale=2; sqrt($first_number)" | bc) # Square
```

By the way!!! It was forgotten to mention an important thing. Variable names should be:

- 1) Written in the English alphabet;
- 2) Does not match the keywords in Bash (Such as: *as, from, with, if* и другие. We'll be learning all the keywords in Bash);
- 3) Must to start from a letter;
- 4) Variable names cannot start with digits ;
- 5) Must not contain any characters other than: letters of the English alphabet, digits, hyphens, or the underscore ( \_ ) symbol ;

## 5. Convertation

When we want to have the same data of a variable but in a different format, we need to perform conversion. Example of code with conversion:

```
# Numbers to strings
number=789
text="$number" # It save number in number type

# Strings to numbers
text="1011"
number=$((text)) # It save text in text type
```

## 6. Conditions

If today is Monday, Tuesday, Wednesday, Thursday, or Friday, we work. If it is Saturday, we work less. Otherwise, we rest. This paragraph is an example of conditions.

In Bash, you can set conditions. Here is an example code with conditions :

```
if [ $VAR -gt 10 ]; then # If is True
    echo "VAR more 10"
elif [ $VAR -eq 10 ]; then # Else, if this is True
    echo "VAR equal 10"
else # Else (If all condition is - False)
    echo "VAR less 10"
fi
```

Here, *-gt* means "greater than", *-eq* means "equal to", and *-lt* means "less than". In short, these are all logical operators. In Bash, there are more logical operators. They are mainly divided into numerical and string operators.

Here are all the logical operators:

For numbers dates:

- *-gt*: greater than
- *-lt*: less than
- *-ge*: greater than or equal to
- *-le*: less than or equal to
- *-eq*: equal to
- *-ne*: not equal to



For strings dates:

- >: greater than
- <: less than
- =: equal to
- !=: not equal to

## 7. Circuit

We gave an example on page 8, and now let's imagine it as a loop that repeats continuously until we retire .

In Bash, there are loops that repeat actions until their condition becomes false, and loops that execute a set of commands multiple times. Here is an example of a loop that executes a set of commands multiple times:

```
for i in {1..5}; do
    echo "Number: $i"
done
# Displays the result from 1 to 5.
# It means performing the command 5 times.
# {1..5} - there, we set the execution to occur a specified number of times.
```

Here is an example of an infinite loop:

Where there are square brackets ( [ ] ), we specify conditions to determine truth and falsehood.

```
COUNTER=0

while [ $COUNTER -lt 5 ]; do
    echo "COUNTER: $COUNTER"
    COUNTER=$((COUNTER + 1))
done
```

# 8. Function

Functions in programming, including Bash scripts, are used for several important reasons. Let's consider an example of creating code with a function:

```
my_function() {  
    local a=$1  
    local b=$2  
    echo "This is a function"  
    echo "Argument a: $a"  
    echo "Argument b: $b"  
}  
  
# Calling a function with arguments  
my_function 3 5
```

Now let's analyze the code :

## 1. Shebang:

This is the first line of the script, called the "shebang". It tells the system that this script should be executed using the Bash interpreter.

## 2. Defining a function:

Here, a function named *my\_function* is defined. In Bash, functions are declared with the function name, followed by a pair of parentheses and curly braces {} to define the body of the function.

Внутри функции:

- *local a=\$1*: A local variable named *a* is created and assigned the value of the first argument of the function (*\$1*).

- *local b=\$2*: A local variable named *a* is created and assigned the value of the second argument of the function (*\$2*).
- *echo "This is a function"*: Выводится строка "Это функция".
- *echo "Argument a: \$a"*: The value of the variable *a* is printed.
- *echo "Argument b: \$b"*: The value of the variable *b* is printed .

### 3. Calling a function with arguments:

Here, the function *my\_function* is called with two arguments: 3 and 5. These arguments are passed to the function, where they become accessible as *\$1* and *\$2*.

#### Detailed job description:

1. **When is the function *my\_function* called with arguments 3 and 5 :**
    - 3 becomes the first argument and is assigned to the variable *a* inside the function .
    - 5 becomes the second argument and is assigned to the variable *b* inside the function.
- **Inside the function:**
    - *local a=\$1*: Variable *a* is assigned a value 3.
    - *local b=\$2*: Variable *b* is assigned a value 5.
    - *echo "This is a function"*: String is displayed "This is a function".
    - *echo "Argument a: \$3"*: String is displayed "Argument a: 3".
    - *echo "Argument b: \$5"*: String is displayed "Argument b: 5".

#### The complete execution process :

1. The script starts with the line *#!/bin/bash*, which instructs the system to use the Bash interpreter.
2. The function *my\_function* is defined.
3. The function *my\_function* is called with arguments 3 and 5.
4. Inside the function *my\_function*:
  - Arguments 3 and 5 are assigned to local variables *a* and *b*.
  - The function prints strings with the text and the values of the arguments.

## 9. Opening files

It's known that you can store data using variables, but you can also store, modify, append, and read data from files. Here's an example of Bash code for reading, appending, and modifying files:

```
# Example of reading from a file
while IFS= read -r line; do
    echo "$line" # $line - the variable storing data from the file
done < "/path/to/file"

# Example of appending data to a file
echo "New string" >> "nameFile"

# Example of writing to a file (overwrite)
echo "The overwritten line" > "nameFile"
```

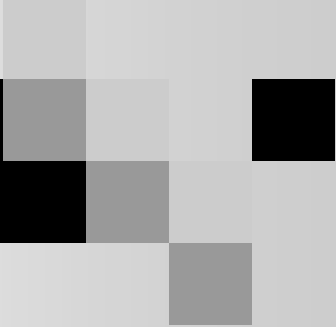
# 10. Dictionary of base variables Linux

Variable purpose	Variable name	Variable description
Path to executable files	\$PATH	List of directories where Bash looks for executable files.
User's home directory	\$HOME	Path to the current user's home directory.
Current user name	\$USER	Current username.
Default shell command	\$SHELL	Path to the default shell executable.
Temporary files directory	\$TMPDIR (или \$TMP)	Path to the directory used for temporary files.
Operating system version	\$OSTYPE	Operating system type.
Standard text editor	\$EDITOR	Name of the standard text editor.
User's personal identifier	\$UID	Numerical identifier of the current user.
User's group personal identifier	\$GID	Numerical identifier of the current user's group.
Standard number of lines per screen	\$LINES	Number of lines in the terminal.
Standard number of columns per screen	\$COLUMNS	Number of columns in the terminal.

# 11. Dictionary of base commands Linux

Command	Purpose
<code>sudo su</code>	Switch to superuser mode
<code>echo text</code>	Show <i>text</i> on terminal
<code>nano path/to/file</code> Пакет: <i>nano</i>	Go to text editor and to edit: <i>path/to/file</i> .
<code>ip addr</code> Пакет: <i>ip</i>	To know IP-address of the host (PC)
<code>cd path/to/directory</code>	Go to: <i>path/to/directory</i>
<code>ls</code>	Посмотреть, что находится в директории
<code>wget https://site.to.file</code> Пакет: <i>wget</i>	Downloading a file from internet
<code>rm path/to/file</code>	Remove a file
<code>rm -r path/to/directory</code>	Remove a directory
<code>cp path/to/file path/for/paste</code>	Coping a file
<code>cp -r path/to/directory path/to/paste</code>	Remove a directory
<code>mkdir nameFolder</code>	Creating folder “ <i>nameFolder</i> ”
<code>kill process</code>	Stop the process
<code>neofetch</code>	Information about system

Command	Purpose
<code>chmod +x path/to/file</code>	Indicates to the system that this is not a text file but rather code to execute Bash commands
<code>git clone https://path.to.repository</code> Пакет: <i>git</i>	Downloading all files from repositories (For example: from GitHub)
<code>sudo ssh -X userOfThisPC@IPofPC</code> Пакет: <i>openssh</i>	Remote access connection and operation via console
<code>sudo ssh -X userOfThisPC@IPofPC programm</code> Пакет: <i>openssh</i>	Remote access connection and operation via a "program" (this can be graphical or console-based)
<code>mv file1 file2 ... /path/to/directory</code>	Moving a file
<code>mv -r /path/directory /path/to/second/directory</code>	Moving a folder
<code>scp /path/to/locale/file username@remote_host:/path/to/remote/host</code> Пакет: <i>openssh</i>	Coping a file to remote PC
<code>gzip file.txt</code> Пакет: <i>gzip</i>	Archive compression <i>file.txt.tar.gz</i>
<code>gzip -l archive</code> Пакет: <i>gzip</i>	Information about archive
<code>bzip2 -d archive</code> Пакет: <i>bzip2</i>	Archive extraction



Bash for anyone

Author: Vadym Savenko @2024