

This is your **last** free member-only story this month. [Sign up for Medium and get an extra one](#)

# Automating Tasks with Shell Scripts



Damilare A. Adedoyin

Apr 3 · 4 min read ★

If you are as lazy as I am, you would agree that writing the same commands over and over could be quite exhausting. It would be really great to execute a series of commands with a single command. Kind of like executing the power of all 6 infinity stones at the snap of a finger.



Thanos and the infinity stones

. . .

I've been using a Unix based computer for about 7 months now, but I only got to know about this possibility recently. The aim of this post is to share how you can automate

your recurring CLI operations in your projects using shell scripts.

In this post, I'll be writing a script that would push a project folder to a remote repository. This script would initialize a git repository, add a Readme.md file, commit all changes and push to a remote repository, all in a single CLI operation.

Let's get right to it, shall we?

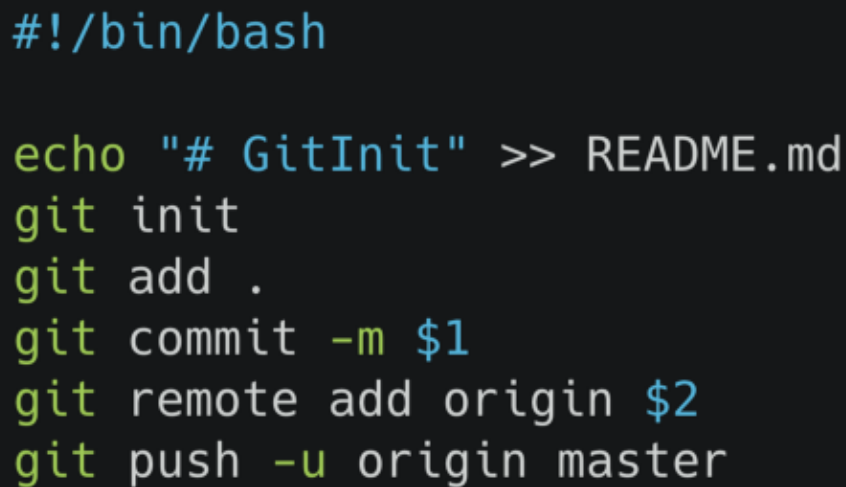


## Step 1

Open a new folder and create a Shell script file. I'll name mine `gitinit.sh`

## Step 2

Add the CLI set of commands



```
#!/bin/bash

echo "# GitInit" >> README.md
git init
git add .
git commit -m $1
git remote add origin $2
git push -u origin master
```

Basically, I did a lazy copy of the lines from new repositories on GitHub

## A few things to note

The `#!/bin/bash` line is called a *shebang*. A shebang basically is a line that tells the parent shell which interpreter to use in interpreting the script. You could also use the `#!/bin/sh` shebang, but to the best of my understanding `#!/bin/bash` is the standard.

Meanwhile, you may have noticed the `$1` and `$2` variables, you may also have an idea of what they represent. They are identifiers for the arguments that would be added when the script is called.

## Let's Execute the Script

The first thing we need to do is to make the script executable. This can be achieved by simply typing into your terminal the following command:

```
chmod +x gitinit.sh
```

Remember `gitinit.sh` is the name of my script. `chmod` basically changes the file mode of any given file, `+x` simply means execution privilege should be added to the file's permission.

To run the script, enter:

```
./gitinit.sh commit-message remote-repo-url
```

Make sure you're inside the same directory where the script file was created.

Here is my output

```
→ gitShell git:(master) ✗ chmod +x gitinit.sh
→ gitShell git:(master) ✗ ./gitinit.sh first-commit https://github.com/Damilare1/GitInit.git
Reinitialized existing Git repository in /Users/damilareadedoyin/Documents/shell/gitShell/.git/
[master (root-commit) a7a1442] first-commit
2 files changed, 10 insertions(+)
create mode 100644 README.md
create mode 100755 gitinit.sh
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 16 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 384 bytes | 384.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0)
To https://github.com/Damilare1/GitInit.git
* [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
→ gitShell git:(master) █
```

Terminal output after executing the script

**The next question is, does that mean I can only run this script inside its own directory?**

Short answer, yes.

If you're creating a script for automating tasks, you probably want it to be available globally such that it can be called from any directory on your computer.

The only way I know to do that is by adding the file to the `$PATH` environment. To view the files in your shell path, simply type `echo $PATH` into your terminal. This would show the available directories on the `PATH`.

```
→ gitShell git:(master) echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin
```

Here's mine

An executable script in any of these directories will be available globally.

## Let's go Global

A thought that would come to mind is simply copying and pasting the script into the directory. That would most likely work too, but I prefer using a *SYMLINK*. A SYMLINK is simply a shortcut to another file or folder. Basically, the file is accessible in the directory, but it isn't really in the directory.

Doing this, I'll be able to make changes to the script from the directory it was created without having to enter into the root directory. Pretty neat.

We go about this by typing:

```
ln -s path/to/folder/gitinit.sh /usr/local/bin/gitinit.sh
```

This is interpreted as, "create a shortcut for `gitinit.sh` at `/usr/local/bin`, where the original file is at `path/to/folder/gitinit.sh`".

**N.B:** You can get the current directory path using `pwd | pbcopy`. This automatically copies into your clipboard.

Then we add the execution privilege `chmod +x gitinit.sh` and call the script with its arguments.

For me, it looks like this:

```
→ gitShell git:(master) ✗ ln -s /Users/damilareadedoyin/Documents/shell/gitShell/gitinit.sh /usr/local/bin/gitinit.sh
→ gitShell git:(master) ✗ chmod +x gitinit.sh
→ gitShell git:(master) ✗ gitinit.sh second-commit https://github.com/Damilare1/GitInit.git
Reinitialized existing Git repository in /Users/damilareadedoyin/Documents/shell/gitShell/.git/
[master 64d80c0] second-commit
 2 files changed, 2 insertions(+)
 create mode 100644 .DS_Store
fatal: remote origin already exists.
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 16 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 569 bytes | 569.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0)
To https://github.com/Damilare1/GitInit.git
 a7a1442..64d80c0 master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
→ gitShell git:(master) █
```

Now we can access the script globally without having to enter the directory in which it was created.

## Removing the script from Global

Alright, it was all fun while it lasted, but it's time to say goodbye.

To remove the script, simply navigate to the root directory it was added to. Remember, we used a SymLink, so I'll just unlink the file in order to preserve the original file for future use. This is done, by:

```
cd /usr/local/bin  
unlink gitinit.sh
```

Voila, the script is no longer global. You might want to confirm by calling `gitinit.sh`.

## Conclusion

I hope you've been able to understand how to use shell scripts in automating CLI tasks. I'm also relatively new to the topic, but if you have any questions or hitches along the way, feel free to drop a comment. Let's learn together.

## Further reading

[Learn Shell](#)

[Shell Scripts](#)

[Shell scripting tutorials by Guru99](#)

[Removing Symlink](#)

[Shell](#)[Shell Script](#)[Shellscripting](#)[Developer](#)[Unix](#)[About](#) [Help](#) [Legal](#)

Get the Medium app

