

In Part 1 of this two-part GitHub tutorial, we examined the main uses for GitHub and began the process of signing up for a GitHub account and creating our own local repository for code.

Now that

these

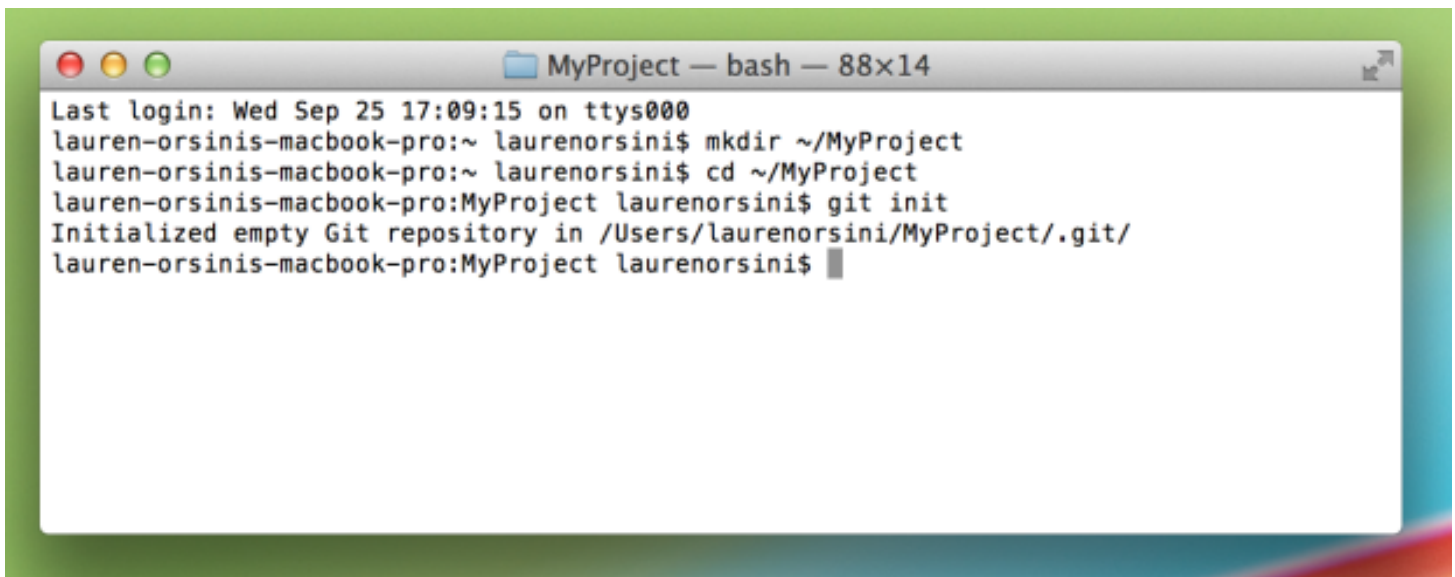
steps

have

been

See also [GitHub For Beginners: Don't Get Scared, Get Started](#)

accomplished, let's add the first part of your project now by making your first commit to GitHub. When we last left off, we'd created a local repository called MyProject, which, when viewed in the command line, looks like this screenshot.

A screenshot of a macOS terminal window titled "MyProject — bash — 88x14". The terminal shows the following commands and output:

```
Last login: Wed Sep 25 17:09:15 on ttys000
lauren-orsinis-macbook-pro:~ laurenorsini$ mkdir ~/MyProject
lauren-orsinis-macbook-pro:~ laurenorsini$ cd ~/MyProject
lauren-orsinis-macbook-pro:MyProject laurenorsini$ git init
Initialized empty Git repository in /Users/laurenorsini/MyProject/.git/
lauren-orsinis-macbook-pro:MyProject laurenorsini$
```

Local repo as viewed from Terminal.

On your next line, type:

```
touch Readme.txt
```

This, again, is not a Git command. It's another standard navigational command prompt.

`touch`

really means “create.” Whatever you write after that is the name of the thing created. If you go to your folder using Finder or the Start menu, you’ll see an empty `Readme.txt` file is now inside. You could have also made something like “`Readme.doc`” or “`Kiwi.gif`,” just for kicks.

You can clearly see your new `Readme` file. But can Git? Let’s find out. Type:

`git status`

The command line, usually so passive up to this point, will reply with a few lines of text similar to this:

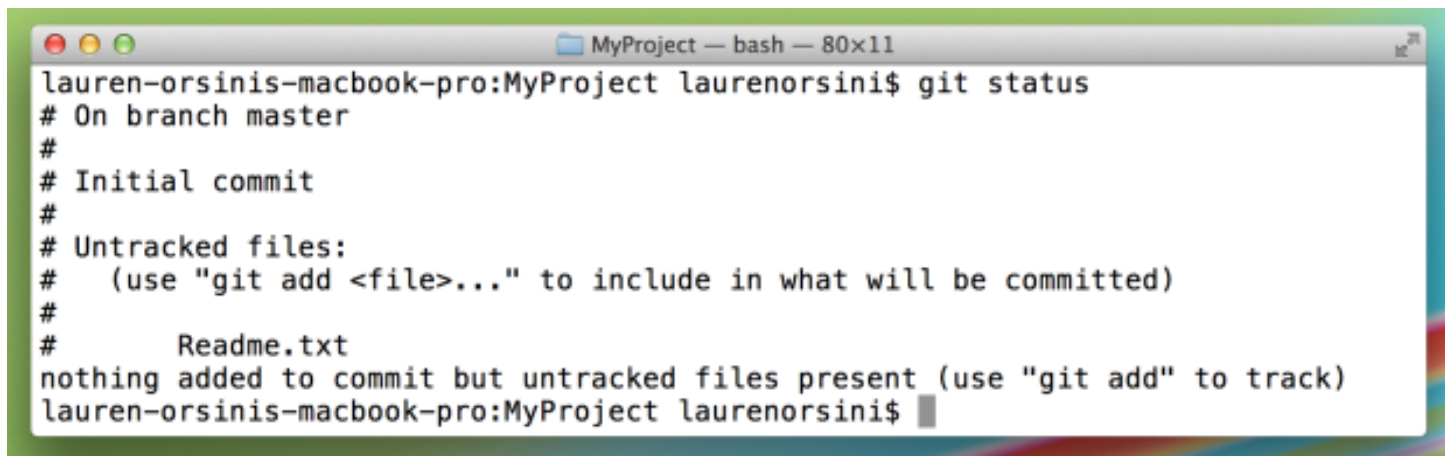
On branch master

Untracked files:

(use "git add ..." to include in what will be committed)

#

`Readme.txt`

A screenshot of a terminal window titled "MyProject — bash — 80x11". The terminal shows the output of the command "git status". The output is as follows:

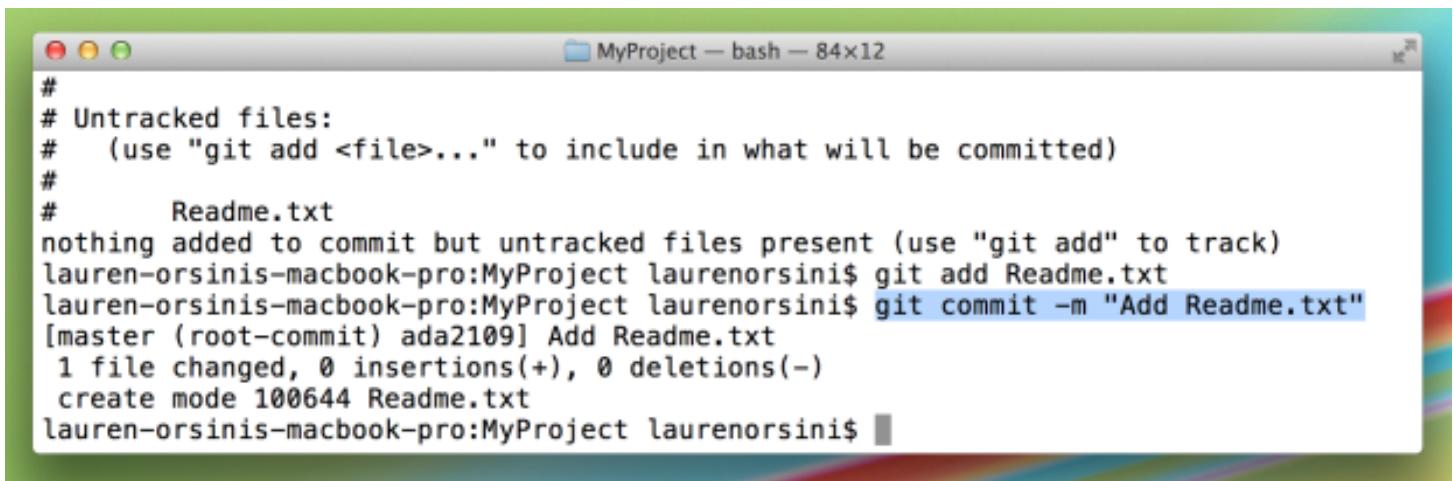
```
lauren-orsinis-macbook-pro:MyProject laurenorsini$ git status
# On branch master
#
# Initial commit
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       Readme.txt
nothing added to commit but untracked files present (use "git add" to track)
lauren-orsinis-macbook-pro:MyProject laurenorsini$
```

What's going on? First of all, you're on the master branch of your project, which makes sense since we haven't "branched off" of it. There's no reason to, since we're working alone. Secondly, Readme.txt is listed as an "untracked" file, which means Git is ignoring it for now. To make Git notice that the file is there, type:

```
git add Readme.txt
```

Notice how the command line gave you a hint there? All right, we've added our first file, so it's time to take a "snapshot" of the project so far, or "commit" it:

```
git commit -m "Add Readme.txt"
```



```
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       README.txt
nothing added to commit but untracked files present (use "git add" to track)
lauren-orsinis-macbook-pro:MyProject laurenorsini$ git add README.txt
lauren-orsinis-macbook-pro:MyProject laurenorsini$ git commit -m "Add README.txt"
[master (root-commit) ada2109] Add README.txt
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 README.txt
lauren-orsinis-macbook-pro:MyProject laurenorsini$
```

The highlighted text is our first commit.

The

-m

flag, as noted in the terms directory in **Part 1**, simply indicates that the following text should be read as a message. Notice the commit message is written in present tense. You should always write your commands in present tense because version control is all about flexibility through time. You're not writing about what a commit *did*, because you may always revert to earlier. You're writing about what a commit *does*.

Now that we've done a little work locally, it's time to "push" our first commit up to GitHub.

"Wait, we never connected my online repository to my local repository," you might be thinking. And you're right. In fact, your local repository and your online one are only connecting for short bursts, when you're confirming project additions and changes. Let's move on to making your first real connection now.

Connect Your Local Repository To Your

GitHub Repository

Having a local repository as well as a remote (online) repository is the best of both worlds. You can tinker all you like without even being connected to the Internet, and at the same time showcase your finished work on GitHub for all to see.

This setup also makes it easy to have multiple collaborators working on the same project. Each of you can work alone on your own computers, but upload or “push” your changes up to the GitHub repository when they’re ready. So let’s get cracking.

First, we need to tell Git that a remote repository actually exists somewhere online. We do this by adding it to Git’s knowledge. Just like Git didn’t acknowledge our files until we used the

```
git add
```

command, it won’t acknowledge our remote repo yet, either.

Assume that we have a GitHub repo called “MyProject” located at <https://github.com/username/myproject.git>. Of course,

```
username
```

should be replaced with whatever your GitHub username actually is, and

```
myproject
```

should be replaced with the actual title you named your first GitHub repository.

```
git remote add origin https://github.com/username/myproject.git
```

The first part is familiar; we’ve used

```
git add
```

already with files. We've tacked the word

`origin`

onto it to indicate a new place from which files will originate.

`remote`

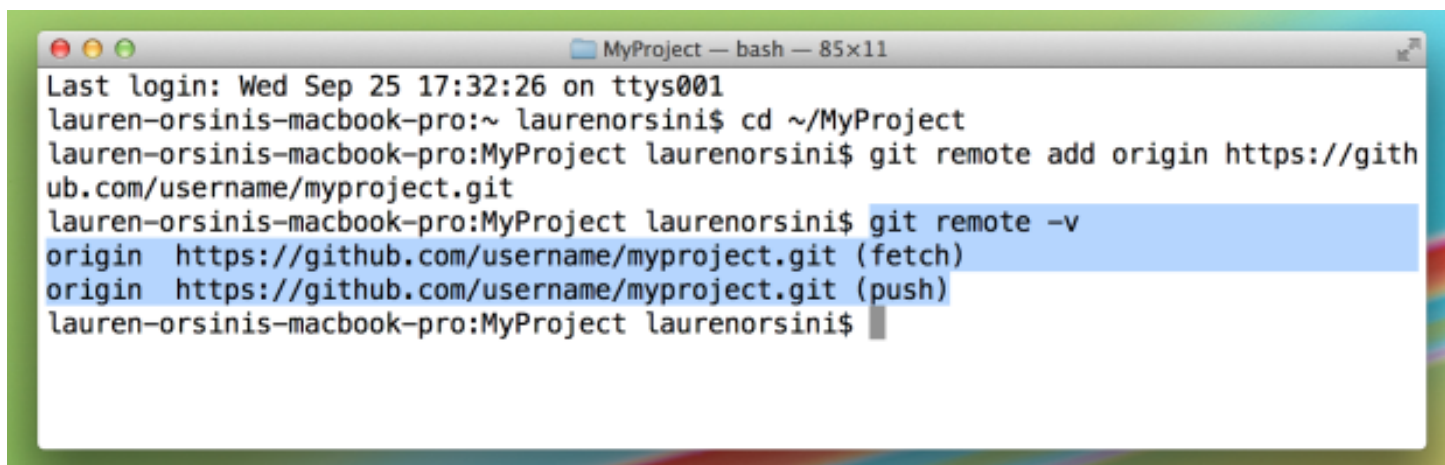
is a descriptor of

`origin`

, to indicate the origin is *not* on the computer, but somewhere online.

Git now knows there's a remote repository and it's where you want your local repository changes to go. To confirm, type this to check:

```
git remote -v
```

A screenshot of a terminal window titled "MyProject — bash — 85x11". The terminal shows the following commands and output:

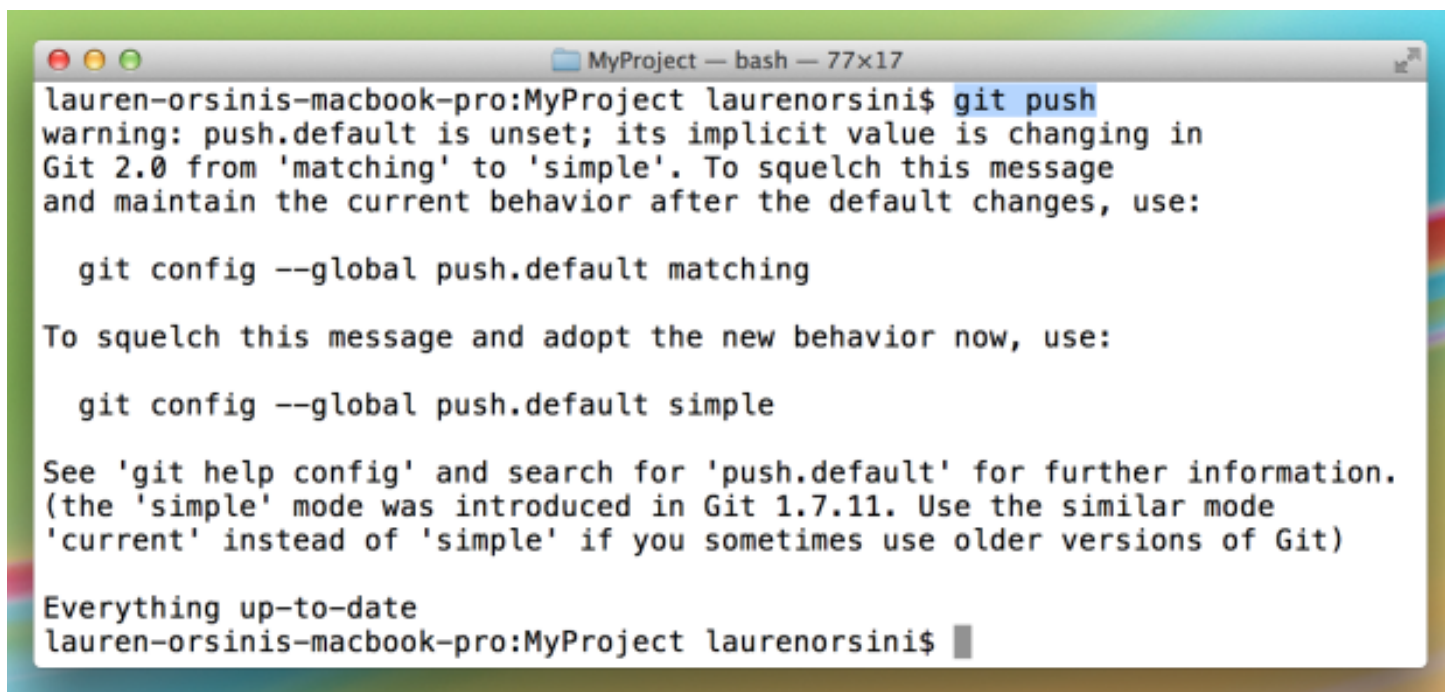
```
Last login: Wed Sep 25 17:32:26 on ttys001
lauren-orsinis-macbook-pro:~ laurenorsini$ cd ~/MyProject
lauren-orsinis-macbook-pro:MyProject laurenorsini$ git remote add origin https://github.com/username/myproject.git
lauren-orsinis-macbook-pro:MyProject laurenorsini$ git remote -v
origin https://github.com/username/myproject.git (fetch)
origin https://github.com/username/myproject.git (push)
lauren-orsinis-macbook-pro:MyProject laurenorsini$
```

This command gives you a list of all the remote origins your local repository knows about. Assuming you've been with me so far, there should only be one, the `myproject.git` one we just added. It's listed twice, which means it is available to *push* information to, and to *fetch* information from.

Now we want to upload, or “push,” our changes up to the GitHub remote repo. That's easy. Just type:

git push

The command line will chug through several lines on its own, and the final word it spits out will most likely be “everything up-to-date.”

A screenshot of a terminal window titled "MyProject — bash — 77x17". The prompt is "lauren-orsinis-macbook-pro:MyProject laurenorsini\$". The user has entered "git push". The output shows a warning about the push.default configuration changing from 'matching' to 'simple' in Git 2.0. It provides instructions to use "git config --global push.default matching" to keep the old behavior or "git config --global push.default simple" to adopt the new behavior. It also mentions that 'simple' was introduced in Git 1.7.11 and that 'current' can be used instead of 'simple' in older versions. Finally, it says "Everything up-to-date" and returns the prompt "lauren-orsinis-macbook-pro:MyProject laurenorsini\$".

```
lauren-orsinis-macbook-pro:MyProject laurenorsini$ git push
warning: push.default is unset; its implicit value is changing in
Git 2.0 from 'matching' to 'simple'. To squelch this message
and maintain the current behavior after the default changes, use:

    git config --global push.default matching

To squelch this message and adopt the new behavior now, use:

    git config --global push.default simple

See 'git help config' and search for 'push.default' for further information.
(the 'simple' mode was introduced in Git 1.7.11. Use the similar mode
'current' instead of 'simple' if you sometimes use older versions of Git)

Everything up-to-date
lauren-orsinis-macbook-pro:MyProject laurenorsini$
```

Git's giving me a bunch of warnings here since I just did the simple command. If I wanted to be more specific, I could have typed

```
git push origin master
```

, to specify that I meant the master branch of my repository. I didn't do that because I only have one branch right now.

Log into GitHub again. You'll notice that GitHub is now tracking how many commits you've made today. If you've just been following this tutorial, that should be exactly one. Click on your repository, and it will have an identical Readme.txt file as we earlier built into your local repository.

All Together Now!

Congratulations, you are officially a Git user! You can create repos and commit changes with the best of them. This is where most beginner tutorials stop.

Advertisement — Continue reading below

However,
you may
have this
nagging
feeling
that you

See also: Github's Tom Preston-Werner: How We Went Mainstream

still don't *feel* like an expert. Sure you managed to follow through a few steps, but are you ready to be out on your own? I certainly didn't.

In order to get more comfortable with Git, let's walk through a fictional workflow while using a little of everything we've already learned. You are now a worker at 123 Web Design, where you're building a new website for Jimmy's Ice Cream Shop along with a few of your coworkers.

You were a little nervous when your boss told you that you'd be participating in the Jimmy's Ice Cream Shop webpage redesign project. After all, you're not a programmer; you're a graphic designer. But your boss assured you that anyone

can use Git.

You've created a new illustrations of an ice cream sundae, and it's time to add it to the project. You've saved them in a folder on your computer that is also called "icecream" to prevent yourself from getting confused.

Open up the Command Line and change directory until you're inside the icecream folder, where your designs are stored.

```
cd ~/icecream
```

Next, initialize Git so you can start using Git commands inside the folder. The folder is now a Git repository.

```
git init
```

Wait, this is the right folder, right? Here's how you check and make sure this is where you stored your design:

```
git status
```

And this is what Git will tell you in reply:

```
# Untracked files:
#   (use "git add ..." to include in what will be committed)
#
#       chocolate.jpeg
```

There they are! Add them to your local Git repository so they'll be tracked by Git.

```
git add chocolate.jpeg
```

Now, take a “snapshot” of the repository as it stands now with the commit command:

```
git commit -m “Add chocolate.jpeg.”
```

Great! But your co-workers, hard at work in their own local repositories, can't see your fantastic new design. That's because the main project is stored in the company GitHub account (username: 123WebDesign) in the repository called “icecream.”

Since you haven't connected to the GitHub repo yet, your computer doesn't even know this exists. So tell your local repository about it:

```
git remote add origin  
https://github.com/123WebDesign/icecream.git
```

And double check to make sure it knows:

```
git remote -v
```

Finally, it's the moment you've been waiting for. Upload that delicious looking sundae up to the project:

```
git push
```

Ta da! With all of these tool at hand, it's clear that Git and the GitHub service aren't just for programmers.