Import functions and variables into Bash with the source command

Source is like a Python import or a Java include. Learn it to expand your Bash prowess.

12 Jun 2020 | Seth Kenlon (Red Hat) (/users/seth) | 61 | 2 comments

Image by : *Opensource.com*

When you log into a Linux shell, you inherit a specific working environment. An *environment*, in the context of a shell, means that there are certain variables already set for you, which ensures your commands work as intended. For instance, the PATH (https://opensource.com/article/17/6/set-path-linux) environment variable defines where your shell looks for commands. Without it, nearly everything you try to do in Bash would fail with a **command not found** error. Your environment, while mostly invisible to you as you go about your everyday tasks, is vitally important.

There are many ways to affect your shell environment. You can make modifications in configuration files, such as `~/.bashrc` and `~/.profile`, you can run services at startup, and you can create your own custom commands or script your own Bash functions (https://opensource.com/article/20/6/how-write-functions-bash).

Bash (along with some other shells) has a built-in command called **source**. And here's where it can get confusing: **source** performs the same function as the command . (yes, that's but a single dot), and it's *not* the same **source** as the **Tcl** command (which may come up on your screen if you type **man source**). The built-in **source** command isn't in your **PATH** at all, in fact. It's a command that comes included as a part of Bash, and to get further information about it, you can type **help source**.

The . command is POSIX (https://opensource.com/article/19/7/what-posix-richard-stallman-explains)-compliant. The **source** command is not defined by POSIX but is interchangeable with the . command.

More on Bash

- Bash cheat sheet (https://opensource.com/downloads/bash-cheat-sheet?intcmp=7013a000002CxqaAAC)
- An introduction to programming with Bash (https://opensource.com/downloads/bash-programming-guide?intcmp=7013a000002CxqaAAC)
- A sysadmin's guide to Bash scripting (https://opensource.com/downloads/bash-scripting-ebook?intcmp=7013a000002CxqaAAC)
- Latest Bash articles (https://opensource.com/tags/bash?intcmp=7013a000002CxqaAAC)

According to Bash **help**, the **source** command executes a file in your current shell. The clause "in your current shell" is significant, because it means it doesn't launch a sub-shell; therefore, whatever you execute with **source** happens within and affects your *current* environment.

```
#!/usr/bin/env bash
echo "hello world"
```

Using **source**, you can run this script even without setting the executable bit:

```
$ source hello.sh
hello world
```

You can also use the built-in **.** command for the same results:

```
$ . hello.sh
hello world
```

The **source** and **.** commands successfully execute the contents of the test file.

Set variables and import functions

You can use **source** to "import" a file into your shell environment, just as you might use the **include** keyword in C or C++ to reference a library or the **import** keyword in Python to bring in a module. This is one of the most common uses for **source**, and it's a common default inclusion in **.bashrc** files to **source** a file called **.bash_aliases** so that any custom aliases you define get imported into your environment when you log in.

Here's an example of importing a Bash function. First, create a function in a file called **myfunctions**. This prints your public IP address and your local IP address:

```
        ip addr | grep inet$IP | \
        cut -d"/" -f 1 | \
        grep -v 127\.0 | \
        grep -v \:\:1 | \
        awk '{$1=$1};1'
}
```

Import the function into your shell:

```
$ source myfunctions
```

Test your new function:

```
$ myip
93.184.216.34
inet 192.168.0.23
inet6 fbd4:e85f:49c:2121:ce12:ef79:0e77:59d1
inet 10.8.42.38
```

Search for source

When you use **source** in Bash, it searches your current directory for the file you reference. This doesn't happen in all shells, so check your documentation if you're not using Bash.

If Bash can't find the file to execute, it searches your **PATH** instead. Again, this isn't the default for all shells, so check your documentation if you're not using Bash.

These are both nice convenience features in Bash. This behavior is

where your functions are stored, because you know they're in your local equivalent of `/usr/include`, so no matter where you are when you source them, Bash finds them.

For instance, you could create a directory called `~/.local/include` as a storage area for common functions and then put this block of code into your `.bashrc` file:

```
for i in $HOME/.local/include/*;
  do source $i
done
```

This "imports" any file containing custom functions in `~/.local/include` into your shell environment.

Bash is the only shell that searches both the current directory and your `PATH` when you use either the `source` or the `.` command.

Using source for open source

Using `source` or `.` to execute files can be a convenient way to affect your environment while keeping your alterations modular. The next time you're thinking of copying and pasting big blocks of code into your `.bashrc` file, consider placing related functions or groups of aliases into dedicated files, and then use `source` to ingest them.

Get started with Bash scripting for sysadmins (/article/20/4/bash-sysadmins-ebook)

Learn the commands and features that make Bash one of the most powerful shells available.

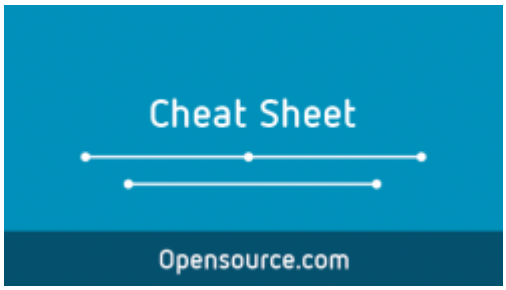
Seth Kenlon (Red Hat) (/users/seth)

(/article/19/12/automation-bash-scripts)

Introduction to automation with Bash scripts (/article/19/12/automation-bash-scripts)

In the first article in this four-part series, learn how to create a simple shell script and why they are the best way to automate tasks.

David Both (Correspondent) (/users/dboth)



(/article/19/11/bash-cheat-sheet)

Bash cheat sheet: Key combos and special syntax (/article/19/11/bash-cheat-sheet)

Download our new cheat sheet for Bash commands and shortcuts you need to talk to your computer.

Bash (/tags/bash) **Linux** (/tags/linux) **Command line** (/tags/command-line)

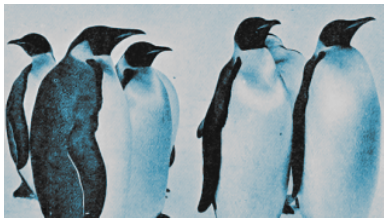
(/users/seth)

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



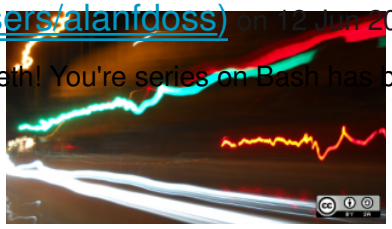
Manage your SSH connections with this open source tool (/article/20/9/ssh-connection-manager?utm_campaign=intrel)

A beginner's guide to SSH for remote connection on Linux (/article/20/9/ssh?utm_campaign=intrel)

Program hardware from the Linux command line (/article/20/9/hardware-command-line?utm_campaign=intr

Alan Formy-Duval (/users/alanfdoss) on 12 Jun 2020 1

This is a great article Seth! You're series on Bash has been very educational!



Aaron Cleaning Acton (/users/aar0nacton) on 25 Jun 2020 0

Great one! Thank you!

actical guide to learning awk (/article/20/9/awk-ebook?utm_campaign=intr20)

Open ports and route traffic through your firewall (/article/20/9/firewall?utm_campaign=intr20)

An introduction to using tcpdump at the Linux command line (/article/18/10/introduction-tcpdump?utm_campaig

(http://creativecommons.org/licenses/by-sa/4.0/)

Subscribe to our weekly newsletter

Enter your email address...

Select your country or region

Subscribe

Privacy Statement

Get the highlights in your inbox every week.

Find us: