readwrite

AI

Blockchain

Connected
Devices

Fintech

Industrial

Lifestyle

Smart Cities

Transport

More

Search …

GitHub For Beginners: Commit, Push And Go

lauren orsini / 02 Oct 2013 / Code

readwrite

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.

> **See also GitHub For Beginners: Don't Get Scared, Get Started**

Now that these steps have been 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.

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
```

AI

Blockchain

Connected
Devices

Fintech

Industrial

Lifestyle

Smart Cities

Transport

More

Search …

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
```
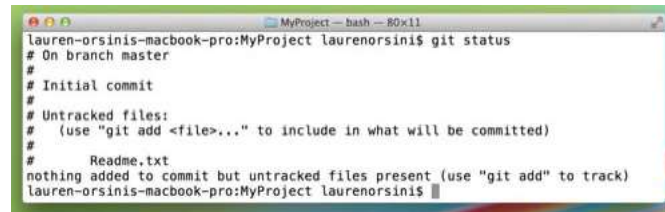

What's going on? First of all, you're on the master branch of your project, which makes sense since we haven't "branched off" of it. There's no reason to, since we're working alone. Secondly, Readme.txt is listed as an "untracked" file, which means Git is ignoring it for now. To make Git notice that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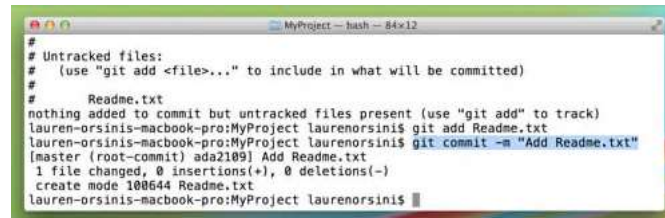

The highlighted text is our first commit.

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.
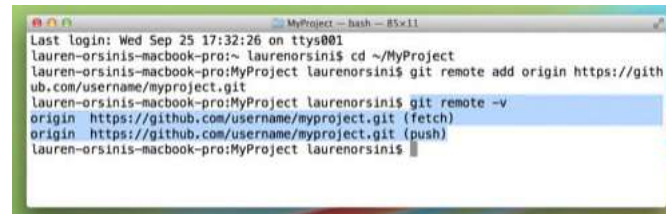
Assume that we have a GitHub repo called "MyProject" located at https://github.com/username/myproject.git. Of course, `username` should be replaced with whatever your GitHub username actually is, and `myproject` should be replaced with the actual title you named your first GitHub repository.

```
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:
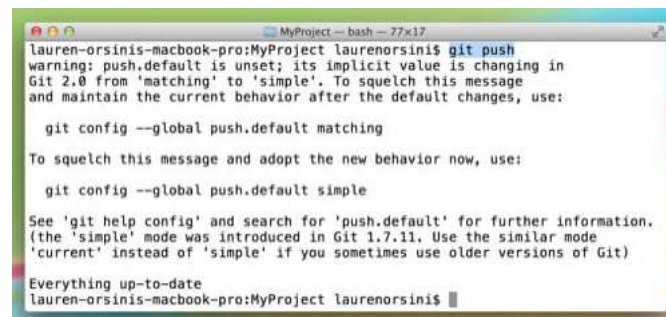
```
git remote -v
```


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."

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.

> **See also: [Github's Tom Preston-Werner: How We Went Mainstream](#)**

However, you may have this nagging feeling that you 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.

Git Resources

CodeSchool's Try Git.

Git is dense, I know. I did my best to make a tutorial that could even teach me how to use it, but we don't all learn in the same ways. Here are some resources I found useful while teaching myself how to use Git and GitHub over the summer:

- Pro Git. Here's an entire open source book on learning and using Git. It looks like a long one, but I didn't need to read anything past chapter three just to learn the basics.

- Try Git. CodeSchool and GitHub teamed up to make this speedy tutorial. If you want a little more practice with the basics, this should help. And if you have some extra money and want to learn everything there is to know about Git, Code School's Git Real should do the trick.

- GitHub Guides. If you're a visual learner, GitHub's official YouTube channel is worth your time. I especially got a lot out of the Git Basics four-part series.

- Git Reference. Got the basics down but find yourself always forgetting the commands? This handy site is great as a glossary reference

- Git – the simple guide. This tutorial is short and sweet, but it was a little too fast for me as a beginner. If you want to refresh on the basics of Git, this should be all you need.

Tags: #GitHub   #How To   #version control

Lauren Orsini

subscriber

Related Posts

The Problem With Believing Coding Is No Longer …

A primer on data encryption

Are Downloads Driving Your Revenue?

A Failure To Communicate In The World Of IoT, P…

readwrite

About   Advertise   Contact   Events   Privacy   Guest Post   Subscribe

Privacy