10 year anniversary

LOG IN

Main menu

[Articles](#) [Resources](#) [Downloads](#) [About](#)

[Open Organization](#)

How to write functions in Bash

Reduce redundancy and maintenance in your code by writing functions.

10 Jun 2020 | Seth Kenlon (Red Hat) (/users/seth) | 27

When you're programming, you're literally defining a procedure, or a *routine*, you want the computer to perform. A simple analogy compares computer programming to baking bread: you list ingredients once to set up the work environment, then you list the steps you must take to end up with a loaf of bread. In both programming and baking, some steps must be repeated at different intervals. In baking bread, for instance, this could be the process of feeding a sourdough culture:

```
STIR=100
SNOOZE=86400

function feed_culture {
  remove_from(pantry)
  add(flour, water)
  stir($STIR)
  sleep($SNOOZE)
}
```

And later, kneading and proofing the dough:

```
KNEAD=600
SNOOZE=7200

function process_dough {
  remove_from(proofing_drawer)
  knead($KNEAD)
  return_to_drawer($SNOOZE)
}
```

Programming and development

- Red Hat Developers Blog (https://developers.redhat.com

- Try for free: Red Hat Learning Subscription (https://www.redhat.com/en/services/training/learning-subscription?src=programming_resource_menu3)
- New Python content (https://opensource.com/tags/python?src=programming_resource_menu1)
- Our latest JavaScript articles (https://opensource.com/tags/javascript?src=programming_resource_menu2)

In programming, these subroutines can be expressed as *functions*. Functions are important for programmers because they help reduce redundancy in code, which in turn reduces the amount of maintenance required. For example, in the imaginary scenario of baking bread programmatically, if you need to change the amount of time the dough proofs, as long as you've used a function before, you merely have to change the value of the seconds once, either by using a variable (called **SNOOZE** in the sample code) or directly in the subroutine that processes dough. That can save you a lot of time, because you don't have to hunt through your codebase for every possible mention of rising dough, much less worry about missing one. Many a bug's been caused by a missed value that didn't get changed or by a poorly executed `sed` command in hopes of catching every last match without having to hunt for them manually.

In Bash (https://opensource.com/resources/what-bash), defining a function is as easy as setting it either in the script file you're writing or in a separate file. If you save functions to a dedicated file, you can `source` it into your script as you would `include` a library in C or C++ or `import` a module into Python. To create a Bash function, use the keyword `function`:

Here's a simple (and somewhat contrived, as this could be made simpler) example of how a function works with arguments:

```
#!/usr/bin/env bash
ARG=$1

function mimic {
  if [[ -z $ARG ]]; then
    ARG='world'
  fi
  echo "hello $ARG"
}

mimic $ARG
```

Here are the results:

```
$ ./mimic
hello world
$ ./mimic everybody
hello everybody
```

Note the final line of the script, which executes the function. This is a common point of confusion for beginning scripters and programmers: functions don't get executed automatically. They exist as *potential* routines until they are called.

Without a line calling the function, the function would only be defined and would never run.

If you're new to Bash, try executing the sample script once with the last line

Functions are vital programming concepts, even for simple scripts. The more comfortable you become with functions, the better off you'll be when you're faced with a complex problem that needs something more dynamic than just declarative lines of commands. Keeping general-purpose functions in separate files can also save you some work, as it'll help you build up routines you commonly use so that you can reuse them across projects. Look at your scripting habits and see where functions might fit.

(/article/20/4/bash-it-out-book)

Learn Bash with this book of puzzles (/article/20/4/bash-it-out-book)

'Bash it out' covers basic, medium, and advanced Bash scripting using 16 puzzles.

Carlos Aguayo (/users/hwmaster1)

(/article/20/4/bash-programming-guide)

Get started with Bash programming (/article/20/4/bash-programming-guide)

Learn how to write custom programs in Bash to automate your repetitive tasks.

(/article/20/1/improve-bash-scripts)

5 ways to improve your Bash scripts (/article/20/1/improve-bash-scripts)

Find out how Bash can help you tackle the most challenging tasks.

Alan Formy-Duval (Correspondent) (/users/alanfdoss)

Topics : **Bash** (/tags/bash) **Command line** (/tags/command-line)

(/users/seth)

About the author

Seth Kenlon - Seth Kenlon is an independent multimedia artist, free culture advocate, and UNIX geek. He has worked in the film (http://www.imdb.com/name/nm1244992) and computing (http://people.redhat.com/skenlon) industry, often at the same time. He is one of the maintainers of the Slackware-based multimedia production project, http://slackermedia.info (http://slackermedia.info)

• More about me (/users/seth)

Recommended reading