You've already learned about what bash scripts are, how to write them, and how to use the echo command. Today we'll talk about passing command line parameters to process them later in scripts. Also, to automate repetitive scenarios, we'll learn how to write functions. Let's start with the parameters.

**Arguments**

Using arguments, you can pass information for processing to a script. When you run a script, the bash shell assigns them to special variables called positional parameters. The null parameter $0 is always the name of the script, and then follows user parameters passed to a script. There can be as many parameters as you want.

For example, there is a simple script called personal\_data.sh. It receives information about the name and age of a person and then outputs it to the console. It looks like this:

#!/usr/bin/env bash

echo "You provided $# facts about yourself!"

echo "Your name is $1"

echo "Your age is $2"

In our example, we are passing two parameters, the name and the age. Therefore, they are assigned respectively to $1 and $2 and will be sequentially read from the console. These positional parameters may also be denoted as ${<parameter>}, for instance, if they contain more than one symbol, ex., ${123}.

The $# variable denotes the total number of the parameters (in our case there are 2 parameters). We also output this number to the console.

Here's what executing the script will look like:

$ bash personal\_data.sh Amy 26

You provided 2 facts about yourself!

Your name is Amy

Your age is 26

In some cases, you need to group all the parameters passed to the script. To do this, you can use the $\*and $@ variables. They both contain all command line parameters, making it possible to access what is passed to the script without using positional parameters. The $\* variable contains all the parameters entered on the command line as a single "word". That is, everything you pass will be collapsed into a single variable. In the $@ variable, the parameters are split into separate "words". There will be a separate position for each parameter.

Great! Now you know how to pass parameters to the script. But what if you need to reuse some part of a script? Constantly typing it by hand is boring and time-consuming, and copying, as you know, is evil. So, in this case, we will need functions.

**Functions**

Functions allow you to write a block of code once and, when you need it again, simply refer to it in the script. Let's take a look at the simplest function as an example. A function can be set in two ways: either by function function\_name {} or function\_name() {}. Okay, let's improve on the previous code:

#!/usr/bin/env bash

personal\_data() {

echo "You provided $# facts about yourself!"

echo "Your name is $1"

echo "Your age is $2"

}

Now the user experience will be a separate function called personal\_data(). We can call this function by adding below in the same file:

personal\_data "Amy" 26

Then we open bash and call the function. Note, that now we do not need to type parameters in the console, just the filename with the function and given parameters.

$ bash personal\_data.sh

You provided 2 facts about yourself!

Your name is Amy

Your age is 26

Cool, the function works!

**Comments**

In order not to forget what the script is doing, one may add comments using # , like in the line # processing the user's data. The comment briefly describes what's going on. Also, if there is no need for some code lines right now, they can be commented out as well, for example: # echo "You provided $# facts about yourself!". This time the script will not output the number of the given arguments:

#!/usr/bin/env bash

# processing the user's data

personal\_data() {

# echo "You provided $# facts about yourself!"

echo "Your name is $1"

echo "Your age is $2"

}

These comments can help you and other developers understand scripts and remember what they are for.

**Conclusion**

So, we've figured out how to make a simple function and how to call it in the console. We've also figured out how to pass parameters to the script and how they can be processed. Now, we hope, it will be even easier and faster for you to work in bash!

Write a function second\_parameter that prints the second parameter.

The string should look like this: The second parameter is <here is the second parameter>

**Sample Input 1:**

2 5 10

**Sample Output 1:**

The second parameter is 5

#!/usr/bin/env bash

second\_parameter() {

echo "The second parameter is $2"

}

**Number of positional parameters**

How many command line parameters can be assigned to positional parameters?

as many as you want

Are there any mistakes in the syntax of this function?

function (

echo "Hello there"

)

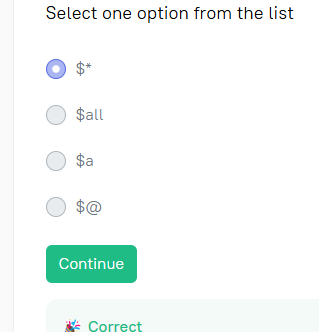
Select one or more options from the list

Wrong type of brackets

There is no function\_name()

**All arguments**

Which variable can be used to access all the arguments from the console as one word?



**Output the total number**

 Report a typo

Write a function total\_number that prints the total number of given parameters. The output line should look like this: The total number of parameters is <there is the total number>

Sample Input 1:

2 5 10 100 55

Sample Output 1:

The total number of parameters is 5

#!/usr/bin/env bash

total\_number() {

echo "The total number of parameters is $#"

}

**What is the output?**

What does $0 stand for?

name of the script

**Output of number and parameters**

Write a partial\_output() function that will output the number of parameters, and then will display the first and third parameters of the bunch.

Sample Input 1:

Mary Jane Ostin

Sample Output 1:

There are 3 parameters

The first one is Mary

The third one is Ostin

partial\_output() {

echo "There are $# parameters"

echo "The first one is $1"

echo "The third one is $3"

}

**Comments**

How can you add comments to your scripts?

Put # before the line of code.

Write a function hello() that takes a username as a parameter and returns the string Hello, user <username>!

**Sample Input 1:**

Mary

**Sample Output 1:**

Hello, user Mary!

#!/usr/bin/env bash

hello() {

echo "Hello, user $1!"

}